AFRL-RI-RS-TR-2015-226

TEAM VIGIR: DARPA ROBOTICS CHALLENGE

TORC ROBOTICS, LLC

OCTOBER 2015

FINAL TECHNICAL REPORT

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

■ AIR FORCE MATERIEL COMMAND ■ UNITED STATES AIR FORCE ■ ROME, NY 13441

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2015-226 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
ROGER J. DZIEGIEL, JR.
Work Unit Manager

/ S /
MICHAEL J. WESSING
Deputy Chief, Information Intelligence
Systems & Analysis Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* OCTOBER 2015	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* OCT 2012 – AUG 2015

4. TITLE AND SUBTITLE TEAM VIGIR: DARPA ROBOTICS CHALLENGE	**5a. CONTRACT NUMBER** FA8750-12-C-0337
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 62702E
6. AUTHOR(S) David Conner, S. Kohlbrecher, A. Romay, A. Stumpf, S. Maniatopoulos , M. Schappler, and B. Waxler	**5d. PROJECT NUMBER** ROBO
	5e. TASK NUMBER PR
	5f. WORK UNIT NUMBER 01
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) TORC Robotics, LLC 405 Partnership Drive Blacksburg, VA 24060	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RIED 525 Brooks Road Rome NY 13441-4505 / DARPA 675 North Randolph St Arlington, VA 22203-2114	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RI
	11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2015-226

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report documents Team ViGIR's efforts in the DARPA Robotics Challenge (DRC) between October 2012 and August 2015. Team ViGIR, a multinational collaborative research and development effort that spanned nine time zones, began as a Track B participant in the simulation-based Virtual Robotics Challenge; after placing in the top six, we began working the Atlas humanoid robotic system developed by Boston Dynamics. Team ViGIR competed in both the DRC Trials and DRC Finals. This report documents our performance, lessons learned along the way, and describes the novel contributions of our team. Specific focus areas include template-based manipulation, footstep planning, and autonomous behavior specification and execution. The software used in the competition and described in this report is being open sourced at http://github.com/team-vigir as part of our commitment to improving the capabilities of humanitarian rescue robotics.

15. SUBJECT TERMS
Robotics, Mobility, Platform Dexterity, Supervised Autonomy, Wireless, Ground

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** ROGER J. DZIEGIEL JR.
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	238	**19b. TELEPHONE NUMBER** *(Include area code)* N/A

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ACKNOWLEDGEMENTS

This report represents the combined efforts of our team, which spreads across two continents with a nine-hour time difference between Germany and Oregon. The major contributors include:

TORC Robotics, Inc
David C. Conner, PhD, Principal Investigator and Team Lead
Ben Waxler, Software Engineer

Technische Universität Darmstadt
Oskar von Stryk, PhD, Principal Investigator
Stefan Kohlbrecher, Onboard Software Lead, PhD Candidate
Alberto Isay Romay Tovar, PhD Candidate
Alex Stumpf, PhD Candidate
Philipp Schillinger, Masters Student

Virginia Tech
Doug Bowman, PhD, Principal Investigator
Felipe Bacim, Operator Control Station Lead, PhD Candidate

Oregon State University
Ravi Balasubramanian, PhD, Principal Investigator
Alex Goins, Masters Student (Phase 1B)
Jackson Carter, Undergraduate (Phase 2)

Cornell University (Phase 2)
Hadas Kress-Gazit, PhD, Principal Investigator
Spyros Maniatopoulos, PhD Candidate

Gottfried Wilhelm Leibniz Universität Hannover (Phase 2)
Sami Haddadin, PhD, Principal Investigator
Moritz Schappler, PhD Candidate

In addition to those listed here, this report builds upon and ties together a number of academic papers published in conference proceedings and journals. These papers are included in the appendices with each contributor acknowledged therein.

1. Summary

In spring 2012, the Defense Advanced Research Projects Agency (DARPA) announced the DARPA Robotics Challenge (DRC). In response, TORC Robotics, Inc (TORC®) led the proposal effort and gathered expertise from across the globe. Team ViGIR – the ***V***irginia-***G***ermany ***I***nterdisciplinary ***R***obotics team – was named in recognition of its original members. TORC Robotics (Blacksburg, VA) led the team and worked on the robot control interface, communications, and behaviors. Researchers at TU Darmstadt led development of the onboard software, including perception, behaviors, and motion planning. Researchers at Virginia Tech led the Operator Control Station (OCS) development. The initial proposal identified researchers from Cornell University and Oregon State University as future contributors.

DARPA selected Team ViGIR as one of 11 funded Track B participants. Team ViGIR – at this point consisting of TORC, TU Darmstadt, and Virginia Tech – attended the project kickoff in October 2012, and began work on developing our software for the DARPA Virtual Robotics Challenge (VRC) held in June 2013. Team ViGIR developed its software in parallel with both the simulation system and the Atlas robot design. In addition to the funded Track B teams, another 115 teams registered as unfunded Track C teams; 26 teams passed the initial qualification tests. Team ViGIR finished with 27 points, which placed sixth out of 22 teams that actually scored points in the VRC.

The VRC results qualified Team ViGIR to receive an Atlas robot built by Boston Dynamics, Inc (BDI). Team ViGIR attended robot training in July 2013, and began set up of their lab. Researchers from Oregon State University joined Team ViGIR at this time, and focused on the hand control and grasping interface. The team modified their VRC software base to accommodate changes to the robot design and software interface. After receiving their Atlas robot on August 27, 2013, the team began intensive experiments and preparation for the December 2013 Trials. At the trials, Team ViGIR scored eight points, which tied them for ninth place. Figure 1 shows Florian, named for the German patron saint of first responders[1], attempting to attach the hose after scoring two points in the hose task. A detailed system overview paper, which discussed the results of the DRC Trials, was published in [1] (Appendix A).

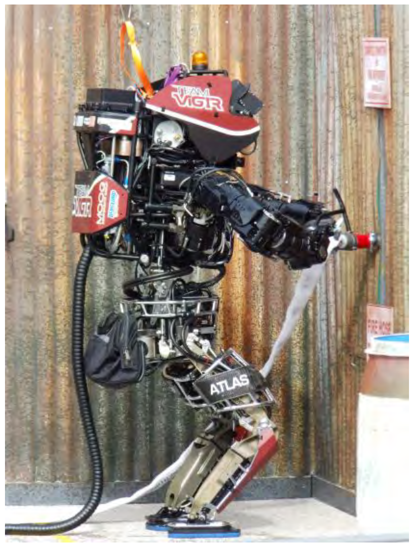


Figure 1. Florian, Team ViGIR's Atlas Robot participating in the 2013 DRC Trials.

Initially, this score missed the cut off for continued participation in the DRC. After Team Schaft dropped out of the competition, DARPA extended partial funding and the invitation to continue for the three ninth place teams. Initially, Team ViGIR defined a streamlined participation plan based on limited funding, but after DARPA provided additional funds in the fall of 2014 and moved the Finals to June 2015, Team ViGIR added Cornell University as originally planned. Additionally, the team added researchers from University of Hannover (Germany) with expertise in system identification and controls.

[1] http://www.publicsafety.net/st_florian.htm (accessed July 30, 2015)

During preparation for the DRC Finals, BDI took possession of the robot for three months to perform the upgrade to the new untethered “Atlas Unplugged” upgrade. BDI delivered the partially upgraded robot on February 21, 2015, about six weeks after the initial plan; Team ViGIR began work with the upgraded robot, but did not receive the upgraded arms until March 24, 2015. Team ViGIR worked through several hardware issues during the spring, and continued to test and refine their software up until they departed for the DRC Finals on May 29, 2015.

Team ViGIR competed in the DRC Finals on June 5-6, 2105 in Pomona, California. On Day 1, the team scored 3 points, and were stopped just shy of achieving the fourth point as our sixty-minute time limit expired. The robot worked well, but the team experienced unexpected communication issues during the run. The operators adapted, but were slower than expected due to software issues caused by a backlog in communications between the robot and field computer. The team adjusted the software, and were cautiously optimistic that they would be able to score 5 or 6 points on Day 2. Unfortunately, a series of hardware issues caused numerous problems on Day 2. In the end, the team earned only 2 points on Day 2, and ended the competition with a disappointing 3 points.

In the months after the competition, Team ViGIR worked to prepare their software for release as open source, and conducted experiments on several advanced features that were not ready in time for the DRC Finals.

This report discusses the results of each phase, the developed software architecture, experimental results, and the status of the software release. The report focuses on Team ViGIR’s specific areas of emphasis and innovation. The report presents future directions for our ongoing research, and concludes with a discussion of the lessons learned. Appendices provide technical details, and describe the software being released as part of our open source effort.

2. INTRODUCTION

This section provides a brief overview of the DRC and an introduction to the members of Team ViGIR. The section then provides an overview of the competition results for Team ViGIR during each phase, and focuses on the programmatic elements of the contract. The section concludes with an overview of the remaining sections of this report, which cover the technical details of our approach.

2.1. Project

In the spring of 2012, the DARPA proposed the DRC to accelerate the development and evaluation of disaster response robots that have the capability for early response and mitigation of disasters. This effort was partly motivated by the earthquake and tsunami that struck the Tohoku region of eastern Japan on March 11, 2011, and led to subsequent damage to the Fukushima Daiichi nuclear plant. The DRC concept was designed to mimic the conceptual tasks that might be required of a robot to respond to the initial damage and avert subsequent catastrophes.

DARPA structured the DRC as four separate funding tracks:

- Track A – DARPA funded teams develop hardware of their own design and software,
- Track B – DARPA funded competitors in the VRC (Simulation Challenge); winners get Government Furnished equipment (GFE) in the form of the Atlas robot developed by Boston Dynamics
- Track C – Self funded competitors in Virtual (Simulation) Challenge that will be eligible for DARPA funding and GFE Atlas after VRC
- Track D – Self funded competitors that develop hardware of their own design and software.

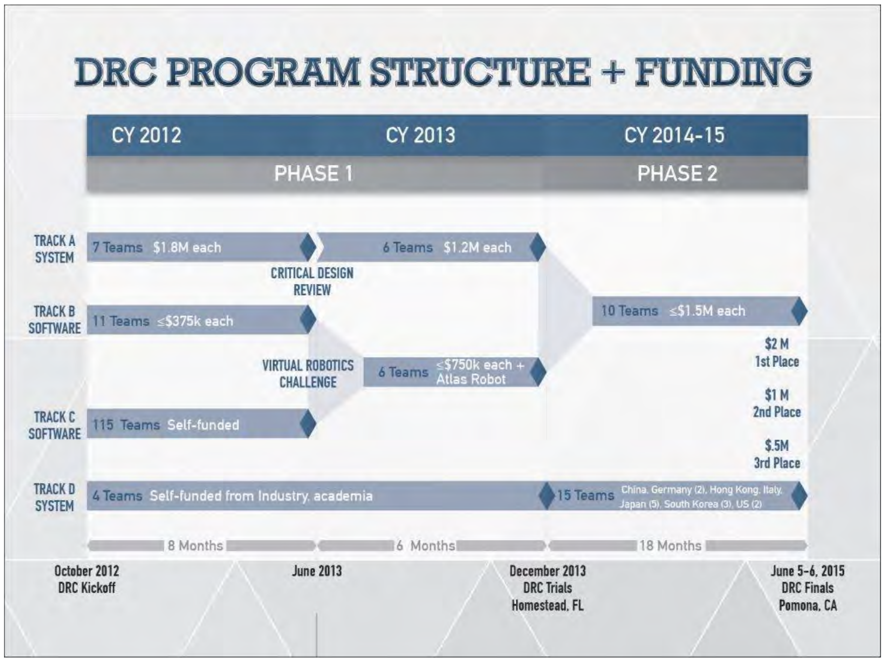


Figure 2. DARPA DRC Structure (courtesy DARPA)

Figure 2 shows the structure and funding levels, along with the final numbers of competitors in each track. Team ViGIR competed as a Track B team in the Virtual Robotics Challenge.

2.2. Team

Team ViGIR[2] – the ***V***irginia-***G***ermany ***I***nterdisciplinary ***R***obotics team – was named in recognition of its original members. The following section provides an overview of team members, and their primary responsibilities. During design and development, the software was conceptually divided into OCS software that interfaced with the human operators, and Onboard software that ran on the robot or field computers. Communications between the OCS and Onboard software was through a degraded communications link. In general, all team members had access to all software, and various members contributed to different components at different stages.

TORC Robotics, Inc (Blacksburg, VA, USA) TORC served as project management, provided technical leadership, and hosted the robot test lab in Blacksburg, VA. TORC (http://www.torcrobotics.com), the primary software developer for Team *V*ictor*T*ango in the 2007 DARPA Urban Challenge, is a leading provider of unmanned and autonomous ground vehicle solutions for the defense, agricultural, automotive, and mining industries. Team *V*ictor*T*ango finished in 3rd place, and was one of only three teams to finish the course without penalty. TORC components and systems have been integrated on over 100 unmanned and autonomous ground vehicle platforms ranging in size from 5 pounds to 240 tons. TORC's robotic components and systems provide customers with rapid solutions by leveraging proven technology to ensure customer success.

TORC personnel were the primary developers of the robot software interface and communication systems used throughout the DRC. TORC provided machine shop access and technician support as needed.

Technische Universität Darmstadt (Darmstadt, Germany) TU Darmstadt, and specifically the Simulation, Systems Optimization and Robotics Group at the Department of Computer Science (https://www.sim.informatik.tu-darmstadt.de/en/), served as the Onboard software lead. TU Darmstadt is one of the leading public engineering research universities in Germany. They conduct research in autonomous robot teams, bio-inspired robots and dynamic modeling and optimization methods. The research results have been honored, among others, with the 1st prize of the EURON/EUROP European Robotics Technology Transfer Award, the Louis Vuitton Best Humanoid Award, and several world championship titles for autonomous humanoid and four-legged robot soccer teams in the highly competitive annual RoboCup competitions. As four-time winners of the Best in Class Autonomy Award in the RoboCup Rescue League, they have provided open-source navigation software that has been reused and adopted by numerous international research groups.

Due to the international character of the group, TORC Robotics could not use its intellectual property; therefore the decision was made to use the ROS system for middleware and base capabilities. TU Darmstadt brought significant experience with ROS to the team.

Virginia Tech (Blacksburg, VA) Virginia Tech, specifically the 3D Interaction lab (http://www.hci.vt.edu/) at the Center for Human-Computer Interaction (CHCI) in the Department of Computer Science, served as OCS lead. CHCI is a world-class interdisciplinary research center at Virginia Tech, exploring the design of technological artifacts to support human activity and the impact of interactive technologies on the user experience. Housed in the Department of Computer Science, CHCI has 29 faculty affiliates across the university, including internationally recognized leaders in areas such as

[2] http://www.teamvigir.org (accessed July 30, 2015)

virtual and augmented reality, information visualization, gestural interaction, graphics and animation, creativity and the arts, and social collaborative computing. CHCI members lend their skills in user interface design, user experience evaluation, and usability engineering to projects in a broad range of application domains.

These three groups – TORC, TU Darmstadt, and Virginia Tech – formed the core of Team ViGIR, and worked together from the beginning of the DRC. After the initial success in the VRC, researchers from Oregon State University joined Team ViGIR full time. The initial proposal called for Cornell University to join at this time; however, after reviewing the then current state of the software, additional costs, and timeline before the trials, TORC in consultation with Cornell decided that Cornell would wait until after the DRC Trials to join.

Oregon State University (Corvallis, OR) Oregon State University, specifically the Robotics and Human Control Systems Lab (http://mime.oregonstate.edu/research/rhcs/), focused on grasping and manipulation, with a specific emphasis on testing and interfacing the robotic hands provided for the Atlas robot. The Robotics and Human Control Systems Lab has two goals: 1) To develop a deeper understanding of the neural control and biomechanics in the human body using robotics techniques, and 2) To develop the design and control methodologies (including human-inspired) that enable robots to operate robustly in unstructured environments. Application areas include robotic grasping and manipulation, mobile robotics, human-robot interaction, and rehabilitation.

Cornell University (Ithaca, NY) Cornell University, specifically the Verifiable Robotics Research Group (http://verifiablerobotics.com/), focused on the automatic synthesis of high-level behaviors and the manual development of autonomous behaviors for the team. The Verifiable Robotics Research Group conducts cutting edge research on high-level, verifiable robotics; the group develops theory, algorithms and tools that allow people to interact with robots at a high-level using language while providing guarantees for the robots' behavior.

These five – TORC, TU Darmstadt, and Virginia Tech, Oregon State, and Cornell – were the original members of Team ViGIR as defined in the original proposal. With the extended budget, Team ViGIR decided to enhance our controls experience and recruited another German research group to develop compliant impedance controllers for whole-body control of the robot, and then focus on getting up and vehicle egress behaviors.

Leibniz Universität Hannover (Hannover, Germany) Hannover, specifically the Institute for Automatic Control (IRT) (http://www.uni-hannover.de/en/), joined our group to focus on system identification and compliant manipulation. The Leibniz University Hanover is among the nine largest technical universities in Germany ("TU9"). The Institute for Automatic Control (IRT), aims to advance the scientific and technological foundations for intelligent and autonomous robots capable of interaction with their environment. IRT developed the first German dynamical walking bipedal robot. Recent focus of the institute is laid on soft-robotics mechatronics and control, physical human-robot interaction, machine learning and optimal control, and human motor control. IRT has been awarded numerous scientific awards, including several best paper awards at ICRA, IROS, and Transactions on Robotics.

2.3. DARPA Robotics Challenge Phases

This section provides a brief historical overview of the different phases of the competition, and describes Team ViGIR's performance in each competition phase.

2.3.1. Phase 1A- Virtual Robotics Challenge

Representatives from TORC Robotics, TU Darmstadt, and Virginia Tech attended the project kickoff October 24-25, 2012 in Arlington, VA. Immediately following the kickoff , the team worked to define the basic software architecture and support infrastructure. The team arranged for a computer donation of five industrial Intel Core i7 machines with NVidia graphics cards from Foxguard Solutions[3].

Given the multinational character of the team, TORC was unable to contribute existing IP to the project; therefore, we chose to base our software on the open source Robot Operating System (ROS) software[4], including the ROS 3D visualization tool rviz[5]. The team chose to make extensive use of existing ROS-integrated tools such as MoveIt![6] And Point Cloud Library (PCL)[7] as the base for algorithm development. To facilitate collaboration, TORC hosted an external wiki-based project-planning site using the Redmine[8] framework, and a GitLab[9]-based software repository. All team members had full access to the sites.

As the robot design and software interface was under development, Team ViGIR developed contingency plans for developing basic stability and walking algorithms, but initially focused on perception, planning, and operator interfaces under the constrained communications. Once it was confirmed that BDI would provide basic walking and stability control for the simulated robot, the team was able to continue its focus on the basic system. From the outset, Team ViGIR planned for a comprehensive approach to operator interaction with the robot, and avoided scripted behaviors that were finely tuned for the simulation tasks; while this may have been better suited for the defined structure of the virtual competition, it would have been impractical for realistic scenarios. Figure 3 shows an operator at the OCS during the VRC.

Contrary to our expectations, it quickly became apparent that both the robot and simulation engine were still under development, and in fact being developed in parallel with limited data sharing. Where we expected to receive a well-defined Application Programming Interface (API) at the kickoff, the initial version was not delivered until December 2012. The Open Source Robotics Foundation (OSRF)[10] did not release the initial API version that supported walking and balancing until Gazebo drcsim 2.2.0 was released on March 11, 2013, only three months before competition. This required unexpected work on our end to adapt to changing software performance and specifications. The team worked to define a flexible software structure, then worked within an agile project management framework to incrementally add capabilities within a spiral development cycle. This allowed us to test some features early, while permitting us to adapt to expected changes to the government supplied simulation software being developed by OSRF.

[3] http://www.foxguardsolutions.com (accessed July 30, 2015)
[4] http://www.ros.org (accessed July 30, 2015)
[5] http://wiki.ros.org/rviz (accessed July 30, 2015)
[6] http://moveit.ros.org (accessed July 30, 2015)
[7] http://www.pointclouds.org(accessed July 30, 2015)
[8] http://www.redmine.org (accessed July 30, 2015)
[9] https://about.gitlab.com/ (accessed July 30, 2015)
[10] http://www.osrfoundation.org (accessed July 30, 2015)

Team ViGIR was one of eleven funded teams competing in Track B teams. As the competition approached, another 115 teams registered as unfunded Track C teams; of these 126 total teams only 26 teams passed the initial qualification tests. During the competition, only 22 teams scored points with Team ViGIR finishing in sixth place with 27 points. Figure 3 shows our operator at the OCS during the VRC competition.

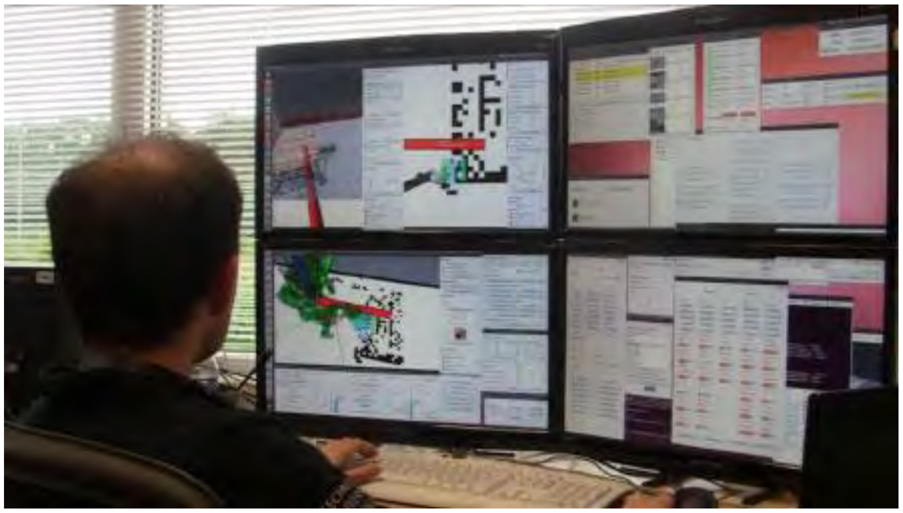


Figure 3. View of OCS during VRC

The team published a brief overview paper [2] about their VRC development and experience; this paper , is included in Appendix A.

2.3.2. Phase 1B- DRC Trials

Team ViGIR attended robot training in July 2013, and began to set up of their lab. As TORC could not have foreign nationals at their garage, Team ViGIR used warehouse space donated by Foxguard Solutions. The space required an electrical upgrade, which delayed our receiving the robot until 27-August-2013. The team set up a short-term housing rental near the lab; one Oregon State student spent the entire semester in Virginia, while TU Darmstadt students rotated through the lab spending a few weeks at a time.

Given the original VRC design focus on providing a flexible user interface and software architecture, the overall software architecture did not change between the VRC and the DRC Trials. On the other hand, the robot interface API was significantly different from the simulation API, and required significant changes to accommodate the new API. Team ViGIR leveraged a shared C++ source file from another Atlas team to develop an approach that worked with the robot hardware and the Gazebo simulation. The limited fidelity of the simulation and constantly evolving hardware interface limited the utility of the simulation for tuning of the control parameters; therefore, while the system used similar control approaches, it used completely different gain sets between the robot and simulation. Remote members of Team ViGIR used the simulation to test logic, user interface, and task process, while the lab team in Virginia focused on hardware testing and control tuning. The software development focus during this time was on adding additional capabilities to existing modules, and improving performance.

Team ViGIR published a detailed system overview paper in [1]; this paper, which discussed the system and results of the DRC Trials, is included in Appendix A.

PhD student researchers from LIRMM[11] in France visited Team ViGIR to work on whole-body motion planning for the ladder task. Their approach used model-based offline planning and optimization and has

[11] http://www.lirmm.fr/lirmm_eng/users/utilisateurs-lirmm/equipes/idh/abderrahmane-kheddar (accessed July 30, 2015)

been successfully applied to the (joint position controlled) HRP-2 robot. However, it was not robust enough to accommodate the significant modeling errors in the Gazebo model of Atlas and controller errors during execution. As neither a better model nor time for model calibration were available before the DRC Trials, Team ViGIR did not use the approach during the Trials. During Phase 2 this research for Atlas did not continue as LIRMM became part of the DRC Finals Team AIST-NEDO through the French-Japanese CNRS-AIST joint

The biggest challenge was the limited time between receiving the robot and the trials. The tasks of converting software to work on the robot, tuning controls to work with the robot hardware, and developing new interfaces for the actual hardware took considerable developer resources. During testing, we were debugging both our own code and the newly released versions of the BDI API. Given limited developers and time, we prioritized development decisions and focused on practicing a limited number of approaches to the tasks. In reviewing the evolving rule changes and developer resources, Team ViGIR made two fateful decisions. First, we chose to skip the driving task to focus on more general manipulation tasks. Second, we chose to focus on the wrong cutting tool based on a perceived ease of triggering. Although the team recognized the importance of stopping development and practicing with features in place, this mythical code freeze did not happen as our testing continued to reveal limitations that required updates. As our main operators were also our main developers, this represented a constant struggle to balance the need for testing and training with the need to fix bugs and extend capabilities. Further complicating this issue for Team ViGIR was the geographically distributed team. Our entire team was only on site together for the month prior to competition; prior to that, only a subset of our team was at the lab at any given time.

During the preparation for the DRC Trials, our robot was reliable and had consistent performance. We had some leak issues and a broken cable, but the Atlas hardware was not generally an issue. Our reported issues to BDI were mostly related to debugging software, sometimes on our side and sometimes with their API (e.g. step index handling). Our robot checked out well prior to shipping to the DRC Trials competition; unfortunately, this consistent behavior did not last after arriving in Homestead, FL.

The robot did not perform well during testing at the Trials. Upon arrival, we found that BDI had replaced a foot due to an apparent sensor issue that had not been seen in Blacksburg. The robot passed initial checkout standing and in manipulate mode, but consistently fell over when walking or stepping. After BDI assisted other teams, they began to checkout our robot overnight, and spent the next day testing and tuning trying to improve stability. They tried replacing the new foot, but continued stability issues prevented us from practicing during our normal slot. BDI eventually diagnosed the issue as a failing hip actuator, and performed a hip replacement the night before competition. These issues severely restricted our practice time at competition to the point that our first successful step was one hour prior to the first event.

Per task performance during the competition is documented in [1], which can be found in Appendix A. Figure 4 shows images from our robot during competition along with the points achieved during each task. We finished the competition with eight total points in a three-way tie for ninth place; the top eight competitors advanced to the DRC Finals. During winter 2014, Team Schaft (now owned by Google) dropped from the competition, which allowed DARPA to extend funding to Team ViGIR and Team THOR, and invite the third Track D team as a finalist.

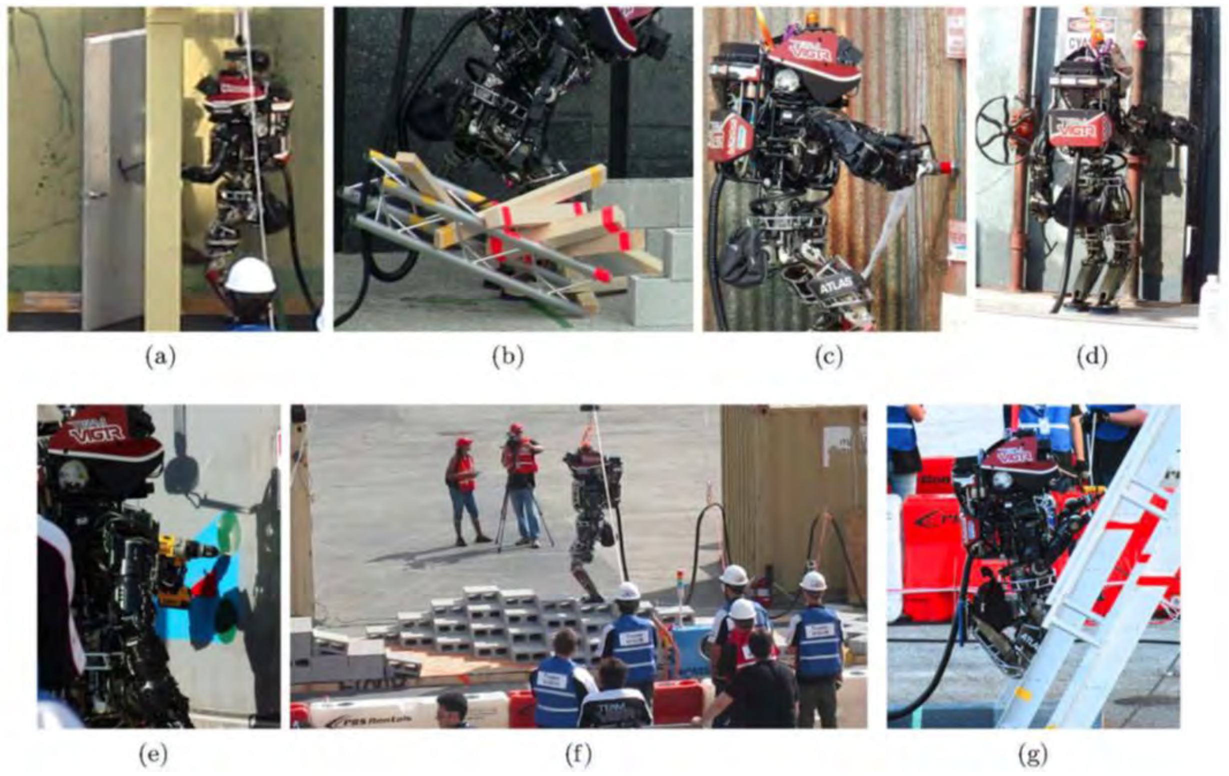


Figure 4. Performance at DRC Trials

2.3.3. Phase 2- DRC Finals

This section provides a historical overview of Team ViGIR's efforts during Phase 2.

2.3.3.1. Downtime (January – April 2014)

After a brief recovery period following the disappointing performance in the DRC Trials, Team ViGIR regrouped to begin documenting our efforts, and strategizing a way forward. During this time, Virginia Tech students approached TORC about using their THOR robot with our software. Dr. Hong, who was in the process of moving his RoMeLa lab from Virginia Tech to UCLA was not interested, but the students remaining at Virginia Tech reached out to the Virginia Tech administration for support in continuing to develop the new robot.

During the negotiations between Dr. Hong and Virginia Tech, DARPA announced the invitation to continue to the DRC Finals, with Team ViGIR and Team THOR splitting Team Schaft's share of the $1 million support contract. Team ViGIR notified Virginia Tech of our intent to focus on the Atlas robot, but agreed that we would make our software available to them to use if they pursued a separate entry. After a prolonged negotiation, Team THOR split into a UCLA/UPenn team using the Robotis THOR-MANG platform, and a new Team VALOR using the new ESCHER robot being developed by Virginia Tech.

Team VALOR chose to leverage Team ViGIR's software for VALOR's high-level planning and operator control interface.

During the delay, Team ViGIR provided DARPA and BDI the opportunity to display the robot at the Pentagon[12].

2.3.3.2. Restart (May – December 2014)

Team ViGIR reworked its budget to reflect the initial agreement for partial funding. The limited funding required creativity in project planning; the team decided that it was impractical to bring Cornell onto the team with such limited funding. While waiting on the final contract details, Team ViGIR began a search for new lab space as our prior lab space had been rented out to a new tenant.

Initially, we discussed available space on the Virginia Tech campus in exchange for our software support for Team VALOR. As the approval process drug out, the Montgomery County Economic Development Authority provided a larger more appropriate space. This space required an electrical upgrade by moving the 480V transformer from our original space to the new lab. This again delayed getting our robot functional until June 19, 2014.

During this set up time, DARPA notified Team ViGIR of the possibility of gaining additional funds due the delay in final competition schedule. Team ViGIR submitted a proposal that included additional test support equipment along with increased hours for researchers, and permitted bringing Cornell back onboard. Under the increased funding, TU Darmstadt partnered with Leibniz University of Hannover to provide researchers with experience in advanced controls. Cornell was able to start in September 2014 under contingent funding for the fall semester; the final ECP contract modification, which provided the same funding level as other Track A and B teams, took effect in October 2014.

One of the distinguishing features of Team ViGIR's proposal was the use of synthesis techniques to generate autonomous behaviors. As Cornell came on board late due to budget uncertainty, the team chose to focus on manual specification of the autonomous behaviors. This approach allowed Cornell researchers to get up to speed on our system, while contributing to the autonomous behavior development for the competition. In parallel, the Cornell team worked on defining the synthesis framework within the ROS SMACH[13]-based hierarchical state machine framework. The team demonstrated these synthesis concepts in experiments after the competition; we discuss these results in this final report and point the way to future research that will continued by members of Team ViGIR.

As team discussed the basic architecture that we used in the DRC Trials [1], we agreed that the basic structure was sound but needed some improvements. First was an improvement to the manipulability of the robot, which was being handled by BDI's redesign of the Atlas robot. Second were improvements to the state estimation and calibration of the robot arms; after researching several alternatives, Team ViGIR

[12] http://www.cbsnews.com/news/pentagon-introduces-atlas-its-new-robo-sapien/ (accessed 30-July-2015)
[13] http://wiki.ros.org/smach (accessed July 30, 2015)

made use of the Pronto[14] system being open sourced by MIT for robot posture estimation, and developed a custom calibration motion to detect and compensate for variable encoder offset issues.

The team made a decision to move to a newer development environment that provided access to newer libraries and offered better long-term support for our future open source efforts. The VRC and DRC Trials used the ROS Hydro ecosystem with an Ubuntu 12.04 Operating System and a hybrid rosbuild and Catkin[15] build system. After discussion, the team decided to migrate to ROS Indigo, which required a jump to Ubuntu 14.04, and chose to migrate all of our software to the catkin build system. The team implemented this changeover incrementally over the summer and fall 2014 while we developed new features. In addition to updated code, this software conversion provided simplified installation and remote deployment options using the catkin install feature.

In order to provide better support for autonomous control and behavioral interfaces, the team decided to standardize on the ROS ActionLib[16] interface. As part of this process, the team converted the robot interface to use ROS Controllers framework[17]. This provided a more ROS-centric development, and better integration with existing tools such as MoveIt!. As the team implemented new interfaces or made improvements to existing modules, some of these, such as the footstep planner were converted to the Action interfaces as well.

The team worked on an approximately eight-week cycle with six weeks of development and simulation based testing, followed by travel to the lab for hardware testing. These test sprints were held in late June 2014, September 2014, and October/November 2014.

During the fall 2014, Vice Media contacted TORC and stated that they wanted to do a report on "*how the software you're developing might help with search and rescue efforts in the future*." We spoke with them on the phone along these lines, but once on site for videotaping, the questions devolved into "killer robots." They published their *Dawn of Killer Robots* video[18] on April 16, 2015, which included footage of Team ViGIR and Team VALOR.

Team ViGIR worked through November 2014 with the original Atlas robot, and then packed and shipped the robot back to Boston Dynamics for the new Atlas Unplugged upgrade. November 2014 through January 2015 included significant development on the grasping interfaces and footstep planner as the team worked remotely without access to the robot hardware and used the old Atlas simulation model.

2.3.3.3. Atlas Unplugged (January – May 2015)

The team anticipated the upgraded robot's return in early January, and planned travel for our German partners for integrated testing beginning in mid-January after an initial checkout period. The focus of this test was to be system identification and compliant controls development, in preparation for working on the fall recovery and vehicle egress motions. Unfortunately, BDI had significant hardware delays.

---

[14] https://github.com/mitdrc/pronto (accessed August 19, 2015)
[15] http://wiki.ros.org/catkin (accessed July 30, 2015)
[16] http://wiki.ros.org/actionlib (accessed July 30, 2015)
[17] http://wiki.ros.org/ros_control (accessed July 30, 2015)
[18] http://motherboard.vice.com/read/inhuman-kind-killer-robots (accessed July 30, 2015)

Given the delays, BDI provided Team ViGIR limited access to the robot at a workweek in Boston starting January 12, 2015; this gave all the teams a chance to work on the newly released API. The new API broke compatibility between the robot hardware and simulation environment, which required a non-trivial conversion effort. This change necessitated having different models for simulation and hardware based testing for several months. After additional delays, BDI pushed delivery of the partially upgraded robot to February 21, 2015, only three and one half months before competition.

During this hardware delay, some of the students at TU Darmstadt decided to qualify their THOR-MANG robot for the DRC Finals as a contingency plan. DARPA accepted this qualification with the stipulation that the entry be treated as a completely separate team. This new team, now called Team Hector, used the Team ViGIR software as a base, and contributed upgrades to several modules as they tested our software.

In early March 2015, the team traveled to the DRC Test Bed event to test the communications bridge under the competition set up. The team identified several issues with our setup, and worked to address those issues as discussed in Section 3.3.

BDI was unable to install the new electric arms on Atlas until late March; these arms had seven degrees of freedom for improved manipulability, but required different control tuning. In addition to the hardware delays, there were several issues and hardware failures as the new design was being beta-tested in the field. During this time, Team ViGIR required three arm replacements, two perception computer replacements, and two perception node swaps to fix a PPS error. Unfortunately, these issues were common to all of the Atlas teams, which taxed BDI personnel as they tried valiantly to support seven robots in the field with the new design. In the end, our robot did not receive its final hardware upgrade until Monday June 1, 2015 at the competition site.

Team ViGIR had planned to assess the IHMC walking and whole body controller. IHMC had similar Atlas hardware issues, therefore they were unable to test their open source software for release in a timely fashion. As we could not test the open source system early enough, we chose not to devote resources to integrating our software with their robot interface and control system, and continued to rely on BDI's base level stepping and balancing controller.

Another casualty of the hardware issues and delays was the development of our compliant controller. Team ViGIR had partnered with researchers from the Leibniz University of Hannover, through the sub-contract with TU Darmstadt, to develop a compliant controller specifically for use while in contact with rigid objects. We intended to use the compliant controller during the cutting task, vehicle egress, and fall recovery motions. Initial development got underway in Q4 2014, with January planned for system ID and testing; BDI did not deliver the hardware until late February, and then delivered the robot with some issues remaining, which severely limited our development time. In the end, the team chose not to use this compliant controller in competition. There were conditions where the performance was worse than the basic position control, and we were unable to get the issues resolved in time for operator training. This report includes results of the efforts related to system identification and controller development in Appendix D; this includes additional fixes and tests conducted after the Finals competition.

Initially, Team ViGIR had planned to focus on the fall recovery and vehicle egress behaviors prior to developing the driving interface. The fallback plan had been to walk the course in lieu of driving if we could not egress reliably. As hardware issues delayed development of our compliant controller, this directly affected development of an egress motion. Given the limited development and testing time due to hardware issues, as well as potential risk to the robot, Team ViGIR chose not to pursue fall recovery or vehicle egress behaviors after the pump upgrade in late March. Prior to the pump upgrade, the robot did not have sufficient power to raise its arms with hands attached, much less push up after a fall. Given the limited spare parts and subsequent down time caused by any needed repair, the team decided it could not risk damage to the robot as the boundary between success and failure is very small, and would require extensive testing and tuning on the robot hardware.

Team ViGIR opened our lab to Team Hector, and allowed them to make use of our test facilities prior to competition in exchange for testing our high-level software, and assisting in development of some capabilities. As the decision to allow a reset instead of requiring vehicle egress came relatively late in the game, Team ViGIR chose to let Team Hector develop the basic driving interface, while we focused on other software issues. Testing with the THOR-MANG robot outdoors was much easier as their robot could more easily fit into the vehicle and operate the controls, and operate outdoors on battery power. Team ViGIR developed a compliant steering handle concept, which we shared with Team Hector. After students working for Team Hector developed the basic interface, Team ViGIR customized the robot commands for Atlas. As this driving interface came together relatively late, the focus was on sending steering commands to robot; the robot did not have any onboard planner for obstacle avoidance and generating steering commands.

Researchers at Oregon State worked on a number of components including a hand guard, hand cameras, and tactile sensing. The team developed a Raspberry Pi-based interface for the hand electronics as described in Appendix B. Initially there were plans to use Takktile tactile sensors, along with grasp quality analysis software; the student did not complete this work in time for proper integration, so the team chose to not use the tactile sensors. In the end, only the small palm cameras were used in competition. The team designed a hand guard to protect the electronics and fingers during a fall, and to provide the ability to push off during a fall recovery. Once we decided against developing fall recovery and no longer needed to push off the ground, we decided that the risk of the necessarily bulky hand guard outweighed the risk to the electronics during a fall; therefore, the guard was not used in competition. A final research thrust was an online grasp planner. While showing promise in isolation, the integration into the larger system proved too much for the student researcher, and the larger team chose not to devote scarce resources to integrating this software as we felt the template-based approach was sufficient for the tasks shown at the South Carolina Test Bed.

Team ViGIR continued our collaboration with Team VALOR by jointly contracting a dedicated tractor-trailer truck to ship our robots and equipment to California. Team Hector added their equipment to the shipment. The truck departed Blacksburg on May 28, 2015. While the truck hauled the equipment, the team took a well-deserved break to get some rest and relaxation before arriving in Pomona May 30. The truck arrived safely on June 1, 2015.

2.3.3.4. Finals Setup (June 1-4, 2015)

Unloading and unpacking proceeded relatively smoothly. DARPA provided sufficient equipment and cooperative personnel to assist the unloading. The set up on site was sufficient for our needs. After concerns regarding the gantry height were resolved, the team began checkout of our robot to verify performance after the transit across the USA. That evening, BDI replaced a faulty component and replaced a damaged footpad. At this point, our Atlas Unplugged robot was finally 100%.

During a subsequent BDI checkout, they reported that our robot appeared to be "running hot," but they could not find any obvious cause that they could fix. Randomly swapping out components did not seem warranted. Over the course of the week, two different BDI technicians gave us the same report, but could not find any resolution.

During this time, Team ViGIR was working to resolve lingering issues with the communications software.

Team ViGIR had the first chance to test the robot on battery power June 3 at Pomona. Unfortunately, the checkout was in another building that did not have access to the DRC network infrastructure; this required us to transport our field and operator computers to the building. We chose to bring only one operator station computer, which caused significant confusion among our multiple operators as they tried to share the one terminal; this made the operator training time less useful than we had hoped. In our normal mode of operation, each operator has an independent workstation that shares data to allow distributed collaboration.

At the dress rehearsal on June 4, Team ViGIR chose to focus on practicing the driving task (Figure 5) and forgo risking damage to the robot in a fall off tether. As the team had not had the opportunity to practice with the Atlas robot in a moving vehicle, the team practiced the driving portion twice by requesting a reset after the first run. During both runs, the communications system and driving interface worked well. In the first run, the team approached the finish line fast and the DARPA observer E-stopped the vehicle prior to crossing the line. The second run went well, and the field team practiced the egress.

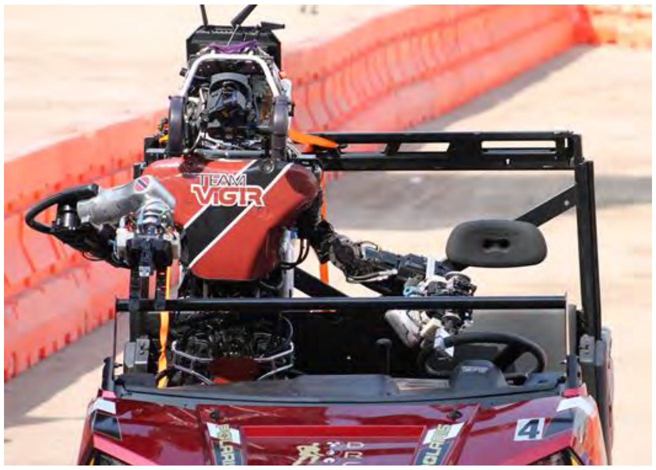


Figure 5. Driving practice during DRC Dress Rehearsal

The biggest issue noted by the field team during dress rehearsal was the difficulty working with the small gantries provided by DARPA. Our team was able to get everything lined up in the time limits allotted, though we did experience some difficulty getting the gantry in position due to the soft dirt at the starting gate. Once underway there were no issues until we attempted to remove the robot from the vehicle with the government furnished gantry at which time the gantry trolley got stuck in place making it very difficult to remove the robot from the vehicle. This issue was reported to DARPA and the gantry was either fixed or replaced before Day 1.

2.3.3.5. Competition Day 1 (June 5, 2015)

On competition Day 1, we received three points as time expired just shy of achieving the fourth.

The actual competition run faced more issues in regards to field operations than the dress rehearsal. In order to make up schedule slippage due to the prior teams issue, DARPA personnel wanted us to start the communications checkout prior to BDI installing the battery even though it would prematurely start our 20-minute setup period. Our Field Team Lead coordinated with government observer to address this issue; however, in the end, we are not certain that we received the full 20-minutes after BDI completed the battery installation. The government gantry was too low to allow us to place the robot in position in the vehicle in the same manner as the prior day. The team had to get four of the five members standing on the vehicle to lower the suspension enough for atlas to get in the correct position for us to drive which cost us several minutes. The team was forced to rush the setup process compared to the dress rehearsal, and our set up over ran our start time by approximately three minutes.

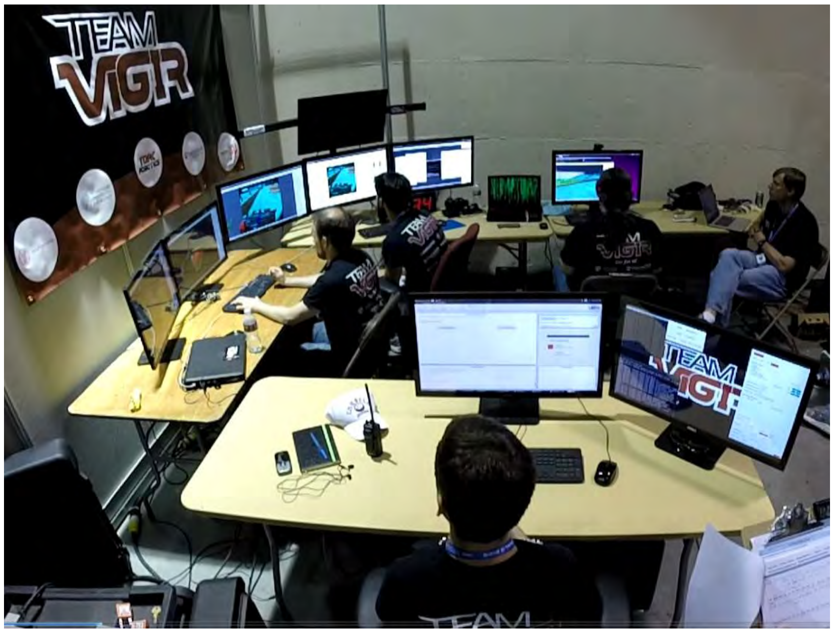


Figure 6. Operators on DRC Finals Day 1

In spite of the issues at setup, our robot named Florian worked well and our operating team directed Florian through the driving course for one point (Figure 6). After crossing the driving finish line, we executed our planned reset without issue.

After the ten-minute reset penalty, the team worked to open the door, but noticed that certain systems were not operating as expected due to communications issues, including an apparent backlog of data sent between the onboard and field computers. This was not the expected degraded communications between the OCS and field. Speaking with other teams, we found that they had experienced similar issues, and their monitoring detected that the communications bandwidth dropped to less than twenty percent of maximum as the robots approached the grand stands. Our teams did not experience these issues during the dress rehearsal, and speculate that the presence of spectators with smartphones and increased media transmissions introduced significantly more interference with the wireless communications between the field and onboard computers during the actual competition.

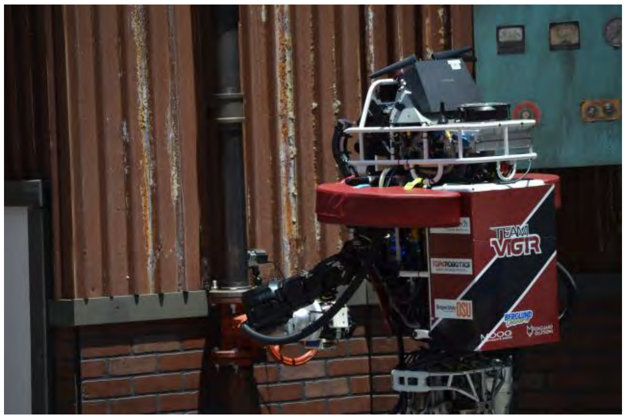


Figure 7. Opening the valve on Day 1 of DRC Finals

The operators were still able to direct the robot to open the door and walk through for our second point, and open the valve for our third point (Figure 7). While we successfully achieved these three points, the operations under these conditions were much slower than expected. Near the end of our run, while reaching for the switch that was part of the surprise task, our right arm overheated and stopped functioning. Time expired before we were able to recover and achieve the fourth point.

At the end of the Day 1 run, our field team, assisted by personnel from Boston Dynamics, recovered the robot without a damaging fall.

After reviewing our performance, we felt that we understood the communication problem and had a plan for Day 2. That night we rearranged the software to minimize the wireless communications with our field computer, and made several changes to reduce our required bandwidth. During preliminary testing that night, the changes were working well.

2.3.3.6. Competition Day 2 (June 6, 2015)

During our robot checkout on the morning of Day 2, the robot and control software worked well. As the field team loaded our robot for transport to the robot course, our operators were cautiously optimistic that we could score 5 or 6 points during our run.

Given issues with the gantry on Day 1, we opted to bring our own gantry to the start for Day 2.

As the team powered up the robot after arriving at the start line, the robot passed the initial checkout including hand operation and arm calibration. At a later point in the checkout, the team discovered that the robot right arm had stopped working and was completely dead. BDI sent a technician over to investigate and DARPA granted us a twenty-minute delay to debug the hardware issue. Our team was still under considerable time pressure to debug the issue and restart our system software while the robot baked in the California sun. The team disconnected the hand and did a full robot power cycle to test if the arm was truly broken; the power cycle restored functionality to the arm. After calibrating the arms, the field team plugged all hand electronics back in and the arm continued to work properly during checkout.

By this time, we had used half of our extension time, so DARPA granted an additional ten minutes before our clock started to load the robot. The insertion of the robot into the vehicle went much smoother with our larger gantry and we were able to start without spending any run time.

During our drive, there was an unexplained communications delay between our operator interface and the robot. At one point on the course, the vehicle did not move when first commanded; after it started moving, our operators requested it to stop, then watched helplessly as the robot continued to drive into a barrier. (The system was not doing autonomous motion planning for the vehicle steering.) After resetting the robot and vehicle to the start line, the robot continued to bake in the sun with its pump running while we waited for the 10-minute penalty to expire.

This time our robot and vehicle successfully crossed the finish line with our team driving cautiously down the course. We requested our planned reset again. After waiting through the remainder of our 10-minute penalty, our team quickly – as compared to Day 1 – opened the door, and began to position the robot to walk through the door. At this point, the robot pump shut off and the robot fell to the ground (Figure 8). After reviewing our operator station screen cast videos, we could see that the reported reason for the pump shutoff was a communications failure with the BDI software. Our software appeared to be operating normally, but the robot was running extremely hot, which may have contributed to the communications issues.

Our robot survived the fall, and our team reset for another attempt at the doorway. During the restart, we again had an issue with the right arm, and decided to bypass the custom hand electronics that may have been damaged during the initial transit to the arena. Unfortunately, after waiting through another 10-minute penalty, the robot fell again half way through the door, likely caused by damage sustained in the original fall. At this point, running low on time, energy, and spirit, our team stopped for the day.

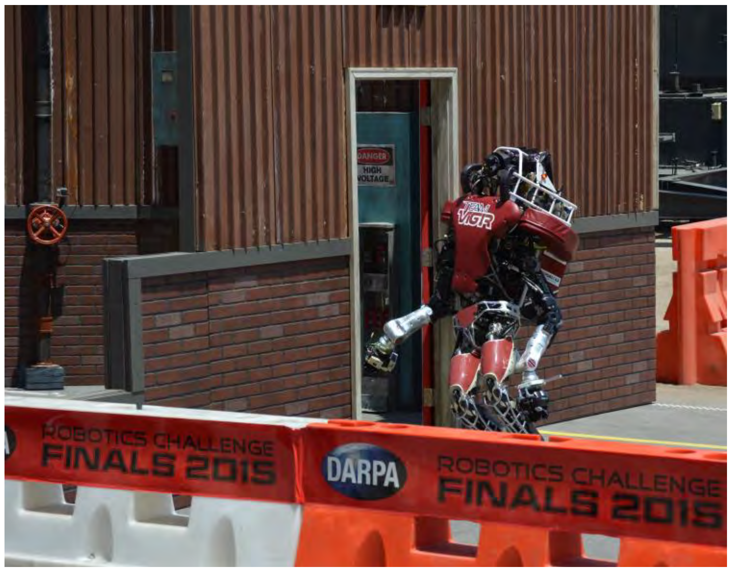


Figure 8. Robot collapsing due to pump shutdown after opening the door on Day 2 of the DRC Finals.

2.3.3.7. *Post Competition*

After the competition, Team ViGIR shipped the robot back to Blacksburg, VA. After the robot checkout in Blacksburg, it appeared that the robot suffered only minimal cosmetic damage during its two falls. During subsequent experiments, a sensor failed on the robot and prevented the robot from being able to step or walk. The team continued testing robot controls, grasping, and manipulation based behaviors. Later more leg sensors failed, which prevented the robot from standing. The technical sections of this report document the results of these experiments. At the conclusion of these experiments, the robot was returned to the government as requested.

2.4. Report Overview

With this historical context in place, the remainder of this report focuses on the technical contributions. The report presents experiments that validate performance beyond that witnessed in the DRC competitions. The report documents the software in its current state, including changes made after the finals in support of our efforts to open source our code base. The main body of the report serves as an introduction to the technical details, which we present in the appendices. Section 3 introduces the design philosophy, software architecture, and innovations developed by Team ViGIR during the course of this competition; Section 4 discusses significant challenges and focuses on the experimental results. Section 3 and 4 reference the same appendices grouped by major component; each appendix contains a brief introduction and embedded PDF files corresponding to technical papers and reports written in another format. Section 5 concludes the report with a discussion of lessons learned, and future work that is necessary to bring the original vision to reality. Section 6 includes a limited bibliography of works published by the team; the papers included in the appendices cite references that are more general. The document concludes with appendices that embed technical papers and reports prepared by the team.

This page intentionally blank.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

From the outset, TeamViGIR functioned as an open collaborative research and development effort where all team members shared and contributed to the code base, with the intent of open sourcing our software at the end of the project to facilitate future development toward humanitarian rescue robotics. This section provides an overview of our software architecture, and ties this to our open source software[19] released concurrent with this document.

The major design focus of Team ViGIR, and a major focus of the DRC, is the development of an approach that leverages the complementary strengths and weaknesses of the robot system and human operator(s). While full bandwidth and update rate access to all sensory systems is available onboard the robot system, cognitive and decision-making abilities of a human operator are still vastly superior for the near future. This is especially true for disaster scenarios, as only very limited assumptions about their structure can be made beforehand. Team ViGIR took the approach of making the operators members of the team, while permitting the robot to exercise supervised autonomy given task level directions. See [1] included in Appendix A for an overview of our design approach.

The software architecture employed by Team ViGIR included the Operator Control Station (OCS) and the *Onboard* software, which includes software running on the robot as well as on an external *field* computer. The field computer looked toward future systems that include more computational power onboard, and would not require a separate field computer. Where the DRC Trials used three field computers and one onboard computer running BDI software, the Atlas Unplugged version used at the DRC Finals included three perception computers onboard in addition to the BDI control computer; Team ViGIR made use of one field computer to handle communications at the DRC Finals. The Communications Bridge (CommsBridge) software developed for this project handled the communications between the OCS and Onboard software. Figure 9 shows the basic architecture followed in this project.

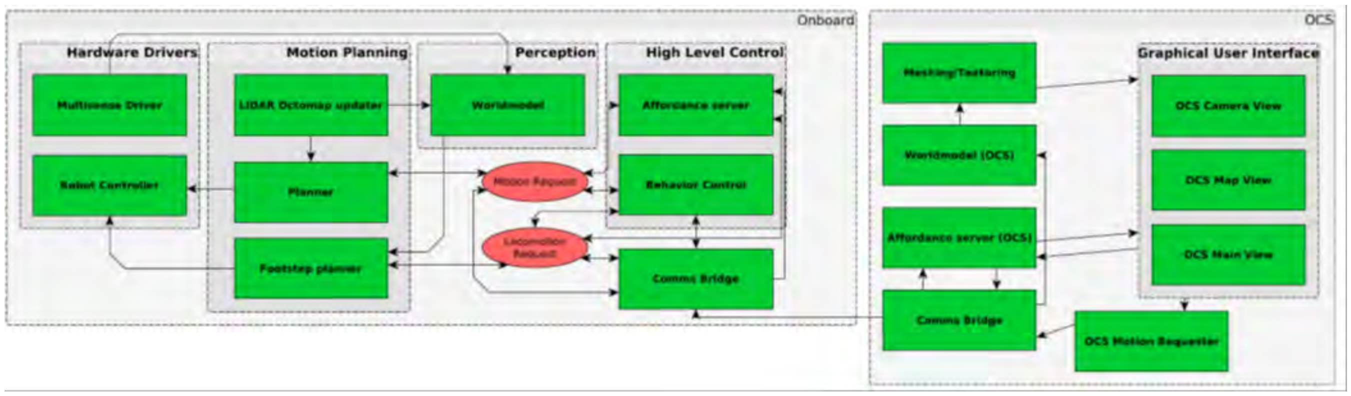


Figure 9. Software Architecture

[19] http://www.github.com/team-vigir

3.1. Operator Control Station

The OCS includes both the User Interface (UI) components as well as a number of software components including OCS-side planning, communications, and multi-operator coordination. In many instances, these non-UI OCS components mirror major components on the Onboard side. This section begins with an overview of design priorities, and then focuses on the operator roles and UI components; Appendix C provides a brief overview of the software included in the open source software release, and the non-UI components in particular.

3.1.1. Design Approach

Section 3 of [1] (Appendix A) provides an overview of our design philosophy from the outset of the project, and its implementation through the DRC Trials. We summarize our primary design principles as follows:

Human capabilities: Since the DRC was a competition with tight time constraints, it was important to leverage the abilities of human operators and take advantage of the things they were good at, rather than working only towards full robot autonomy. For example, humans can easily pick out salient features in real-world scenes and describe their position and orientation. This led to our use of 3D templates. Templates (3D models of important objects/features in the environment) allow the primary and secondary operators to annotate perception data with semantic information. For example, if the operator sees a known object in the point cloud (e.g., a tool), he can insert a template representing that tool in the 3D view at that location, thus informing other operators and the onboard systems about that object. Figure 12 in Section 3.1.3.1 shows the addition of template information into the scene; Section 3.2.4 discusses template use in more detail.

Pre-visualization: Software on the OCS side has access to a wealth of information about the robot and the environment, providing an opportunity to visualize proposed actions virtually before executing them on the physical robot. To make decisions about whether to execute, cancel, or modify the action, operators must be able to visualize the expected results. Thus, a second major feature of our OCS is the “ghost” or simulation robot, which is a transparent duplicate of the ATLAS robot visualization. The ghost robot allows the primary operator to plan and validate motions before executing them with the physical robot. The ghost robot is also color-coded to give the operators feedback about the internal state of the onboard systems, such as collision checking for motion planning, to prevent unexpected actions. Both of these features can be used in and visualized at any of the views described below.

Multiple operators: Although autonomy was an important goal of the DRC, it was clear from the outset that human operators would play a major role in making high-level decisions and giving supervision and direction to the robot’s (semi-) autonomous capabilities. It was also clear that a single operator would not have enough perceptual or motor bandwidth to take in all the information coming from the robot and provide all the information needed by the robot. Thus, we designed an OCS that could be run in multiple instances with multiple configurations, tailored for multiple operators with different roles.

Parallelism: Closely tied to the concept of multiple operators is the idea that multiple actions can be performed on the OCS side in parallel. An operator can plan the next movement while the current one is being executed. Multiple operators can be working on planning, template placement, visual inspection of sensor data, and other operations, all at the same time. This principle was critical to enable good performance under tight time constraints and in the presence of degraded communications.

Appropriate specificity: Our OCS design had to strike a careful balance between generic interfaces that could be used in any situation, and highly specific interfaces that were tailored for individual tasks, sub-tasks, or robot capabilities. For the DRC Trials, DARPA provided all task information ahead of time, so a task-specific OCS might be quite successful. However, we wanted to demonstrate a flexible OCS that could be used for multiple tasks with unknown parameters, such as the surprise task at the DRC Finals. Our only task-specific UI was a specialized widget for driving.

Advanced interfaces: Our team included members with expertise in virtual reality (VR) and 3D interaction, and we felt from the outset that these technologies might be beneficial for robot operation. Immersive VR for 3D visualization, either from the robot's point-of-view or elsewhere, could allow operators to easily access any view of the robot and its environment; this could prove very useful for visual inspection of alignment and positioning. 3D interaction could give operators powerful techniques for manipulating objects, such as 3D templates, with multiple integrated degrees-of-freedom. At the same time, we realized that these interfaces would be experimental in this domain, so we focused much of our effort on a more standard desktop interface (albeit one with multiple monitors and 3D mouse capabilities).

Iterative design and evaluation: Like all good UI development efforts, our OCS design needed constant testing and iteration. Fine-grained iteration took place throughout the project. A major new iteration was planned and developed after the DRC Trials.

In analyzing the OCS used at the DRC Trials, we noted two major issues. The first issue was a lack of integration of specialized control widgets, which increased the learning curve of our UI; a key goal of development during Phase 2 was to better integrate these widgets and make them accessible from the main UI through the use of pop up context sensitive menus and readily accessible icons.

The second issue was the use of our multiple operators. In the run up to the DRC Trials, we had limited time to train on stable system software. This led to different people having different specialties, with operators switching roles during the tasks (e.g. step planning vs. manipulation). This directly led to a loss of situational awareness that caused a fall during the door task at DRC Trials. Thus, during Phase 2, the team worked to provide streamlined control interfaces with better UI integration to simplify the use of interfaces, and to better define the roles and responsibilities of each type of operator.

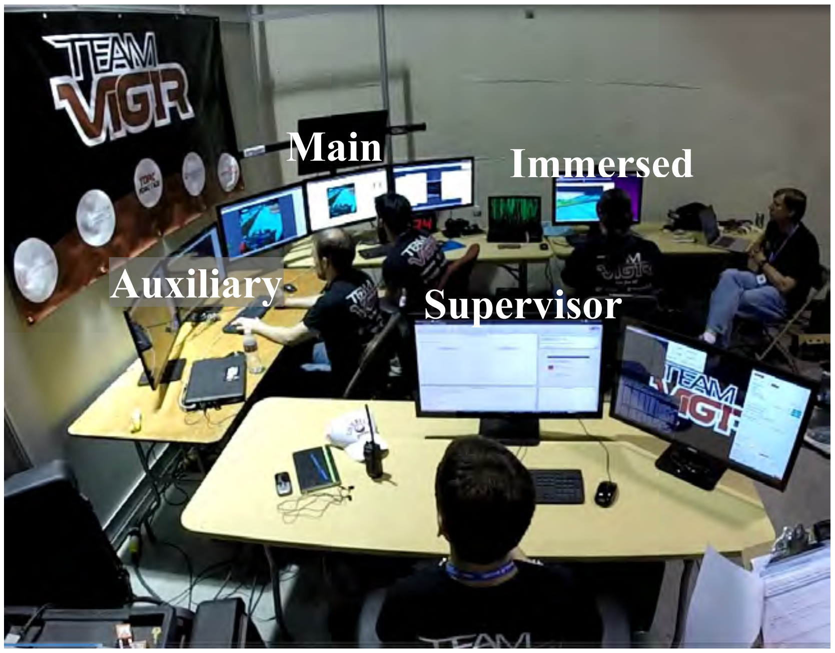


Figure 10. Layout of the operators during DRC Finals

A final design goal was to incorporate better 3D visualization tools for fine alignment, and validation of positioning.

3.1.2. Operator Roles

Team ViGIR used multiple operators for both the DRC Trials and Finals. The individual operator stations were separate instances of the same UI that shared data between operators; thus, if one operator requested a point cloud, the same point cloud would be visible on all stations. This allowed the operators to coordinate verbally with one another, which permitted operation as a "Wizard of Oz" interface where one operator could request another to gather the additional information needed [1]; this reduced the cognitive load on any one operator.

For the DRC Finals, Team ViGIR used four operators with well-defined roles:

- Supervisor
- Main
- Auxiliary
- Immersed

Figure 10 shows the arrangement of these four operators during the DRC Finals. The remainder of Section 3.1.2 describes the primary roles of each operator.

3.1.2.1. Supervisor

The Supervisor was responsible for overseeing and managing the execution of high-level behaviors via our Flexible Behavioral Engine's (FlexBE) graphical user interface; this is presented in Section 3.1.3.4. The supervisor was also responsible for keeping the operators on task and ensuring that operations were conducted according to plan.

3.1.2.2. Main Operator

The Main Operator was responsible for the interacting with the OCS UI to plan or verify motion generated by behaviors, and for conducting manual operations if autonomous behaviors failed. The main operator was responsible for specifying footstep goals, managing templates, and performing manual manipulation.

3.1.2.3. Auxiliary Operator

The Auxiliary operator was responsible for gathering perception data in support of the main operator in order to maintain a high-level of situational awareness, as well as inserting templates or other semantic information as requested by behaviors. For our team, the Auxiliary operator also served as team lead during the run, and was responsible for making the final decisions on tactics during the run.

3.1.2.4. Immersed Operator

For the DRC Finals, Team ViGIR added an operator station that included an Oculus Rift DK2[20] virtual reality head-mounted display (HMD). The HMD's 3D position and orientation was tracked, allowing the

[20] https://www.oculus.com/en-us/dk2/ (accessed July 30, 2015)

operator to move/turn his head naturally to obtain new views of the 3D scene. This permitted the Immersed Operator to visually inspect fine alignments (e.g., will the robot fit through the door if the proposed footstep plan is used?) and assist in situational awareness by making use of both the 3D sensor data and the modeled information including the robot and object templates.

This operator was running an instance of the same OCS used by other operators, with a specially-designed 3D stereoscopic view for the Oculus Rift. Navigation was performed with a pair of Razer Hydra 6-DOF controllers; buttons on these controllers were also used to toggle or adjust various aspects of the 3D view shown in the HMD and to quickly move to different points-of-interest in the environment. Initially, we expected the Immersed Operator to aid in template manipulation as well, since the 6-DOF controllers are ideal for rapid placement and rotation of 3D templates, but this feature was not tested sufficiently before the Finals to allow its use.

3.1.3. Major User Interface Features

The Main and Auxiliary operators had a separate instance of the three major interfaces (main, map, and camera views); the Supervisor had access to a specialized FlexBE interface to the behavior executive; and the Immersed operator had a specialized version of the main view.

Since we use ROS, we took advantage of the several existing UI tools that it provides, mainly *librviz*[21] and *rqt*[22]. Leveraging the existing tools in librviz for visualizing 3D data communicated via ROS was very important given the short development timeline. All of the major views use existing or customized (e.g., adding support to our own methods for picking geometry) versions of rviz plugins; the team implemented some completely new plugins that implement some of the unique features of our OCS (e.g., templates). In the development of our main widgets, we extended the base librviz capabilities with *Ogre*[23] and *Qt*[24]. For the development of simple 2D widgets, we used rqt extensively; this allowed us to quickly prototype widgets during development that acted as windows for specific controllers on the onboard side (e.g., footstep controller parameters). The OCS now integrates these more specific widgets, which can be accessed and hidden by clicking specific icons on the major UI windows. Figure 11 shows the screen view of the Main operator station, which includes all three major interfaces, during the DRC Finals valve task. This view shows the camera view to the left, main view in the center, and map view to the right; in this case, additional specific widgets cover the map view.

[21] http://wiki.ros.org/rviz (accessed July 30, 2015)
[22] http://wiki.ros.org/rqt (accessed July 30, 2015)
[23] http://www.ogre3d.org/ (accessed July 30, 2015)
[24] http://www.qt.io/ (accessed July 30, 2015)

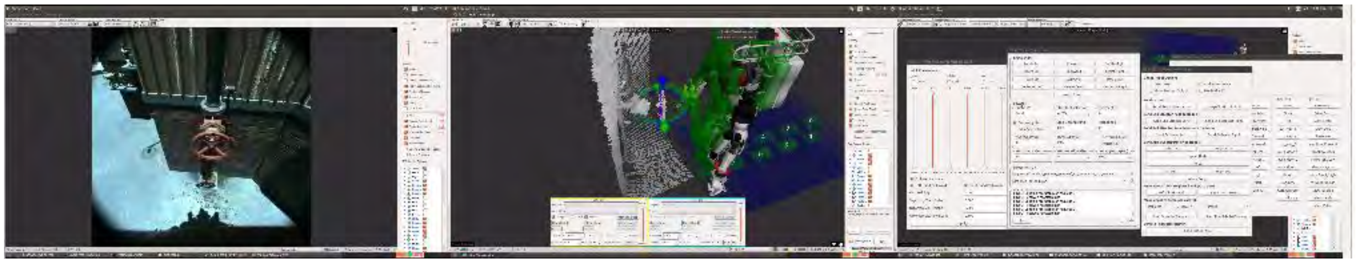


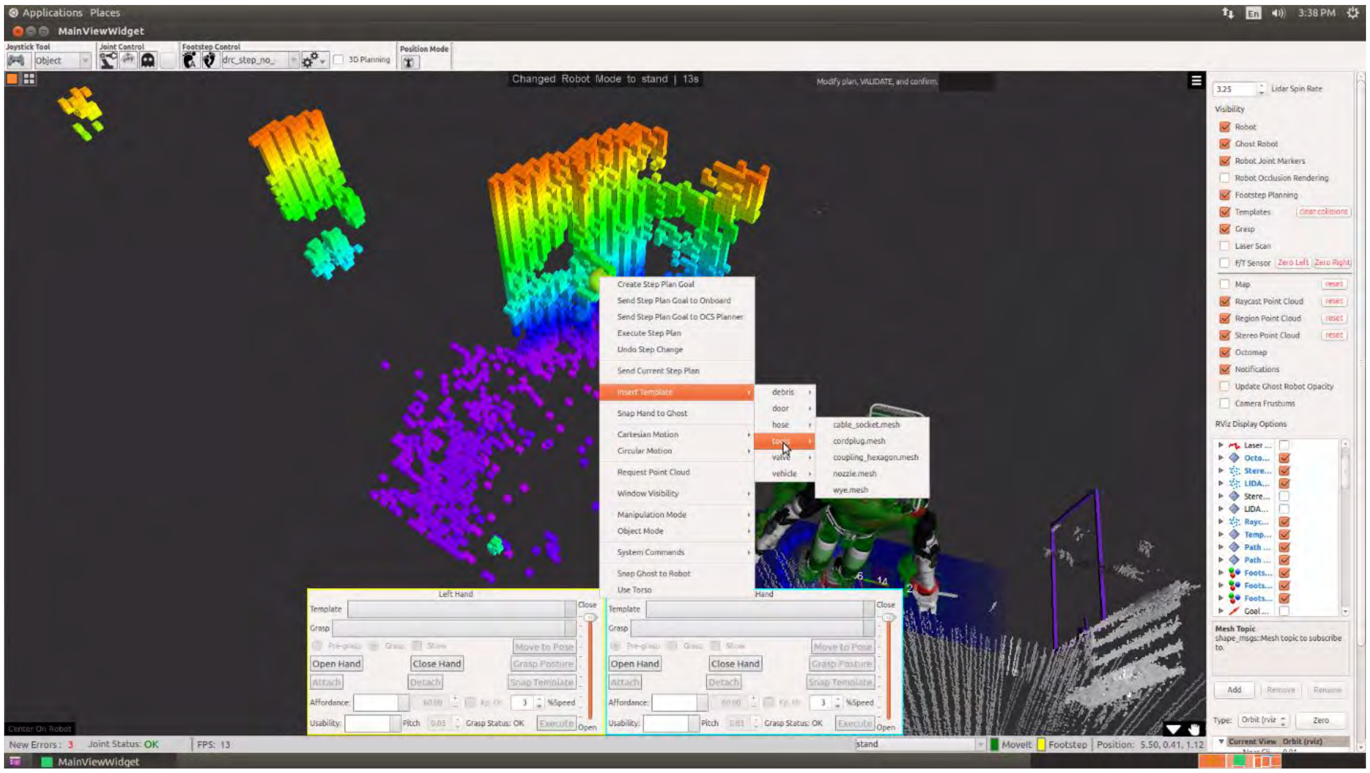
Figure 11. Main operator views – camera, main, and map – during of DRC Finals valve task on Day 1. Note that by task specific control widgets, which can be accessed from the main view, cover the map UI.

3.1.3.1. Main View

The *main view* widget, which is primarily used for visualization of 3D data and fine manipulation control, is an interactive 3D view built on the librviz base. The main view includes custom extensions to simplify selection and addition of template information, and manipulation of 3D data. It allows the operator to control end effectors, visualize the 2D and 3D reconstructions of the environment, annotate these visualizations with templates, and plan robot motion by controlling the ghost robot. A single icon in the upper left allows the operator to toggle between a single 3D view and four 3D visualizations with different points of view and settings (orthographic/perspective) to facilitate spatial judgments and aid depth perception.

The main view includes a number of visualization and control components. The right panel on the main view includes options for controlling what data is displayed on this particular display; the controls include all of the standard RViz marker types. The hand grasp controls, which interface with our template-based affordance scheme described in 3.2.4, are shown in the bottom middle of the view; these can be accessed via the hand icon on the lower right corner of the view. The top menu bar includes icons for accessing specific joint and footstep control widgets; clicking these icons toggles the display of these widgets for a specific instance of the view.

The main view also includes context sensitive pop-up menus to provide easy access to common control interactions. Figure 12 shows a close up of the main view during the Day 1 valve task, where a pop-up menu is being used to insert a template into the world model. To avoid too much clicking and menu selection, most of the options in these pop-up menus are also accessible via keyboard shortcuts (i.e., hotkeys).



Changed Robot Mode to stand | 13s

Modify plan, VALIDATE, and confirm.

New Errors : 3 Joint Status: OK FPS: 13

Position: 5.50, 0.41, 1.12

Figure 12. Main View showing placing a template via context menu onto selected Octomap cell

In this example, the operator has selected a particular cell within the Octomap[25] representation of LIDAR data. The operator is in the process of selecting the proper valve template, which will automatically be placed at the referenced Octomap cell. After an operator places a template marker in the 3D view, any of the operators can perform fine alignment using customized versions of the ROS interactive markers[26].

A key use of the main view is to manipulate templates and visualize the target pose of the robot prior to execution through the use of the "ghost robot." Figure 13 shows the ghost robot in the "pre-grasp" pose used before inserting the valve turning attachment into the valve. As discussed in Section 3.2.4, the pre-grasp target is defined relative to the template placed relative to the 3D world frame. The main view allows the operators to verify the template placement and the target robot pose relative to sensor data. The operator can easily monitor execution errors by checking the final pose of the actual robot against the ghost robot. A simple hotkey allows the operator to snap the ghost to the current state of the robot. The operator can also select an end effector of the ghost to allow for direct manipulation of the end effector target pose by using interactive markers.

[25] http://wiki.ros.org/octomap (accessed July 30, 2015)
[26] http://wiki.ros.org/interactive_markers (accessed July 30, 2015)

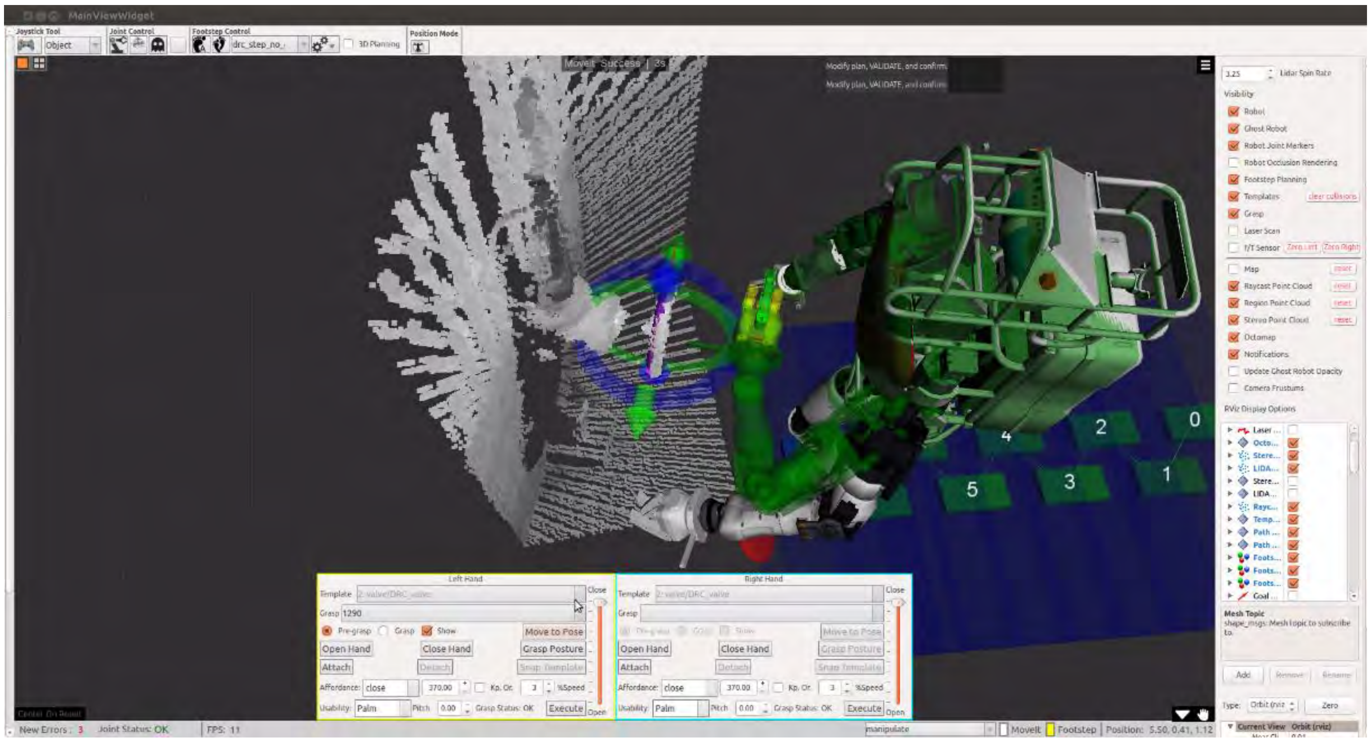


Figure 13. Main View showing the target position of the ghost robot relative to valve template.

3.1.3.2. Map View

The *map view* is a top-down orthographic view widget that is used for navigation and to request more information about the environment. The operator can select a region of interest in the environment by clicking and dragging to create a box selection as shown in Figure 14, and then choose what type of data are needed (e.g., a grid map, LIDAR/stereo point clouds, etc.). Fine control over the amount of data being requested helps in reducing the amount of information transmitted over the network and what is shown on the screen.

The map view provides context sensitive menus for interacting with the footstep planner and footstep execution actions. Figure 15 shows the grid map display on the map view; the grid map is used by the footstep planner as described in Section 3.2.5.

Any sensor or 3D modeling data, including the ghost robot or templates, will be shown projected into the map view by default; these projections can be disabled by unselecting the appropriate item on the right hand side of the map view.

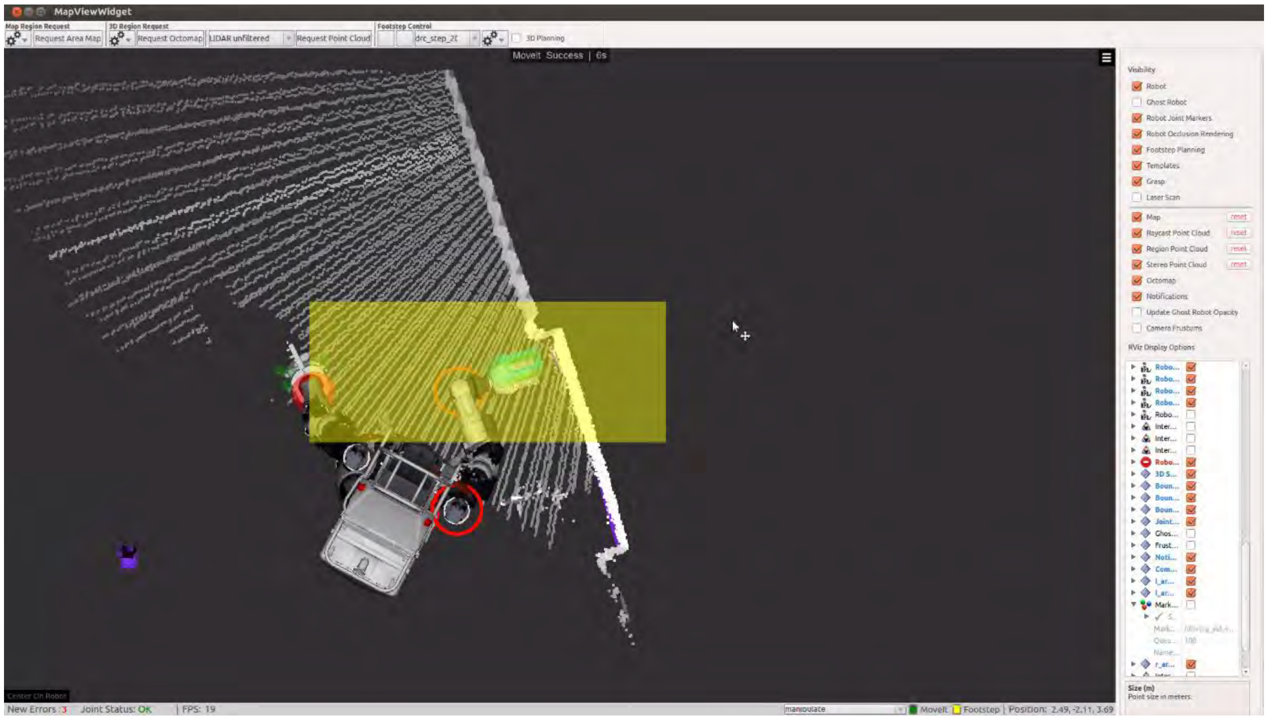


Figure 14. Map view showing region of interest selection

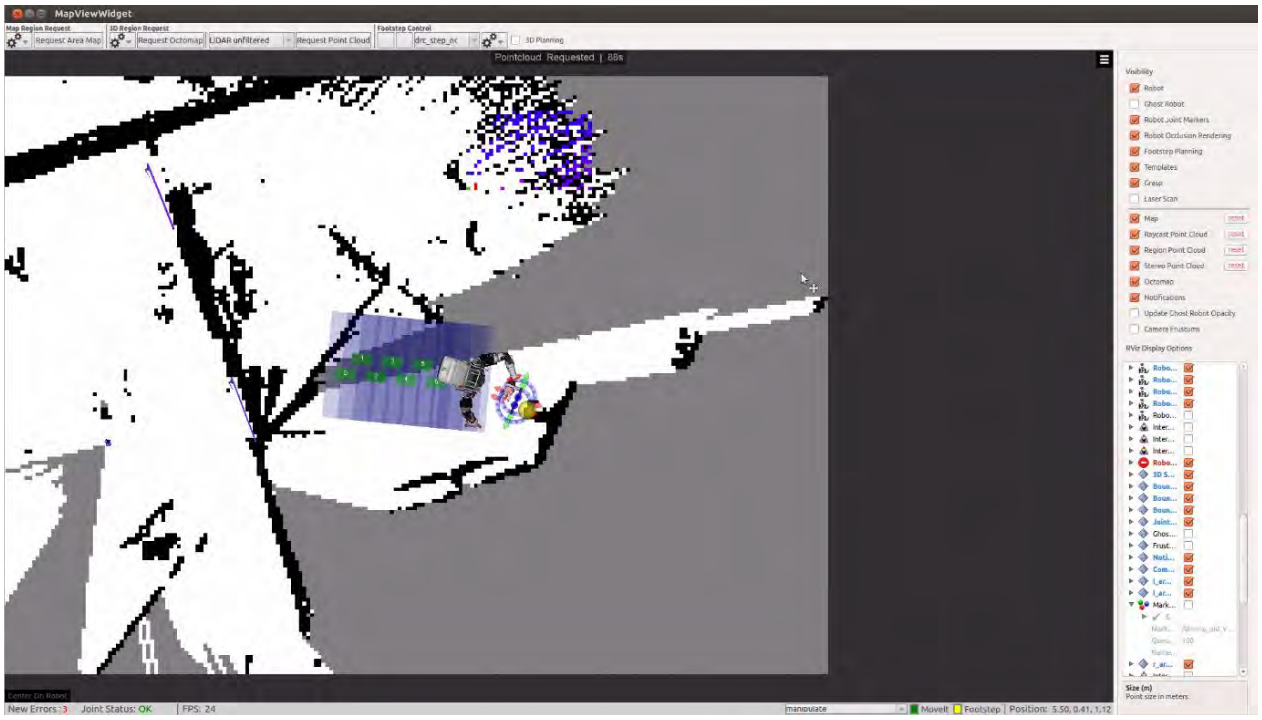


Figure 15. Map view showing the grid map used for footstep planning

3.1.3.3. *Camera View*

The *camera view* allows the operator to request single images or video feeds with varying resolution from every camera on the robot, with up to four images displayed at a time. Three-dimensional data – including sensor data, templates, and robot models – can be overlaid on the images to validate the sensor data and catch errors due to drift in position/orientation estimation. Figure 16 shows an example overlaying 3D point cloud data and a valve template over the main camera feed during the valve task on Day 1 of the DRC Finals; the yellow sphere projected into the image represents the section target corresponding to Figure 12.

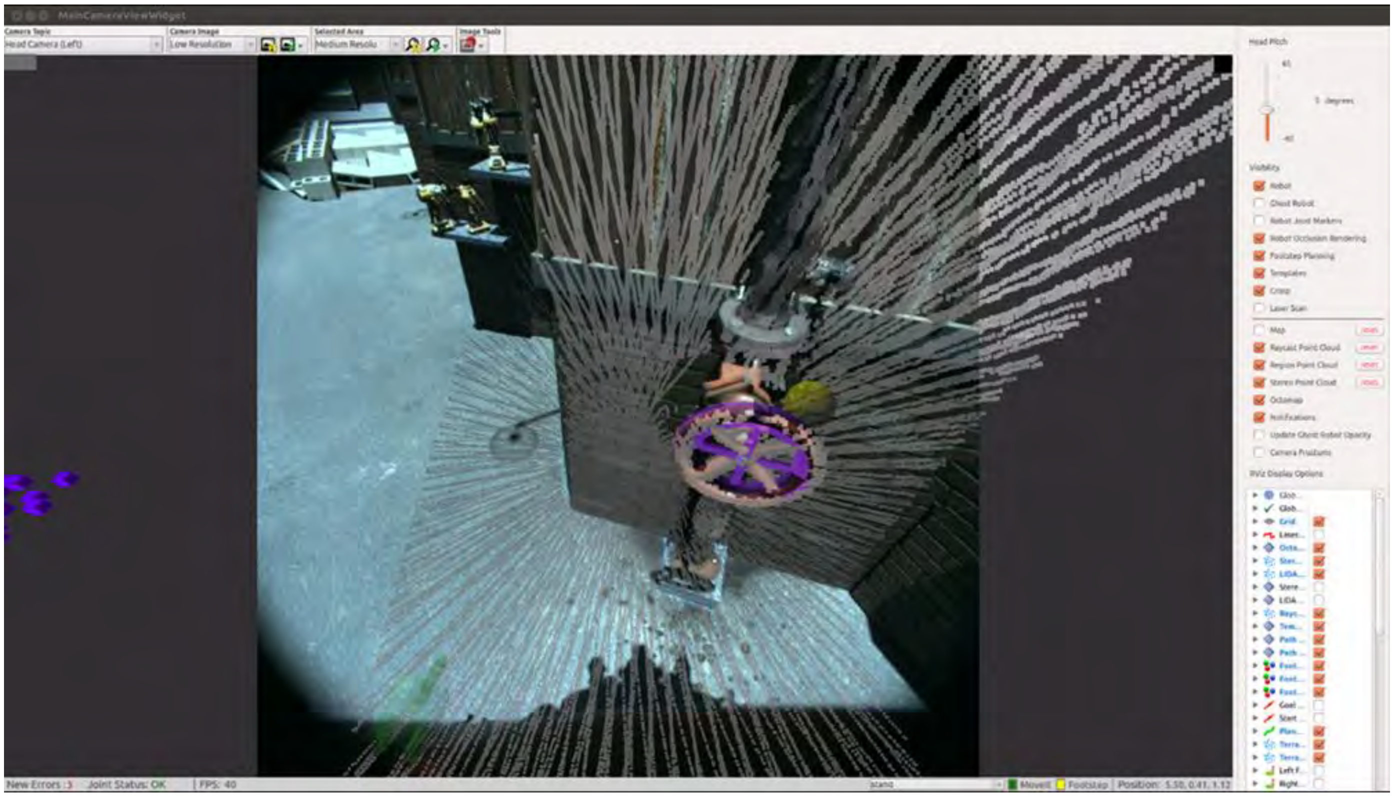


Figure 16. Camera view showing point cloud data and valve template

3.1.3.4. *Behaviors View (FlexBE GUI)*

The Flexible Behavior Engine (FlexBE) discussed in Section 3.2.6, which was developed as Team ViGIR's approach to high-level control, increases the reliability of high-level behaviors by giving the operator a clear understanding of what is happening internally, and allows the operator to intervene as necessary. FlexBE includes an extensive graphical user interface for both development and execution of behaviors as shown in Figure 17.

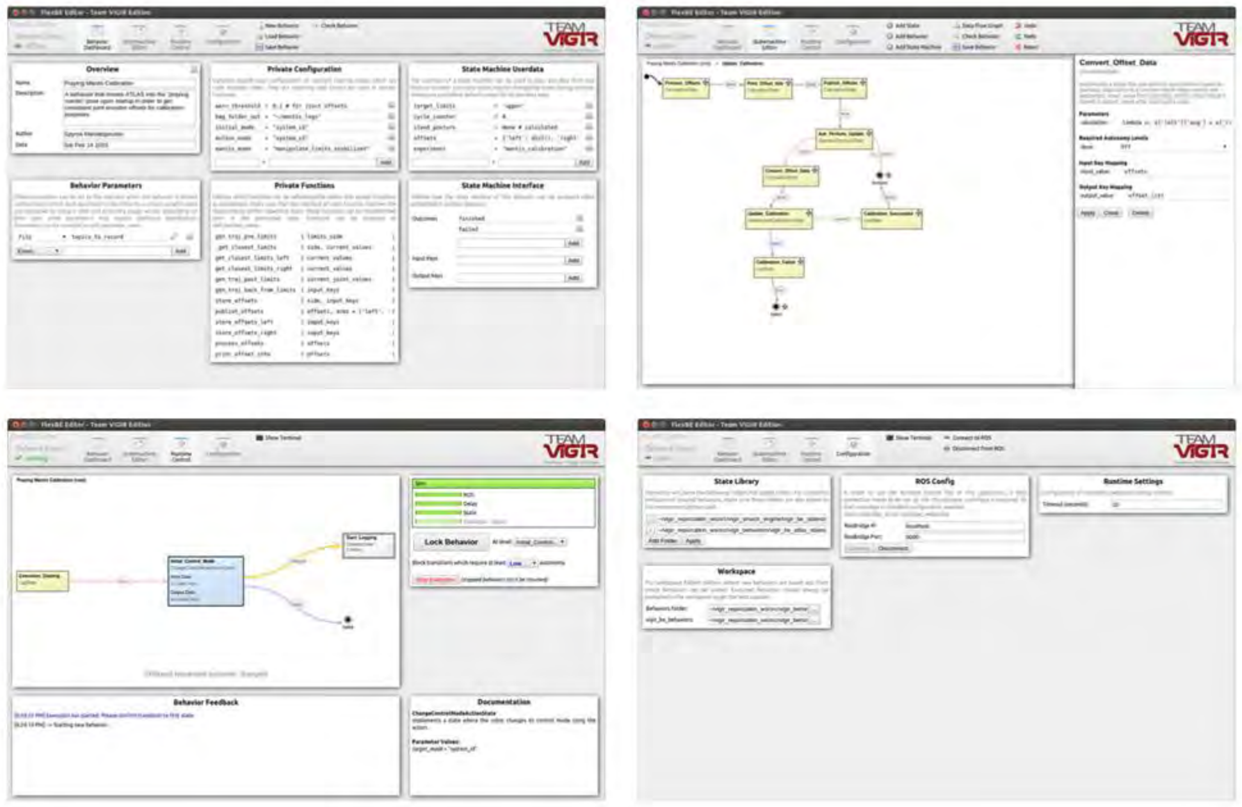


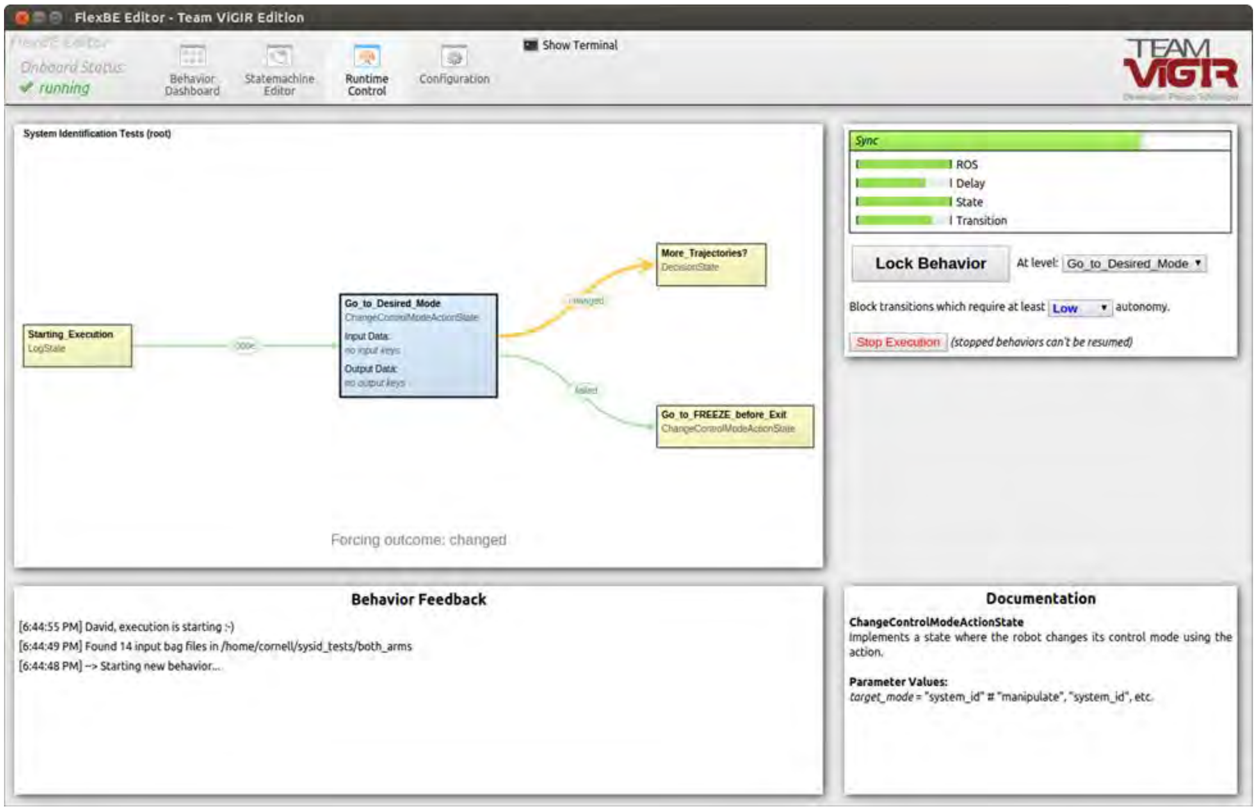
Figure 17. FlexBE, the Flexible Behavior Executive, showing the four primary views.

Clockwise from the upper left these are the: Behavior Dashboard, Statemachine Editor, Configuration view, and the Runtime Control view.

As shown in figure 1, FlexBE's user interface consists of four different views. The first two on the top row are mainly used for development as discussed in Appendix G; the lower right view is just for configuration of the user interface itself. The lower left view is used during robot operation to monitor and control execution of the behaviors in real time.

The Runtime Control view, shown in detail in Figure 18, can start and monitor execution of developed behaviors. When a behavior is running, the view shows the currently active state in the center of its main panel, the previous state at the left, and possible next states at the right. Furthermore, textual feedback is provided as well as again documentation of the active state in order to help the operator to understand what the robot is about to do.

As communications between the OCS operator and the onboard software was subject to delays, the FlexBE user interface included a synchronization status bar This "RC Sync" bar provided a mechanism for monitoring command execution and connection quality between the operator's interface and the onboard behavior engine. As you can tell from the status in Figure 18, the issued transition command is about to get completed while there is a short, but not critical delay in the communication.



Behavior Feedback

```
[6:44:55 PM] David, execution is starting :-)
[6:44:49 PM] Found 14 input bag files in /home/cornell/sysid_tests/both_arms
[6:44:48 PM] --> Starting new behavior...
```

Documentation

ChangeControlModeActionState
Implements a state where the robot changes its control mode using the action.

Parameter Values:
target_mode = "system_id" # "manipulate", "system_id", etc.

Figure 18. FlexBE Runtime Control View.

Forcing the transition "changed" while monitoring behavior execution. Since the robot is in the field, the command cannot be executed immediately due to the communication delay.

Another feature of the Runtime Control interface is the ability to "lock" states to allow for online modification of the behavior. State locking and editing is presented in Appendix G.

3.2. Onboard Systems

3.2.1. Robot Controls and Interface

Team ViGIR developed a custom C++ interface that used ROS ActionLib and ros_controllers to interface the remaining system software to the robot via the BDI proprietary API, and to convert data to/from the BDI data structures. This section discusses the architecture, the approach to joint position control used at the Finals, and implementation of more advanced control strategies.

3.2.1.1. *Interface Architecture*

The *vigir_atlas_controller* interface followed the ROS Control paradigm[27] of a controller manager that invokes controllers that interact with a robot hardware interface. The *vigir_atlas_controller* interface took this one step further and used three instances of the controller manager to guarantee the order of execution.

The first manager handles control mode controllers including the custom controller that accepted mode change action requests and one that handled stability monitoring and fall detection. Team ViGIR extended the BDI control modes (e.g. STAND, WALK, MANIPULATE) to allow multiple modes that specified different combinations of joint controllers and modes. For example, we differentiated between *stand* and *stand_manipulate*, which activated the upper body joint controllers.

The second controller manager interfaced with a number of joint trajectory controllers[28] that handled control of various appendage chains (e.g. left arm, right leg, torso, whole body), and provided the ability to send per joint trajectories to a designated appendage chain. Depending on the particular control mode selected, different controllers would become active with different gain sets selected, as discussed in the next sub section.

The third controller manager handled whole robot behaviors such as footstep control in STEP or WALK, or the compliant controller. The compliant control uses joint targets defined by the joint trajectory controllers.

The *vigir_atlas_controller*, along with the individual controller implementations, can be found in the vigir_atlas_ros_control repository in the software release; this code cannot be open sourced due to the use of BDI proprietary libraries. The package depends heavily on open sourced packages in the vigir_ros_control repository, which provides the structure for the three controller managers, and loading the Gazebo simulation robot model into a dynamics model[29] that is used for kinematics and dynamics calculations for the controllers. See the package source code for more details and Appendix J for usage guidelines.

3.2.1.2. *Joint Position Control*

The *vigir_atlas_controller* interface used the ROS joint trajectory controllers to accept FollowJointTrajectory[30] actions using the ROS trajectory_msgs/JointTrajectory.msg[31] format. The controller interpolates the trajectory commands to yield an instantaneous joint position command. This is used to calculate the servo valve commands using a combination of encoder-based PID control, and the embedded BDI actuator based position control.

[27] http://wiki.ros.org/ros_control (accessed July 30, 2015)
[28] http://wiki.ros.org/joint_trajectory_controller (accessed July 30, 2015)
[29] https://bitbucket.org/rbdl/rbdl/ (accessed July 30, 2015)
[30] http://wiki.ros.org/joint_trajectory_controller (accessed July 30, 2015)
[31] http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectory.html (accessed July 30, 2015)

Each per appendage chain controller used a customized version of the PID controller from the ROS control_toolbox[32]. The customized version in our fork includes an integral reset when the controller is activated to provide bumpless control based on the old joint command. This PID uses the encoder based joint position estimates for more accurate positioning; the interface passes the controller output to the robot in the *ff_const* term used by BDI[33].

To provide faster response, and robustness to variations in the communications, the interface also makes use of the embedded PD joint position controller provided by BDI. These gains are set based on the desired control mode, and passed to the robot each time step. The controller tracks the offset between the actuator based position estimate and the encoder based position estimate, and adds the offset to the embedded joint position command to maintain consistency with the trajectory command.

After calibration, this combined approach proved reliable and was used at the DRC Finals.

3.2.1.3. Advanced Control

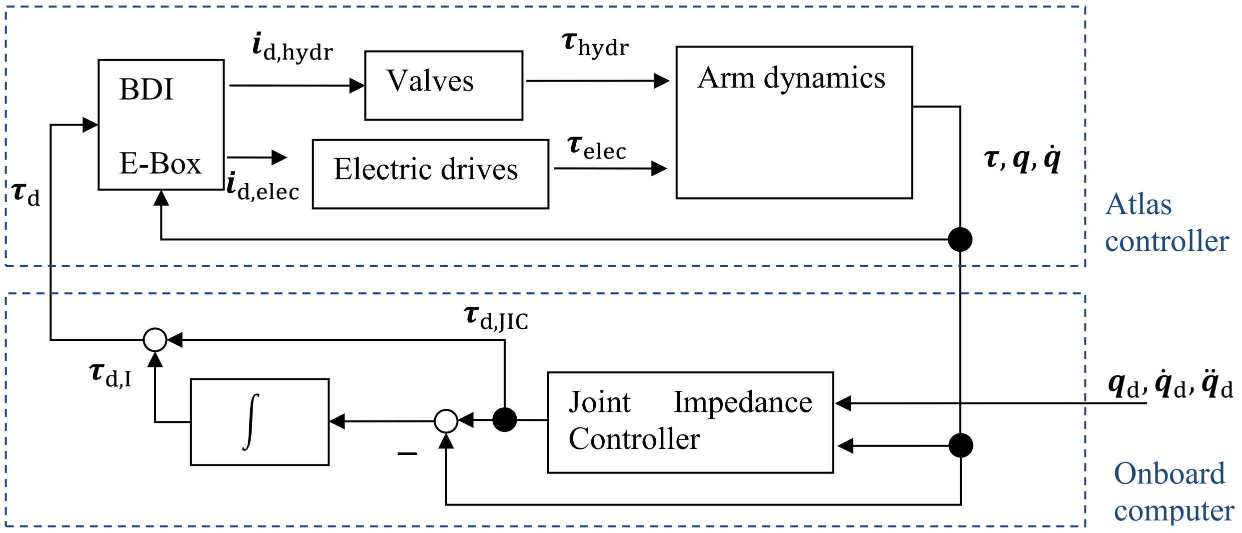
From a theoretical point of view, there is no exact model-based feedforward or feedback compensation possible with the above joint position control scheme, since the hydraulic arm joints are commanded at hydraulic current level, which is equivalent to the joint velocity. Model based calculations give joint torques, so the addition of these quantities does not result in a physically feasible model of the controlled system, unlike for example for electric motors, where the commanded value is also the motor torque or the equivalent electric current.

Further, the disadvantage of the PD position control is that a good position accuracy can only be achieved with high parameter gains, where the robot is not compliant and collisions often result in a robot fall due to high contact forces.

To overcome these disadvantages for the arm control, we investigated and implemented a model based controller concept called joint impedance control. While this approach was not used during the DRC Finals due to some lingering issues, we present it here for completeness; Section 4.2.1 presents our post-Finals experimental results.

Joint impedance control uses a cascaded control scheme consisting of an inner joint torque loop $(\boldsymbol{\tau}, \boldsymbol{\tau}_{\mathrm{d}})$ with an outer PD position control $(\boldsymbol{q}, \boldsymbol{q}_{\mathrm{d}})$ with variable damping gains and model based compensations as seen in Figure 19. This controller is configured with the more intuitive parameters joint stiffness for position tracking and modal damping coefficient for velocity tracking and interaction behavior. For the explicit formulation, see Appendix D.

[32] http://wiki.ros.org/control_toolbox (accessed July 30, 2015)
[33] Boston Dynamics Atlas Robot Software and Control Manual, ATLAS-01-0019–v3.3, pg. 19



$\boldsymbol{i}_{\mathrm{d,hydr}}$
$\boldsymbol{\tau}_{\mathrm{hydr}}$
BDI
E-Box
Valves
Arm dynamics
Electric drives
$\boldsymbol{\tau}_{\mathrm{elec}}$
$\boldsymbol{i}_{\mathrm{d,elec}}$
$\boldsymbol{\tau}_{\mathrm{d}}$
$\boldsymbol{\tau}, \boldsymbol{q}, \dot{\boldsymbol{q}}$
Atlas controller
$\boldsymbol{\tau}_{\mathrm{d,JIC}}$
$\boldsymbol{\tau}_{\mathrm{d,I}}$
$\int$
Joint Impedance Controller
$\boldsymbol{q}_{\mathrm{d}}, \dot{\boldsymbol{q}}_{\mathrm{d}}, \ddot{\boldsymbol{q}}_{\mathrm{d}}$
$-$
Onboard computer

Figure 19. Block diagram of the Joint Impedance Controller control scheme

In Figure 19, on the side of the onboard computers, we first denote the arm joint torque $\boldsymbol{\tau}$, the joint positions and velocities $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ from the Atlas API and the desired joint trajectory $\boldsymbol{q}_{\mathrm{d}}, \dot{\boldsymbol{q}}_{\mathrm{d}}$ from the onboard trajectory planning from MoveIt! as an input to the joint impedance control scheme. Further, we denote the desired joint torque calculated by the impedance controller algorithm $\boldsymbol{\tau}_{\mathrm{d,JIC}}$, the added desired joint torque of the integral term $\boldsymbol{\tau}_{\mathrm{d,I}}$, and the resulting desired joint torque $\boldsymbol{\tau}_{\mathrm{d}}$ commanded to the BDI E-Box through the Atlas API. On the other side, the desired joint torques are calculated into the desired electric current controlling the hydraulic valves $\boldsymbol{i}_{\mathrm{d,hydr}}$ and desired electric motor current $\boldsymbol{i}_{\mathrm{d,elec}}$ in the corresponding actuators. From these inputs, the actuator dynamics give the actual hydraulic and electric joint torques $\boldsymbol{\tau}_{\mathrm{hydr}}$ and $\boldsymbol{\tau}_{\mathrm{elec}}$. This internal process is not covered in our scheme.

For the inner joint torque loop we discovered, that the proportional joint torque control based on the hydraulic pressure in the valve has a high steady-state error which directly results in position errors. We implemented an outer integral loop for the joint torque to increase the joint torque tracking performance. Appendix D shows the detailed results of the influence of the integral gain.

Figure 19 explicitly emphasizes the location of the implementation of the different control blocks. This has a strong influence on the stability, since the communication from the BDI E-Box to the onboard computers running our custom code suffered from a 2-3ms time delay. Presumably due to this delay, only lower damping coefficients compared to other impedance controller implementations (e.g. in Hannover robotic labs) leads to a stable behavior in all robot states.

The dynamic arm model we used consisted of inertial, centrifugal, coriolis and gravitational forces and a viscous and Coulomb friction model; therefore, we only neglected the torso movement and the complexity of the friction on the real system.

3.2.2. Perception

The perception system is responsible for gathering data from the onboard sensors, and making the data available to the operators and planning systems. For the Atlas robot, the sensors included an inertial measurement unit, joint state measurement, and the integrated Multisense stereo camera and LIDAR sensor.

A system wide overview of our perception system in given in [1] in Appendix A; this subsection discusses the major upgrades to this system for the DRC Finals.

3.2.2.1. State Estimation

The BDI API provides both internal state estimates for the robot joints as well as an estimate of the robot pose relative to a fixed frame with the origin relative to the pose where the robot has been switched on. The estimate is based on proprioception and IMU sensing. The internal system uses knowledge of the standing foot for forward kinematics based motion estimation that is fused with IMU data.

This state estimation system provides sufficient performance for many applications such as stepping and walking on flat ground. When stepping over rough terrain, however, even slight drift by a few centimeters can result in the robot falling. Team ViGIR and other teams identified this shortcoming during the DRC Trials [1]. To reduce drift, we switched to using MIT's pronto[34] state estimator, which exhibits lower drift due to improved forward kinematics estimates. In principle, pronto can completely eliminate drift by using the LIDAR sensor for external sensing; we opted not to use LIDAR-based corrections during the DRC Finals, as the external sensing approach used in pronto relies on a static world assumption. This could be violated during a competition run due to moving people, equipment or other unmodeled motion in the environment of the robot.

3.2.2.2. Constrained World Modeling

To effectively leverage the human operator's cognitive and decision making capabilities, a state estimate and world model must be made available over the constrained bandwidth link between robot and operator. With ATLAS onboard sensors providing data at a rate in excess of 100 MB/s compression is both crucial and a significant challenge.

The (communication) constraints under which the perception system has to work changed over the course of the competition as follows:

- In the VRC competition, a bandwidth budget for communication between robot and operator was allocated for each mission and communication was cut off after the budget was exceeded
- In the DRC Trials, communication was constrained by limiting bandwidth and introducing latency, alternating between a "good comms" and "bad comms" setting.
- In the DRC Finals, 3 communication channels were used, one 9600 baud line from robot to operator, one 9600 baud line in the opposite direction and one high bandwidth connection that is blocked for a period of 10-30 seconds

Team ViGIR designed the perception system to provide situational awareness and state estimation for the operator under all of these conditions. To achieve reliable and efficient manipulation with a remote operator in the loop, 3D geometry data is crucial. This data is compressed and handled by the

[34] https://github.com/ipab-slmc/pronto-distro

Worldmodel Server, which aggregates 3D data from the Multisense LIDAR and makes it available via a ROS interface that allows for the selection of regions of interest, aggregation history size and filtering parameters.

In the case of available bursty communication, two instances of the world model server are used, one for the onboard/robot side and one for the OCS side. As direct transmission of point cloud data is impossible, specialized processing on LIDAR data is performed to make each packet compact enough to fit within a standard 1500 Byte limit and compress it as to be able to transmit a maximum of data during a communications burst. Direct transmission of point cloud data generated onboard the robot would cause prohibitive bandwidth cost as the point cloud representation with at least three floating point values for each Cartesian point is not a compact one . For this reason, the natural and compact representation of a laser scan as an array of range values is used instead. To fully reconstruct the 3D geometry captured by a single scan, a high fidelity projection of the scan has to be performed however, taking into account motion of the LIDAR during the data capture process. If this motion is not considered, scan data shows visible skew and ghosting (double walls) when converted to a point cloud. We thus use the following approach:

- Perform a 3D high fidelity projection onboard the robot and perform self-filtering
- Compress the scan data by writing the range values to a uint16 array representing millimeters and also encoding self filtering information. Threshold and map intensity information to a uint8.
- Add information about the scanner transform in world frame, one transform for the start of the scan and one for the end.
- Split the compressed scan into chunks that are small enough to be compressible to less than 1500 Bytes.

On the OCS side, the compression process is reversed and resulting scan data is used to update the OCS world model. This approach improved consistency of the data.

The size of a LaserScan message is dominated by the range and intensity fields, with a Hokuyo UTM30LX-EW providing 1080 measurements per scan. For compression, float value ranges are converted to millimeters and stored in a unsigned 16 bit number.

Self-filtering of robot parts from LIDAR data requires knowledge of the whole transform tree of the robot and thus has to be performed on the onboard side if transmission of high bandwidth transform data to OCS is to be avoided. Per default, self-filtering is thus performed onboard and compressed laser scan data is annotated with a bit per scan point indicating if it belongs to the robot.

Intensity data is converted from float to a unsigned 8 bit number. Here, a loss in fidelity is acceptable as intensity is mainly used for visualization and a range of 2^8 values is sufficient.

Table 1. Comparison of message sizes for laser scan representations

	Standard LaserScan [Bytes]	Localized LaserScan [Bytes]	Compressed Localized LaserScan [Bytes]
header	>~ 16	-	-
metadata	7 * 4	-	-
ranges	4 * 1080	2 * 1080	< ⅓ * 2 * 1080
intensities	4 * 1080	1 * 1080	< ⅓ * 1080
total	8684	3240	< 1080

With the bandwidth constraints encountered at the DRC Finals, the transmission of geometry data was not feasible when the high rate data line was blocked. For this reason, the operator(s) had to rely on previously transmitted data during the outage period. The system updated robot state information over the 9600 baud line, which allowed the operator to see robot motion relative to previously retrieved 3D geometry data.

3.2.2.3. Textured Meshes

To provide the highest practically possible amount of fidelity for this 3D geometry data, Team ViGIR developed an infrastructure for generating textured meshes out of both LIDAR point clouds and stereo camera based depth images. Compared to plain point cloud visualization, Figure 20 shows that this approach allows for a clear view of geometry and texturing of mesh surfaces, which is more intuitive for scene understanding.

ATLAS cannot perform rotation of the Multisense sensor head around the yaw axis, greatly limiting the field of view of the main sensor system. Prior to the ATLAS v5 arm upgrade, this issue was much more severe, as the volume of good manipulability for the arms was outside the Multisense field of view. To remedy this issue, Team ViGIR developed a system for rectification the Fisheye lenses of the SA cameras using a ROS integrated version of the OCamLib library[35]. This allows generating novel rectified views from fisheye images not exhibiting severe distortion that otherwise makes judging of spatial relations difficult for operators; See Figure 21 for an example. With the better arms of the ATLAS v5 version and the relocation of SA cameras from the chest to the upper head, this functionality was deemed less crucial and integration for ATLAS v5 was skipped.

[35] https://sites.google.com/site/scarabotix/ocamcalib-toolbox

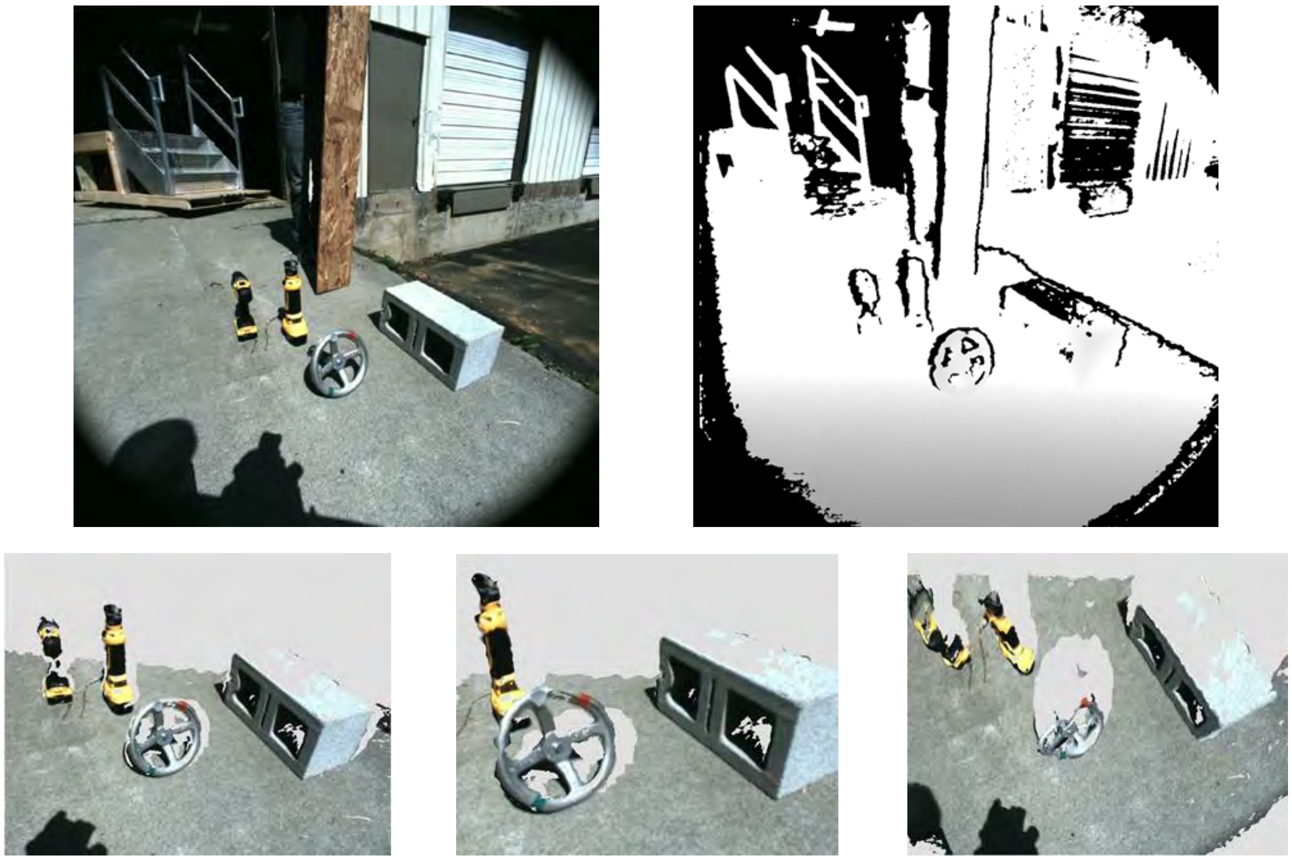


Figure 20. Mesh-based Visualization.

Top row: RGB and stereo-based depth image; bottom row: three novel views of the textured mesh

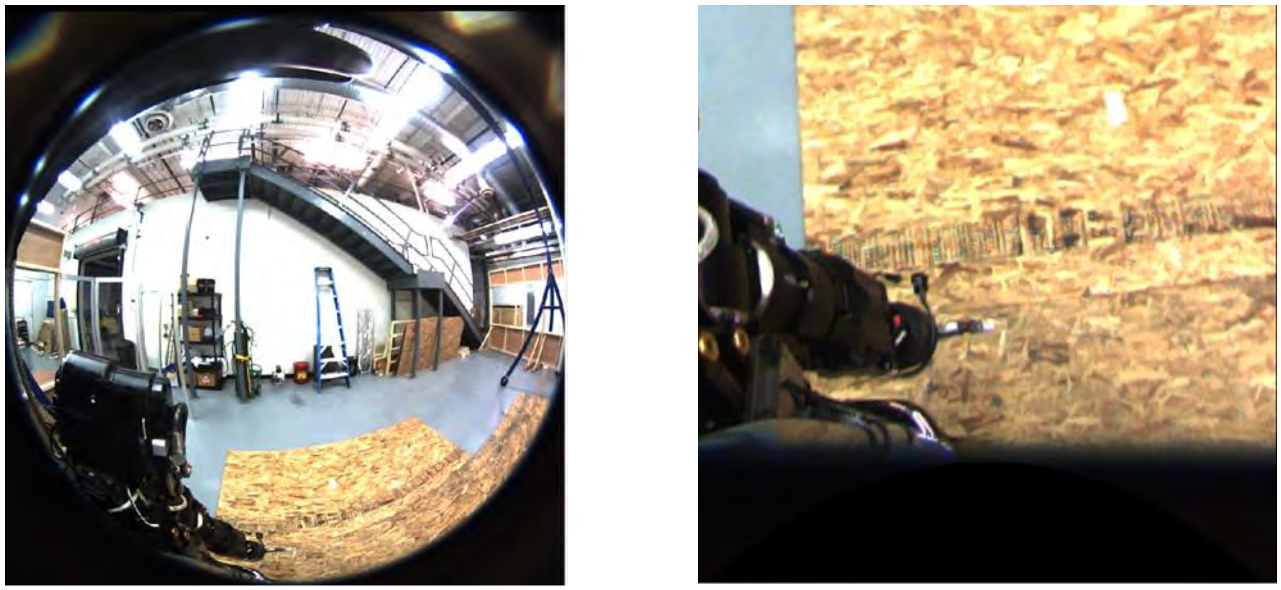


Figure 21. Fisheye Camera Rectification.

Distorted fisheye image (left). Rectified image close demonstrating a virtual ideal pin-hole camera (right).

3.2.3. Motion Planning

The motion planning system provides the backend that allows the system to perform complex joint motions in a reliable and intuitive fashion as is necessary for manipulation tasks. Given the unstructured nature of disaster environments, automated collision avoidance is a desirable capability as it allows to significantly reduce the workload for the operator and is required for carefree task-based planning. After an evaluation of existing approaches, Team ViGIR chose to base its motion planning system on the MoveIt! planning system, which is integrated with ROS. Full ROS integration, an active user community, and capability of real-time obstacle avoidance were reasons for the selection of MoveIt!. A comprehensive overview of development up to the DRC Trials is available in [1].

3.2.3.1. Planning Backend

To allow for reliable manipulation, the MoveIt! API was used and DRC-specific capabilities were implemented in a separate move_group capability plugin. This offered the advantage of retaining standard MoveIt! library planning features, while simultaneously allowing the development of extended capabilities specific for DRC tasks.

With limited reachability, especially before the ATLAS v5 upgrade, it often was desirable to provide the capability to plan with torso motion as to compensate for limited arm reachability. Restricting the range of motion of single joints is not an intended use case with MoveIt!, so this capability was added additionally.

Per default, trajectory execution speed could not be changed online. Instead, trajectories would always be time parametrized according to the velocity limits supplied in the robot model (URDF) file. To allow for changing the execution speed online, a velocity scaling factor has been introduced that can be set on a per motion plan request basis. This addition has already been merged into standard MoveIt!.

An iterative parabolic time parametrization approach is used as the standard approach for generating trajectories per default. During experiments on Atlas, this approach was shown to produce significant velocity and acceleration spikes, resulting in jerky arm motion due to the splines that were defined between knot points. The default time parameterization was changed to do a velocity scaling iterative parabolic calculation, followed by a recalculation of the interior velocities and accelerations assuming piecewise quantic splines with continuous velocity and acceleration at the knot points. This resulted in smoother motions.

The planning system is exposed via a ROS Action server interface and thus provides feedback about the planning and plan execution process. The Action interface is the sole entry point for requesting and executing motion plans and is used for (in order of increasing autonomy) tele-operation, affordance-based manipulation planning, and for motion plan requests generated by the behavior executive. For tele-operation, an onboard node translates compressed and compact motion requests by the operator into an Action request that then gets forwarded to the planning system.

While the default motion planning system performs well for "standard" manipulation tasks requiring only upper body motion, sampling based planning falls short for planning whole body motions that require the consideration of balance constraints. To support this need, Team ViGIR integrated the optimization-based

Drake[36] planning approach developed by MIT. The choice to use either the default sampling-based planning approach or to use Drake is specified by the plan request. Drake has also been integrated with the “ghost robot” on the OCS side and the operator can use Drake-based whole body inverse kinematics to pre-plan tasks like reaching towards the ground for picking up objects (see Figure 22). As this capability was not required during the DRC Finals, it was not used there.

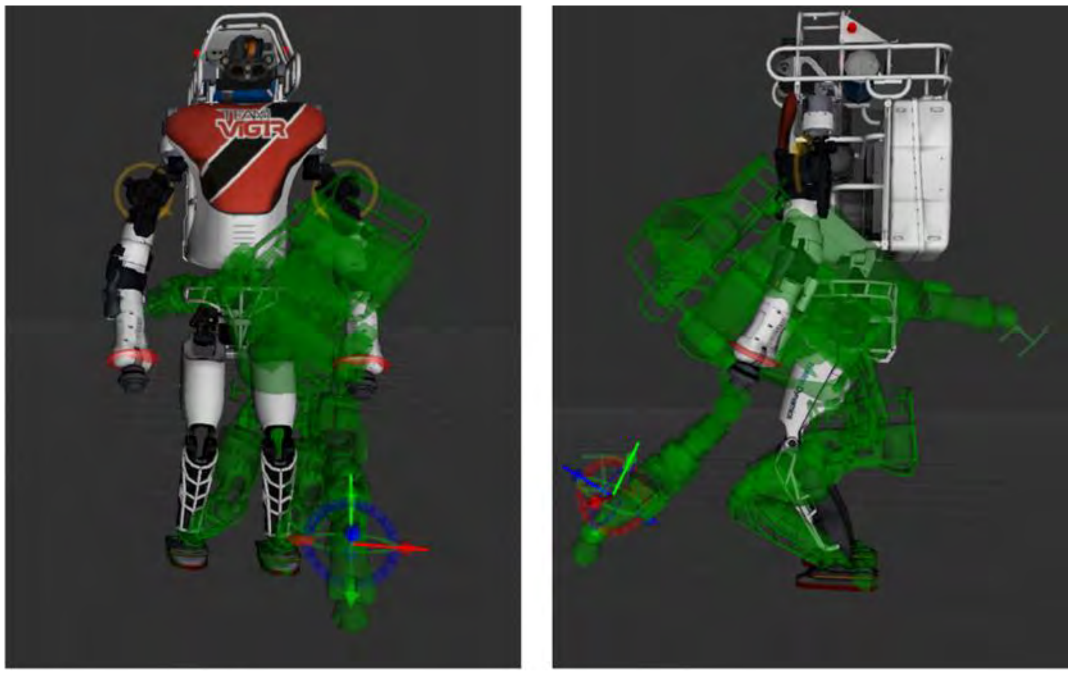


Figure 22. Using Drake inverse kinematics for reaching down to the ground with the “ghost robot”

3.2.4. Manipulation

Team ViGIR focused on developing a manipulation approach that will allow the operator and the robot to cooperate and perform efficient high-level interaction with the remote environment. This approach is based on the concept of Object Templates[37] (OT); see [3] in Appendix E. An OT is a 3D mesh in a virtual environment that is augmented physical and semantic information related to the object of interest that it visually represents. An operator inserts the OT into the OCS scene, and manipulates the template to align with sensor data that corresponds to the real object. Once an OT is aligned, its specified 3D position can then be used to perform locomotion to approach to it and arm motion planning to grasp and manipulate the real object.

3.2.4.1. Affordances

We based our approach on the concept of affordances, which are the possibilities of action that an object in the environment offers. In the current state of the art, several teams converged to a similar affordance

[36] https://github.com/RobotLocomotion/drake
[37] The term object template can also be found in this report as, e.g. "valve template" to refer to specific objects or just "template" if the object is already implied.

based manipulation approach (MIT, IHMC, NASA). These three teams for example, use their OT to provide potential grasp poses to the operator as well as information about manipulation standing positions. They are also used to generate end-effector trajectories when objects are grasped, e.g. when they want to turn the valve, they manually rotate the OT in their user interface and send the generated trajectories to the robot.

In contrast, the approach developed by Team ViGIR goes beyond the state of the art because it presents the operator the affordances of the object; see Appendix E for more details. In addition to being used for standing poses and grasp poses, the OT internally defines the motions that the object offers and allows the operator to easily select the required affordance (e.g. selecting and clicking the Turn affordance) (see Figure 23). The OT provides the necessary information regarding path constraints that enable the planning software to generate the desired trajectories and perform the manipulation motion using the motion planning capabilities presented in Section 3.2.3.

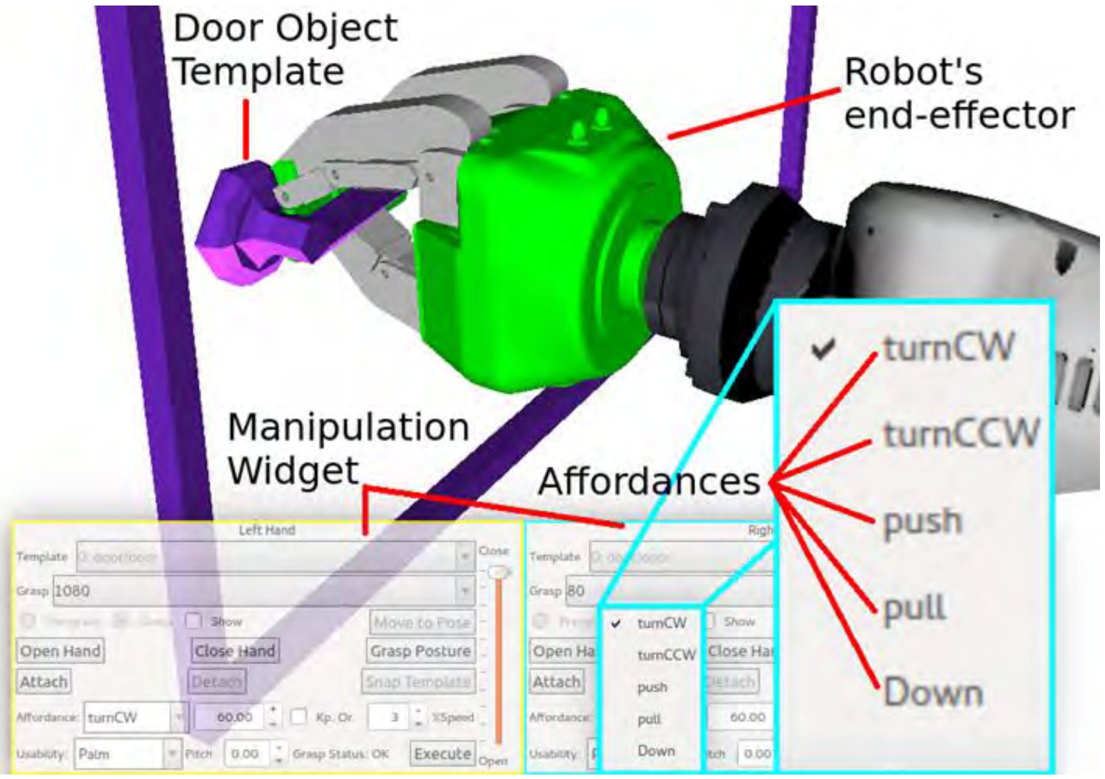


Figure 23. The Object Template of a door being grasped by the robot's end-effector.

The Manipulation Widget is shown for both hands (left is yellow and right is cyan). The affordances combo box is zoomed in to show the available motions of the door, e.g. turn Clockwise (CW) or turn counterclockwise (CCW) as well as pushing and pulling, among others.

3.2.4.2. Object Template Library

The manipulation tasks during the VRC and the DRC Trials were well defined and the objects required to manipulate were known a priori. Nonetheless, Team ViGIR created an Object Template Library (OTL) that can include any number of objects. This accounts for potential unknown objects that might be available in a disaster scenario; similar to the surprise tasks presented during the DRC Finals. The OTL is divided into three blocks of information: the object library (physical and semantic information), the grasp pose library (end-effector grasp pose information), and the stand pose library (robot stand pose information). The grasp pose library and the stand pose library have a relationship of many to one with the object library. Each object in the object library has a unique type that is used to relate one or many grasps to one OT as well as for stand poses. An entity-relationship model using Crow's foot notation[38] can be seen in Figure 24.

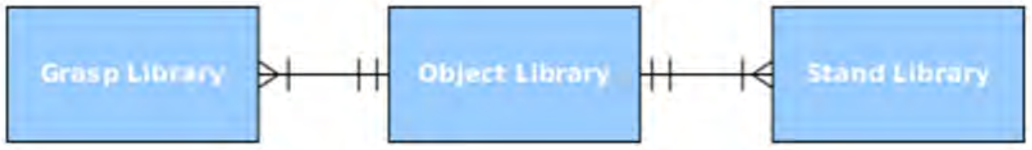


Figure 24. Relationship between objects, grasps and stand poses libraries using Crow's foot notation

3.2.4.3. Object Template Server

The Object Template Server (OTS) implements the Object Templates concept. The OTS is responsible of loading and providing OT information to any client that requested it. For example, the Main View widget will request 3D geometry mesh information from the object template to display, as well as finger joint configuration while displaying potential end-effector poses to grasp such object. Other clients such as the Manipulation Widget (Figure 23) could request grasp information and affordance information from the OTS. Additionally, Section 3.2.6 describes how the autonomous behaviors use the OTS provided information.

Given the network setup constraints on the DRC, the OTS was required to provide information for both, the OCS side and the Onboard side. In the OCS side, the OTS provides information to all the widgets that use OTs. It also manages the instantiated OT that the operator has inserted in the 3D environment. To replicate the same status in the Onboard side, another instance of the OTS is created in the Onboard side. The OTS in the Onboard side is responsible of keeping OT information to be considered for motion planning, e.g. as collision objects or attached collision objects to the robot. Both OTS were kept synchronized through the Communications Bridge; in case there was any synchronization issue, both OTS are re-synchronized by instantiating a new OT. The architecture of the OTS can be seen in Figure 25.

[38] Crow's foot notation: http://tdan.com/crows-feet-are-best/7474 (accessed July 30, 2015)

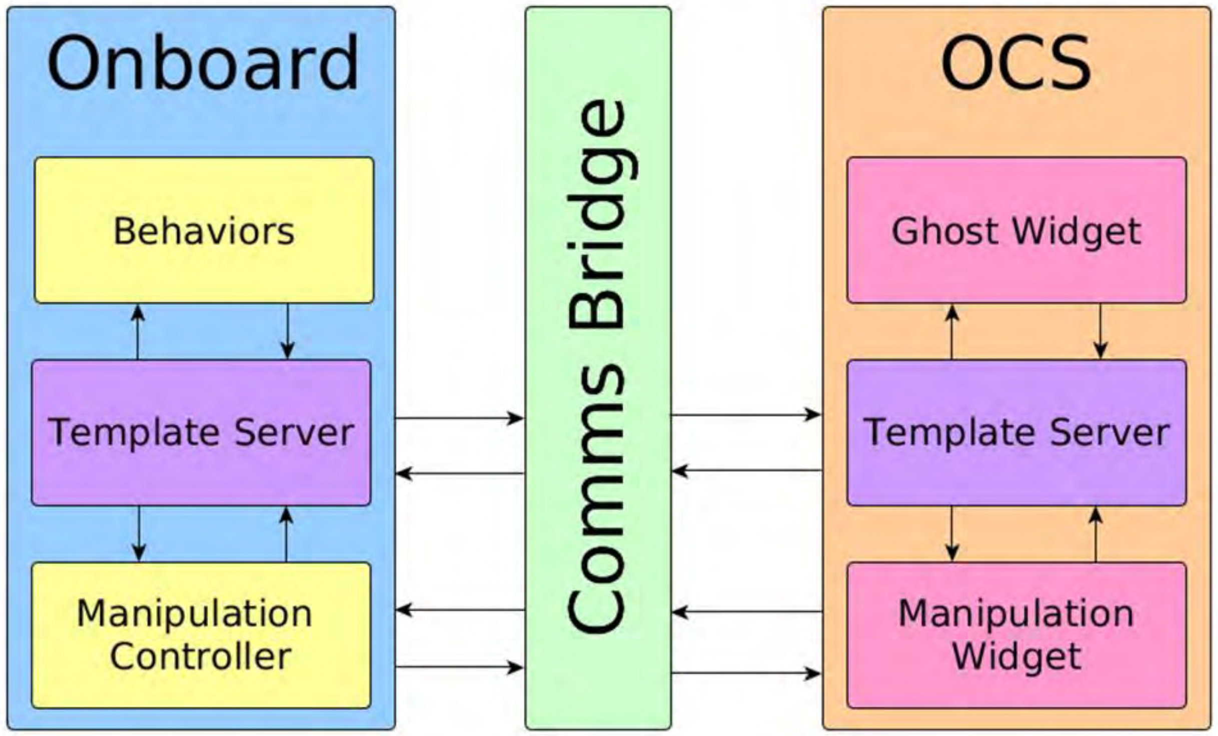


Figure 25. Object Template Server communication concept.

Object Template Server (purple) is instantiated in both, OCS (orange) and Onboard (blue) sides. Each OTS provides information to the controller blocks in Onboard (yellow) and to the user interface widgets in the OCS (pink). Additionally, both OTS are kept synchronized through the communications bridge (green).

3.2.5. Footstep Planning

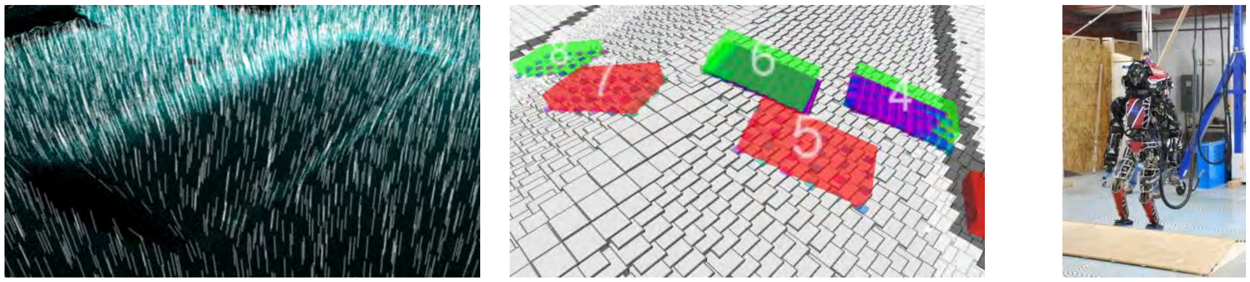
A key challenge of the DRC was enabling the robot be able to tackle locomotion tasks such as the traversal of sloped stairs, ramps and rubble. While Team ViGIR depended on the BDI footstep controller for stepping and stability, the specification of footstep placements remained a significant challenge; Team ViGIR extended an existing planner for 2D environments to handle this more complex 3D terrain.

The footstep planner has to satisfy two main capabilities: The planner has to solve the navigation problem of finding the shortest safe path in a given environment. Secondly, it has to generate a feasible sequence of footstep placements, which can be executed by the robot with minimal risk of failure. Additionally, the DRC competition discouraged the use of slow footstep planning approaches due to mission time limits. Here, operator performance highly depends on the speed and performance of the used footstep planning system, so planning efficiency becomes important. It is desirable that the planning system provides all parameters of the walking controller for each step, so that the complex low-level walking controller interface is completely hidden from the operator to reduce the chance of operator error. Our footstep planning approach satisfies these needs, and requires the operator to only provide a goal position to start planning.

Footstep planning systems have not been applied to human-size real robots in complex terrain scenarios such as the DRC before. Although the increased size of the humanoid robot enhances the locomotion versatility, dynamics have a larger impact on the robot system, making stability control challenging. Therefore, the footstep planner has to trade-off the versatile locomotion capabilities and risk of falls; This is difficult given the lack of detailed knowledge or feedback of the underlying walking controller.

The DRC tasks required the capability to solve difficult terrain traversal tasks in full six Degrees of Freedom (DoF). As a suitable implementation was not readily available, we decided to extend significantly an existing open source footstep planning approach for flat surfaces. We have chosen to extend the approach of Garimort and Hornung[39] as it already was available for ROS and is based on the proven search-based ARA* (Anytime A*) planning algorithm delivering the best solution within a specified time limit. As the robot operates on state estimates based on noisy sensor data, there is no huge benefit of having the global optimal solution at all. Therefore, the operator may be satisfied with a suboptimal solution, which is close to the global optimum, but can be found in significantly shorter time.

Prior to the DRC Trials we have introduced the first search-based footstep planner capable of generating sequences of footstep placements in full 3D under planning time constraints and using an environment model based on on-line sensor data. The planner solves the navigation problem of finding shortest paths in difficult terrain scenarios while simultaneously computing footstep placements appropriate for BDI's walking controller. The planner comes with an improved 3D terrain generator which is recently able to generate terrain models for the footstep planning system on-line (see Appendix F). It is able to efficiently compute the full 6 DoF foot pose for foot placements based on 3D scans of the environment. This new terrain model generator has already been applied and validated successfully for real world scenarios. In addition, our novel collision check strategy based on ground contact estimation allows the planner to consider overhanging steps which enhances significantly the performance in rough terrain scenarios. Figure 26 shows a real world example of the entire footstep planning pipeline consisting of perception, planning and execution. More detailed information about this approach is available in our published work [4] and [1].



Terrain map showing surface normals **Generated footstep plan on OCS** **Execution by the real robot**

Figure 26. Footstep Planning Pipeline

After the DRC Trials, the footstep planner was refactored into a complete robot agnostic footstep planning framework that could be used by variety humanoid robot systems including those of Team VALOR, and eventually Team Hector. Our main objective is to provide a versatile and highly capable footstep planning framework using ROS, while at the same time retaining the ability of integration and expandability. Users of the framework only have to implement and extend robot specific functionality to interface with the planner. Already implemented, tested, and proven algorithms can be left untouched to decrease the possibility of error.

[39] http://wiki.ros.org/footstep_planner

The footstep planning framework is based on a versatile plugin and parameter management system. Plugins have been added for all points where the user might want to take influence on the planner's behavior (see Figure 27). These plugins allow efficiently adding custom code into the planning system without any modification to the framework itself. The plugins are maintained by a dedicated plugin manager was written which is used to obtain efficiently all available plugins filtered by their semantic functionality. Details about the entire plugin system are provided in Appendix F.

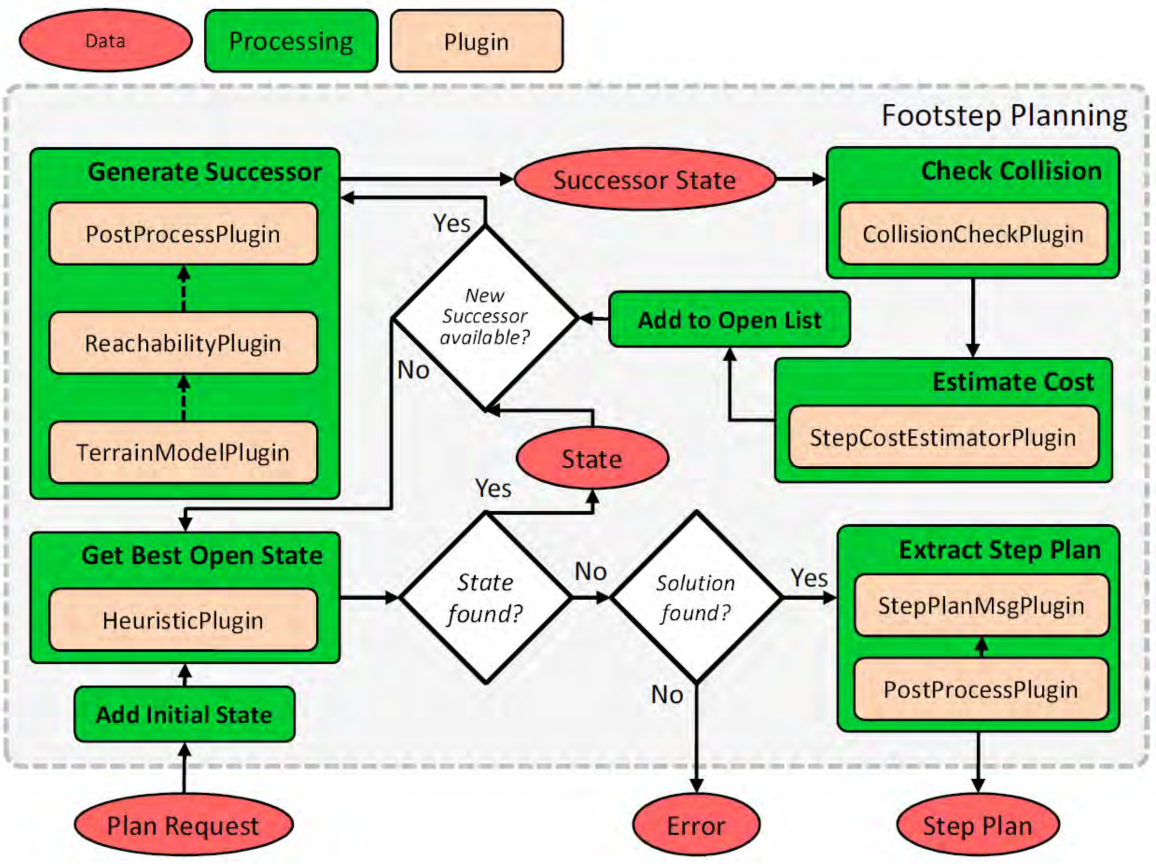
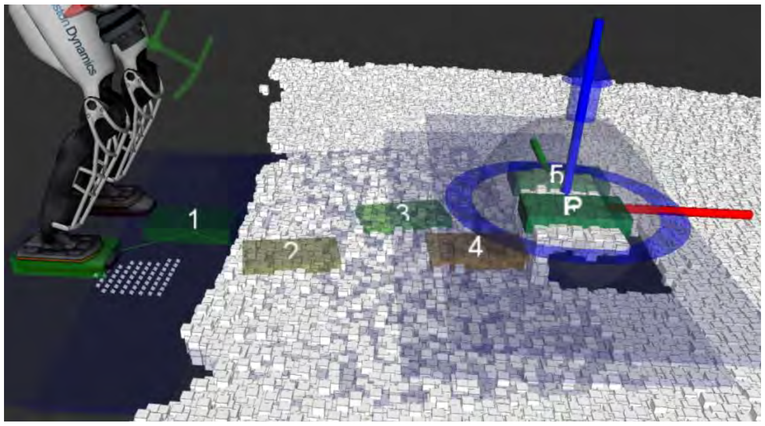


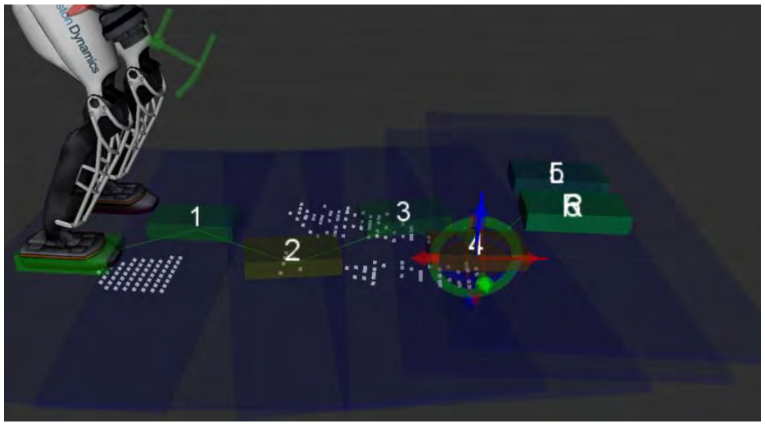
Figure 27. Advanced footstep planning system architecture
Simplified illustration of the footstep planning pipeline showing where plugins can be used to affect the planner's behavior.

As all user created code needs usually their own parameters to run correctly, a parameter management system has been introduced as well. This system is able to overcome the basic conflict of rigid message types needed by ROS for interprocess communication and the need of flexible content of parameter sets due to user defined parameters (see Appendix F).

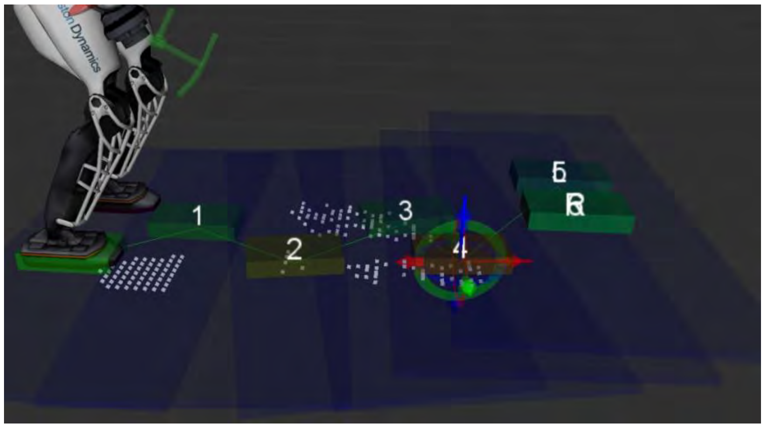
During the DRC Trials we have noticed the inability to refine generated footstep plans as a shortcoming. Although, the planner is able to generate feasible plans, there always remains a possibility that the resulting plan contains undesirable steps due to noisy sensor data. In this case, the operator previously had to request a new step plan in the hope to get a better result which may end in an infinite loop without mission progress. For this reason, the footstep planning system was extended to provide multiple services to manage footstep plans. These services can be used by user interface to enable interactive footstep planning allowing full human in the loop planning. This mode allows for plan stitching, plan revalidation and editing single steps with assistance of the footstep planner (more details see Appendix F). The operator is able to quickly adjust single steps while the planner will automatically update the 3D position of the new foot pose if enabled and provides immediate feedback if the modified step sequence is still feasible for the walking controller. This new interactive planning mode significantly improves mission performance during locomotion tasks, which is exemplarily demonstrated in Figure 28.



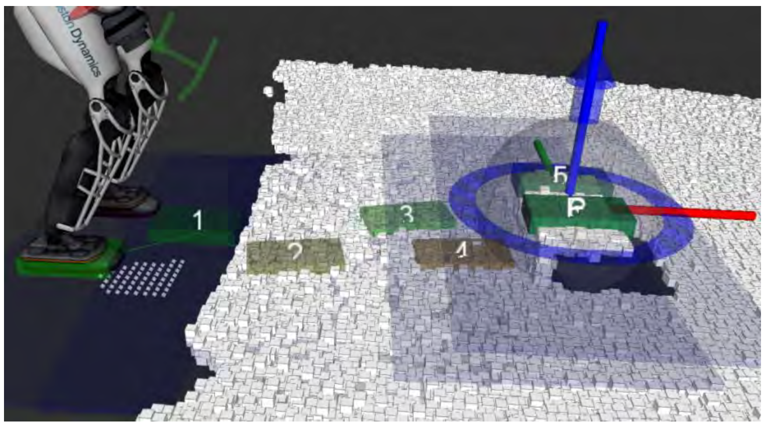
Operator has received a step plan for getting on top of a cinder block. In this case the operator is not satisfied with placement of step 4 as it is too close in front of the cinder block.



Operator selects step 4 for editing. Terrain model has been hidden for a better visibility of interactive marker.



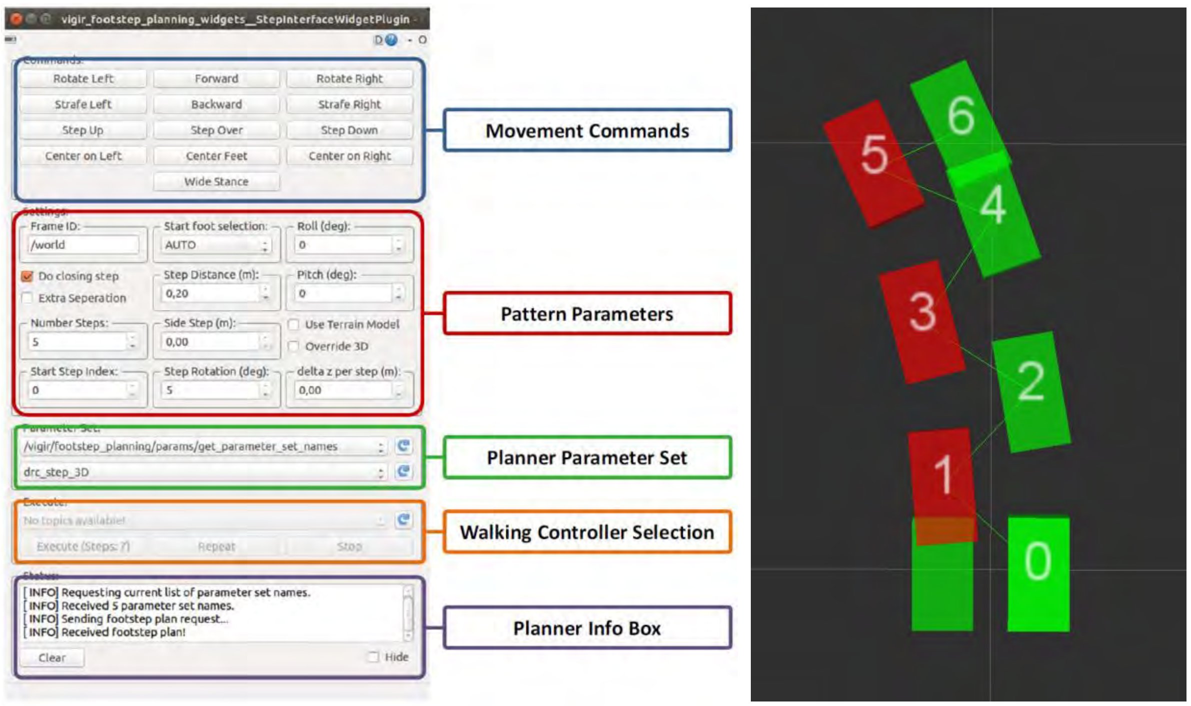
Step 4 has been moved slightly away from the cinder block by the operator.



Final result of modified footstep plan which is ready for execution.

Figure 28. Example how the operator is able to modify a generated footstep plan.

As the performance of the planning system highly depends on the quality of the world model, situations may occur where the planner gets stuck and does not deliver any feasible results. For this special case a pattern based mode was introduced which allows the operator to command simple movements. A special user interface was implemented which allows to define the pattern to be generated (see Figure 29).



Frame ID: /world
Start foot selection: AUTO
Roll (deg): 0
[✓] Do closing step
[ ] Extra Seperation
Step Distance (m): 0,20
Pitch (deg): 0
Number Steps: 5
Side Step (m): 0,00
[ ] Use Terrain Model
[ ] Override 3D
Start Step Index: 0
Step Rotation (deg): 5
delta z per step (m): 0,00

/vigir/footstep_planning/params/get_parameter_set_names
drc_step_3D

No topics available!

```
[INFO] Requesting current list of parameter set names.
[INFO] Received 5 parameter set names.
[INFO] Sending footstep plan request...
[INFO] Received footstep plan!
```

Figure 29. Step pattern widget (left) and resulting step plan (right)

3.2.6. High-level Behavior Control

Team ViGIR's based our approach to high-level behavior control on modeling robot behaviors as hierarchical state machines, which allows for modular composition and intuitive specification in different levels of abstraction. In addition to the logic of execution, behaviors also encode the data flowing through the behavior. Detailed monitoring of the state of execution and any errors that occur assists the operator when giving commands. The developed framework is able to cope with severe restrictions on the communication channel to the robot and is robust regarding runtime failure. In addition, verification of specified behaviors greatly reduces the risk of failure at runtime. This section presents the onboard Flexible Behavior Engine (FlexBE); Section 3.1.3.4 previously introduced the operator-side graphical user interface (FlexBE GUI). Appendix G provides an extensive treatment of the entire FlexBE system.

The concept of level of autonomy allows the system to use the individual capabilities of both robot and operator in a cooperative manner. Each behavior transition defines a level of autonomy that is required to execute the respective transition. There are four different autonomy levels: Off, Low, High, Full. The autonomy level mechanism allows the operator to reduce the autonomy of the onboard software and thus prevents the robot from making decisions on its own. As a result, behaviors are able to deal with changing uncertainty in scenarios while using the same state machine for implementation of the actions to be taken.

Figure 30 depicts a task-level behavior, "Open Door" in the FlexBE framework. A behavior consists of states (yellow), state machines (gray), and other, embedded behaviors (pink). The transitions (arrows) define the logic of the execution. Their color indicates the required autonomy level, which are illustrated in Figure 31.

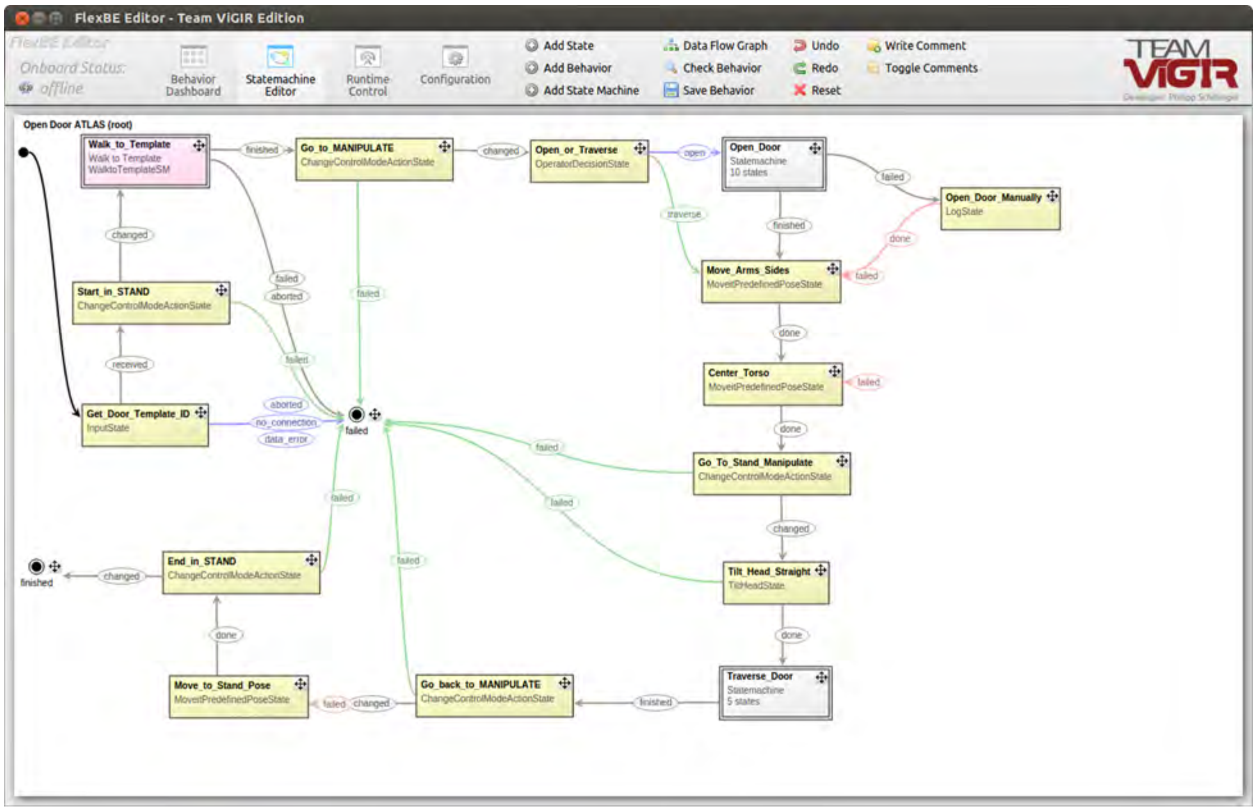


Figure 30. Task level "Open Door" behavior in the FlexBE framework

FlexBE monitors the state status, and if a transition is otherwise enabled, FlexBE will prevent the transition from occurring if the operator has reduced the autonomy level below that specified for the state transition. This allows the operator to adjust the permissions given to the robot on the fly based on changing conditions in the field. The FlexBE UI indicates this blocking by recoloring the transitions as shown in Figure 32.

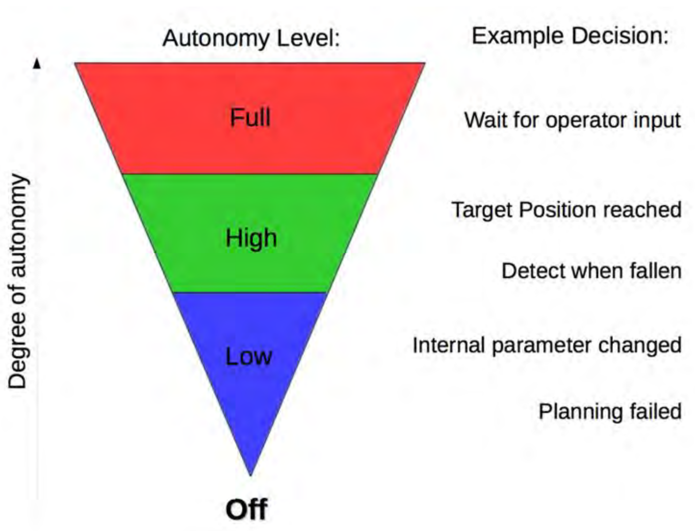
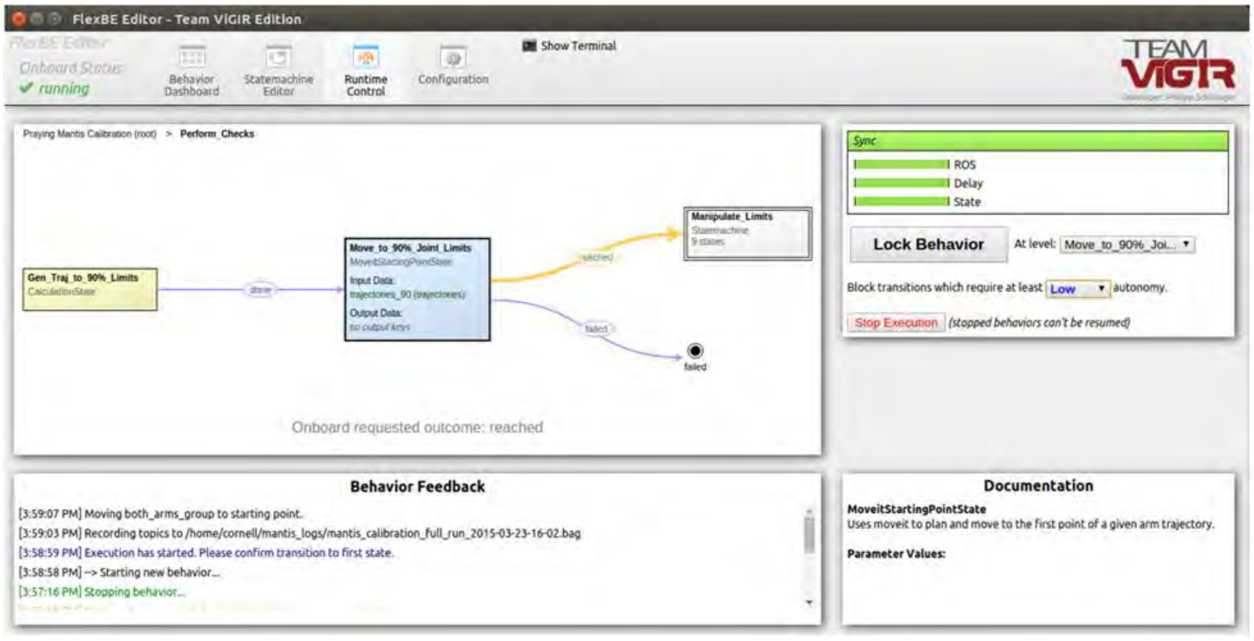


Figure 31. Example decisions for different Autonomy Level



```
[3:59:07 PM] Moving both_arms_group to starting point.
[3:59:03 PM] Recording topics to /home/cornell/mantis_logs/mantis_calibration_full_run_2015-03-23-16-02.bag
[3:58:59 PM] Execution has started. Please confirm transition to first state.
[3:58:58 PM] --> Starting new behavior...
[3:57:16 PM] Stopping behavior...
```

MoveitStartingPointState
Uses moveit to plan and move to the first point of a given arm trajectory.

Parameter Values:

Figure 32. Supervising a behavior during its execution (FlexBE runtime control view).
The state "Move_to_90%_Joint_Limits" returned the outcome "reached", but the behavior is not authorized to transition to the next state because the required autonomy level of that transition ("High", green) is higher than the current autonomy level set by the operator ("Low", blue).

In addition to the logical flow of the process, the behavior also encodes the flow of data through the states as shown in Figure 33.

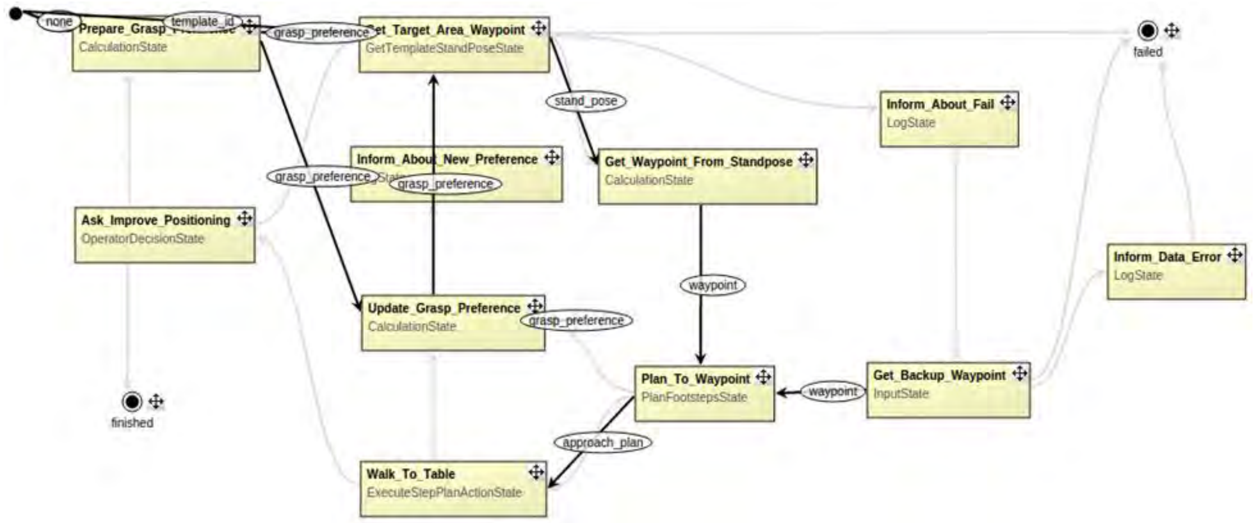


Figure 33. A behavior also encodes the flow of data (black arrows; transitions are grayed out).

The ability to perform runtime modifications is the most complex command available in FlexBE. It enables the operator to make arbitrary changes to the structure of a behavior without the need for stopping, compiling and re-starting it. Although this capability is very helpful regarding adaptability to unexpected situations, it also introduces some challenges. FlexBE takes steps to avoid failures related to runtime modifications and defines constraints to preserve consistency across versions of a behavior. Figure 34 illustrates an active, but locked, behavior.

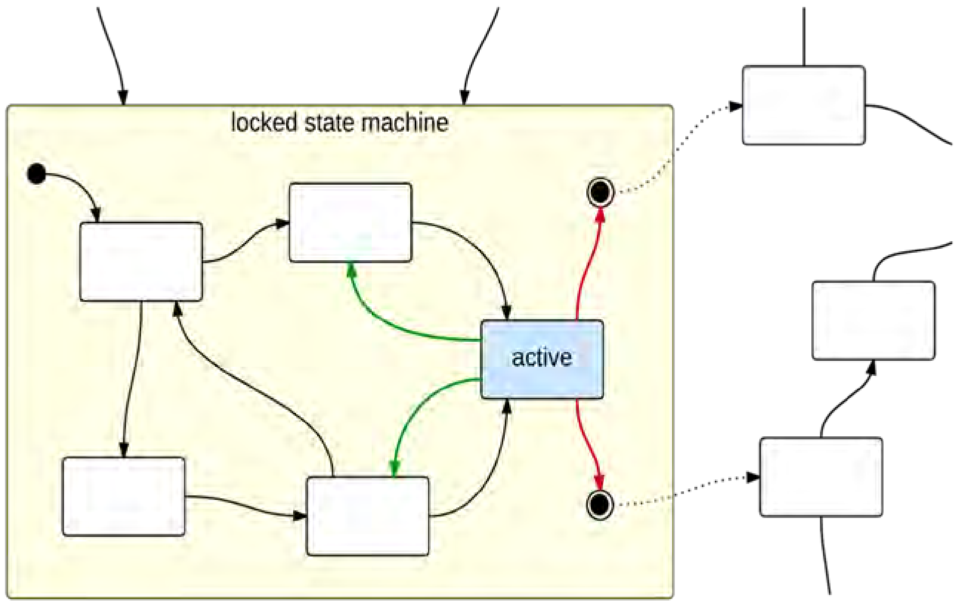


Figure 34. Behavior is running, but currently locked in one of its sub-statemachines. Blocked and allowed transitions are colored red and green, respectively.

When a behavior is locked in one of its states or sub-statemachines, these components are still executed, but the behavior cannot proceed. As depicted in Figure 34, internal sub-statemachine transitions are allowed, while outcomes causing a transition to the next state at the level of the locked container would be blocked. This mechanism ensures consistency across changes, without requiring the robot to pause and wait for the operator to make changes.

FlexBE is built on top of the SMACH[40] high-level executive Python framework. Although SMACH offers a solid basis for defining hierarchical state machines, the provided features are not sufficient for realizing what is required for our behavior control approach. Therefore, to create a powerful behavior engine supporting a high level of abstraction, FlexBE extends the SMACH framework with some features inevitably required to realize the concepts of cooperation and communication between operator and robot. In brief, the extension is made by inheriting the SMACH classes StateMachine and State (see Appendix G for details).

Section 4.2.4 and Appendix H present the behaviors that were developed over the DRC. Those behaviors are based on the FlexBE behavior engine, were designed in FlexBE's Editor, and are executed via FlexBE's Runtime Control interface (both components of FlexBE's GUI). In addition to behaviors, Appendix H enumerates all states and presents extensive experimental demonstrations.

3.3. Communications Bridge

During the DRC competitions, the robot onboard/field computers were connected to the OCS computers via a 1 GB/s network connection that passed through a network traffic shaper; the traffic shaper introduced communication restrictions intended to mimic the effects of poor wireless communications and encourage robot autonomy. All operator interactions with the robot occurred through the OCS hardware, with commands sent to the onboard software via the traffic shaper connection

As stated above, our team chose to use ROS for our communications middleware. The ROS system uses a publisher/subscriber model with a centralized *roscore* to coordinate communications between ROS nodes. This is not suitable for use with the communication challenges defined for the DRC competitions, as the system cannot tolerate a loss of communications of any node to the centralized *roscore*. For this reason, the team chose to use two separate ROS networks for the onboard and OCS software and develop a custom communications bridge (CommsBridge) to handle data transfer between the ROS networks. As the same topic names are used on both sides, the setup allows seamless testing as a single ROS network. Section 2.4 in [1], which is included in Appendix A, describes the specific communication challenges used during the DARPA VRC and DRC Trials, and the design of our CommsBridge for those competition.

For the DRC Finals, DARPA implemented a new communications restriction plan to increase the need for autonomy. The plan featured two always on channels that permitted 9600 bits per second data between robot and OCS; a third channel provided periodic bursts of 300 Mbits/s of data from the robot to the OCS, followed by variable blackout periods.

In reviewing the prior CommsBridge design in light of the new restriction, there were several relevant features – templated topic handling, compression, and custom state handling – and a few that required changes.

[40] http://wiki.ros.org/smach (accessed July 30, 2015)

With periodic bursts of high-rate data, image compression and region of interest selection were deemed less relevant, and the ability to send image data via UDP over the high rate channel more relevant. TCP communications of compressed images was deemed problematic as the channel might open or close in the middle of an image transmission; the lost packets would render the entire image useless. Instead, Team ViGIR developed an approach to divide the image into tiles that could be individually compressed and transmitted in one single UDP packet. The image tiles were reassembled into a coherent image on the OCS side of the CommsBridge as shown in Figure 35. The previous image data was retained so that lost packets did not result in a completely corrupted image.

A few systems that required significant amounts of data transmission were split to have a mirrored approach between the OCS and onboard. An example was the footstep planner; when running the CommsBridge, a special OCS/Onboard Footstep manager handled coordination between the OCS controls and two OCS/onboard footstep planner instances. This reduced the required communications through the always on data channels.

Team ViGIR implemented and tested these changes during Q1 2015, and the approach seemed to be working well in our lab. At the initial testing at the South Carolina Test Bed in March 2015, we uncovered a major shortcoming of our approach relative to the particular implementation of the DARPA communications. While our average rate was well below the limitations, the burst rate was higher and the limited packet buffer design implemented by DARPA would overflow causing the system to drop numerous packets. Team ViGIR revisited the design, and implemented a per channel relay.

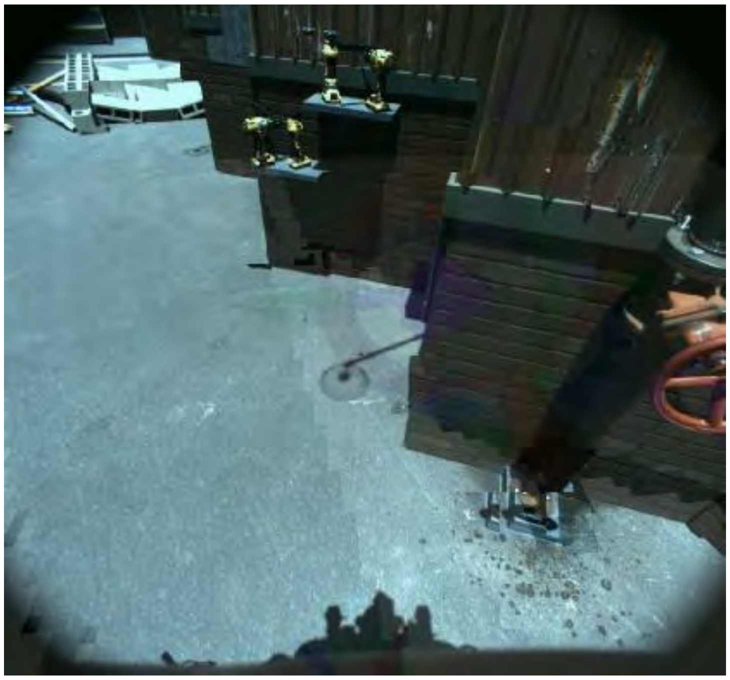


Figure 35 Video capture with artifacts

This software worked by connecting to a list of signals on either side and organizing each packet to send across based on a predetermined priority of the message. The bridge adhered to the bandwidth limit by calculating the wait it needed based on the bandwidth that particular bridge was configured for and would keep itself busy during that wait time by preparing the next packet. Multiple bridges were created to handle the fat pipe, each handling specific parts with the amount of bandwidth we wanted to allocate to each. This system worked well in testing and had one side of the bridge running on the field computer and the other side on a dedicated OCS machine.

At the heart of the Comms Bridge software we have several instances of the bridge node, each configured to send across a specific set of messages at the bitrates required to keep them within the bandwidth limitations. The nodes operated by tagging every message it received with a timestamp, priority, and a few flags based on how that individual signal was configured and storing them in a map where message priority and time stamp dictated its position. Then the next time the node had to busy wait for its next chance to send a packet, it would go through the map of messages it needed to send and started taking the messages off the top until it went through all of the messages. If a message was too big for the current packet it was skipped over but left in the map for the next packet. Then if the node had a big enough packet to send, it would check to see if it had waited long enough for it too not exceed the bandwidth restrictions by sending this next packet and do so if it could. To prevent holding onto stale data, the node would ignore the minimum packet size if it had been too long since the last time it sent a packet. The receiving side of the bridge was very simple where all it would do is extract the data from each packet and retransmit it on its side of the bridge for other software to use.

To ensure that we could send everything we wanted specific messages such as the robot state, images, and LIDAR data, were handled in a special manner as discussed in Section 3.2.2.2 to allow us to compress the data even further than we could with a generic message. To handle dropouts, a buffer of the last 30 seconds of compressed LIDAR data was sent over multiple times a second to make sure the latest point cloud data could be reconstructed on the OCS.

State data used a custom packing format. Joint positions were encoded as signed 2-byte numbers to represent $\pm\pi$ to 1/10,000 radian as opposed to a 8-byte double precision number. Likewise, pose information was defined using six 2-byte numbers to represent positions relative to a periodically updated reference position and the qx,qy,qz values of scaled normalized quaternion. The reference pose was updated every 16 seconds using a standard double precision pose. The remaining data signals were structured such that they compressed the data as much as they could on their own.

4. RESULTS AND DISCUSSION

Given the project overview and system background presented in Sections 2 and 3, this section discusses the particular challenges of the project and the technical results of our approach. Section 4.1 discusses the significant challenges faced by our team that affected our performance. Section 4.2 presents experimental results for the major sub-systems first introduced in Section 3.2; the results refer to the appendices for technical details.

4.1. Significant Challenges

This subsection discusses particular challenges, both programmatic and technical, that our team faced during the course of this project. Particular attention is paid to the issues that directly impacted our performances during the competition events.

4.1.1. Schedule

The primary challenge facing the team was schedule. The project entailed the most challenging robotics program to date that implemented on an extremely aggressive timeline. Team ViGIR faced an additional challenge of building our team and infrastructure from scratch. Where other groups had extensive histories with humanoid robotics, we assembled Team ViGIR for this particular project. Furthermore, the team lacked an existing automated unit and simulation-based testing framework; the effort to set such a system up required resources that we did not have available within the confines of this project.

As discussed in Section 2.3.1, we defined the basic structure of our software architecture during the VRC while both the simulation and robot hardware were being developed in parallel. The lack of specificity up front delayed implementation of some controllers, and required subsequent rework. Later differences between the simulation and robot API have required addition rework under the extremely compressed timeline between robot delivery and the DRC Trials.

The compressed schedule, limited developer resources, and hardware issues on site at the DRC Trials limited our ability to train operators for the DRC Trials. A few mistakes during the competition kept us from directly advancing to the DRC Finals, which ultimately cost us at least four months of development time, and six months until our robot was again ready for testing. This delay prevented us from bringing Cornell onboard early, and limited the autonomous behavior development we could do. This self-inflicted wound to our schedule prevented portions of our system from being ready for testing prior to the robot departure in November 2014.

The biggest challenge leading up to the DRC Finals, and cause of subsequent scheduling issues, was the Atlas Unplugged hardware issues as discussed in Sections 2.3.3.3 and 4.1.4. These hardware issues were mostly due to the compressed development schedule that BDI was working under.

4.1.2. Geographic Dispersion

A unique aspect to our team was the diversity in both nationality and geographic location. Darmstadt, Germany to Corvallis, Oregon spans a nine-hour time difference, which made communication and coordination a constant challenge. The team made extensive use of web-based project tools including a Redmine issue tracker and wiki for collaborative sharing of information, and Git-based shared code repository. Weekly teleconferences were held via Skype, but the lack of face-to-face time led to integration issues with some sub-systems.

Travel costs and extensive time away from family limited the amount of testing for colleagues in Germany. While our planned development and test sprints worked well in the fall, the constant hardware issues negatively affected test schedules in spring 2015 as travel plans needed to be changed. Some planned tests could not be run during time on-site due to recurrent hardware issues, and could not be adequately tested in simulation due to the simulator fidelity issues discussed above.

4.1.3. Simulation

The simulator fidelity was a significant disappointment; from our perspective, the issues were primarily due to the lack of coordination between OSRF and BDI. The simulation did not perform well after the VRC, as BDI required a proprietary library that they did not update. We did not have our own simulation environment (c.f. IHMC), and our geographically dispersed team required the simulation for system checkout.

Several significant issues made it especially difficult for our team. The updated system models could not walk in simulation until spring 2015; this required use to maintain different setups to test basic step controllers and manipulation. The system swayed in MANIPULATE mode to the point that we could not test grasping and manipulation without "pinning the hip." These issues prevented testing of integrated behaviors such as "walk to the table and pick up cutting tool" during crucial phases of the project. The inconsistencies between the robot and simulation API's (e.g. number of joints, naming conventions) likewise caused difficulties and required developer resources.

4.1.4. Hardware

Compared to the relatively reliable hardware used in the DRC Trials, the Atlas Unplugged version had numerous hardware issues during 2015 as discussed in Section 2.3.3.3. The initial delivery was delayed by six weeks, and then had recurrent hardware issues as it was being beta tested in the field. While other teams had similar hardware problems, the delays significantly affected our team due to the geographic dispersion.

The final hardware issue occurred on Day 2 of the competition in what we surmise to be a failure initiated by a problem in the custom hand electronics and compounded by overheating due to the delay. As discussed in Section 2.3.3.6, the robot had an initial arm failure that delayed our start while the robot sat in the California sun. An unexplained communication issue caused issued during the driving task. After additional delays due to resets, the robot experienced an unexplained communications error that induced a pump shutdown. The robot interface continued to update prior to the shutdown, which indicates the software was operating; one possible explanation is an overheating issue. Other teams reported issues with their switch when overheated. Unfortunately, our onboard logging was not operational during this phase, and we cannot reconstruct a definitive cause.

4.1.5. Developer Resources

Team ViGIR was fortunate to have our core group of developers with us throughout the project; however, this small group required significant assistance from a larger group of student volunteer developers and some limited part time software developers. The complex system, both the actual robot software and the ROS catkin build system, had a steep learning curve and required very capable developers. Integration of new team members was made more difficult by evolving software and rules, and the struggle to maintain online reference documentation under the schedule pressures. In several cases, new developers were unable to grasp the system, and therefore consumed more resources than they contributed. Some developers made good progress on some novel aspects, but were unable to get their software integrated independently, and required too many resources from the core team. In other cases, the students made significant contributions, but were only with the project for a short time.

The allocation of scarce developer resources was made more difficult due to changes in the hardware or simulation system design and to changes in the rules. For example, the team invested in developing a compliant whole body planning and control framework based on the expectation that the robot would need to egress and get up from a fall without a reset. After investing resources to get these researchers up to speed and integrated with the team, and make software modifications to support their efforts, the delays to the robot hardware delivery and limitations of the hardware performance prevented the development of the compliant controller in time for the competition. Furthermore, changes to the rules rendered this effort unnecessary. Thus, while the controller team made good progress as detailed in Sections 3.2.1.3, 4.2.1, and Appendix D, the investment did not pay off at the competition because of external issues.

4.1.6. Build and Test Infrastructure

Team ViGIR lacked a dedicated developer to handle infrastructure and testing. This led to shared responsibility across the core developer team. Early on, Team ViGIR recognized the need for an automated build and test environment, but lacked the in-house expertise in both the testing tool chain and ROS build system. The team attempted setting up such a system twice. The first automated build system was based on the existing infrastructure at TU Darmstadt, but did not include automated testing and was only accessible to certain people on the team. The team abandoned the second effort to set up a common build and test infrastructure due to personnel changes and resource restrictions in the lead up to the DRC Finals.

Lacking such a system, it was up to individual developers to test their changes prior to merging into the main code branch; unfortunately, changes that worked in one part of the system, could negatively affect another sub-system. Lacking a robust high-fidelity simulation as discussed above, the team did not have an automated way of testing behaviors and integrated system capabilities. Without automatic simulation-based validation, these errors could go undetected outside the full system integration. Thus, the team faced a constant struggle to balance keeping an up to date integrated system for testing with the operators, with premature introduction of bugs into the system that would negatively impact other developers productivity. The geographic dispersion of our team magnified this issue.

The large integrated build environment could take a significant amount of compile time for relatively minor changes to base messages or headers. Thus, a simple change to one package might result in a significant delay for the developer of an unrelated package just due to build time. There are tools to manage this complexity within the ROS catkin ecosystem, but lacking a developer dedicated to infrastructure, the team was unaware of some of these, and did not get them integrated into our system prior to the competition.

4.1.7. Communications

After working well during the DARPA VRC and DRC Trials, the CommsBridge development represented a significant challenge during spring 2015. As discussed in Section 3.3, issues discovered at the DRC Test Bed in South Carolina necessitated a change in our design relatively late in the development cycle. This, combined with delays in system development caused the hardware delays and changes in developer availability, led to delays in getting a fully functional CommsBridge until the team was on site in Pomona, CA. Beyond taxing the developers, this affected the full system testing the team was able to do during network checkout in the lead up to the Finals. In spite of these issues, the system worked well during the dress rehearsal on June 4, 2015.

During the competition, the team experienced unexpected communications issues between the field computer and the onboard computers. Team ViGIR had arranged its behaviors software running on the field computer with the communications bridge software; this decision was a legacy of using the behaviors to do automatic logging on the field computer for certain tests. Under this arrangement, our normal bandwidth across the network between onboard and field was well below the 300 Mb/s rate, and appeared to give ample headroom for wireless packet loss. At the competition, as the robot approached the grand stands we began to experience a communication backlog that prevented our autonomous behaviors some working reliably. While we were not monitoring the network bandwidth directly, we heard from the WPI/CMU team that they saw their monitored bandwidth drop to less than 50 Mb/s, which was above our average through put, and likely contributed to a network backlog. In spite of this loss of autonomous behaviors, our operators were able to adapt and score three points and nearly scored a fourth point.

In the evening after the Day 1 competition, Team ViGIR worked to rearrange their software to reduce the expected communications across the wireless channel. During testing that night, and in checkout prior to our Day 2 run, the changes appeared to be working well. Unfortunately, the aforementioned hardware problems impacted our run on Day 2.

While the autonomy worked as expected during our run on Day 2, we did have another delay evident from our video of the operators console during our driving task. At one point the operator can be seen giving commands, but the vehicle does not immediately respond. The vehicle then begins to respond to the commands, but does not stop when commanded and contacts a barrier. As our logs were not enabled during this run, we are unsure if this was caused by our CommsBridge or the wireless communications.

Overall, the communications with the robot cause significantly more unexpected issues at the DRC Finals than in the earlier stages. In the future, we will work to improve our CommsBridge and incorporate monitoring of the bandwidth across all channels, along with automatic logging that does not require the operator to start the logging process.

4.2. Experimental Results

4.2.1. Robot Modeling and Control

Model based compensations like dynamics and gravitation compensation need exact knowledge of the model parameters. Experiments with the joint impedance controller using the given CAD based parameters provided by BDI showed that further identification was necessary to execute trajectories without jerky motions and to achieve gravity compensation where the arms are backdriveable with moderate force and hold in position without interaction.

For the identification, a base parameter regressor formulation of the robot arm dynamics is needed, which cannot be provided by a numerical library such as the RBDL, which was used for the trials [1]. All kinematic and dynamic equations had to be computed analytically using computer algebra systems, and parameter regrouping algorithms had to be applied. Appendix D explains the explicit algorithm based on IRT expertise and design tools.

We iteratively ran dynamic trajectories optimized for parameter excitation and identified the dynamic parameters. By using the latest identified parameters in the model, we could execute the trajectories smoother and faster in order to iteratively improve the next identification results.

Appendix D presents our experimental results that show a better velocity and similar position tracking performance for arbitrary trajectories than with the existing PD position controller. The especially good velocity tracking leads to smoother movement compared to the sometimes shaky movements with our current PD gainset. See Appendix D for figures and characteristic values used for the controller comparison.

Another advantage of the model based control approach is the ability to observe disturbance forces. We implemented a joint torque disturbance observer, which is able to detect collisions only from regarding the measured joint torques without the need of the force-torque sensors, which suffered drift and calibration issues. In our experiments shown in Appendix D, we demonstrate the ability to switch to a safe gravity compensation-only mode after a collision with an obstacle. See Appendix D for the implementation of the disturbance observer and explicit results.

4.2.2. Manipulation

To evaluate the Object Template manipulation approach we present both, the results obtained during the manipulation tasks in the DRC and also individual laboratory experiments. Detailed results of the DRC Trials can be found in [3] included in Appendix E. These experiments show how a human operator using OT can interact with the remote robot in a high-level task command manner. Appendix H shows experiments of how Team ViGIR used the OT in a higher level autonomy.

During the DRC Trials, the hose task was the most challenging task for manipulation. It required picking up the fire-hose, align it and attach it to a wye turning the nozzle which have 1cm^2 knobs around it. Even though there was no Atlas team that successfully attached the fire-hose to the wye, the time analysis presented in [3] shows that using the Object Template approach Team ViGIR was the fastest team to pick up the hose and bring it in a position near the wye. Team ViGIR ran out of time just shy of attaching the fire-hose, having the nozzle turned but no threads engaged (see Figure 36 and video[41]).

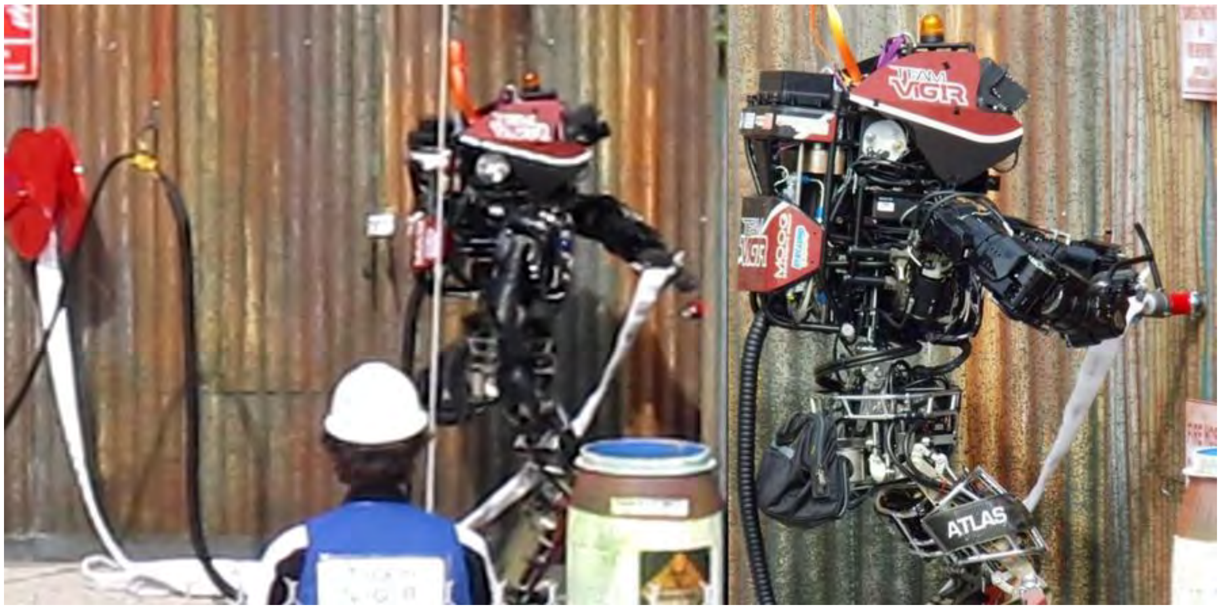


Figure 36. Team ViGIR during the Hose Task in the DRC Trials.

Another task in the DRC that required constrained paths for manipulation was the Valve task. Because of simplicity, the lever valve was turned using Cartesian teleoperation. The other two circular valves were turned using the Circular Markers developed for the Trials. While the main operator was in charge of placing the end effector inside the valve, the auxiliary operator placed the axis of rotation of the Circular Marker matching the axis of rotation of the valve. After the alignment was complete, the robot was commanded to perform the circular motions required to turn the valve (see Figure 37).

For the DRC Finals, we improved our approach as described in Section 3.2.4 and we were prepared to perform all manipulation tasks using affordance based manipulation (see Figure 38). Object Templates were created for the door, the valve and the drill describing the required motions that the robot needs to perform to achieve the manipulation task. We tested manipulation of these objects using the approach and preliminary results can be seen in Appendix E.

Unfortunately, due to communication issues during the first day of the Finals and hardware issues during the second day, we were only able to show our approach applied to the door and valve tasks. Nonetheless, after the DRC Finals, Team ViGIR continued performing experimental evaluation of the approach.

[41] https://www.youtube.com/watch?v=qHYGPMgysXI

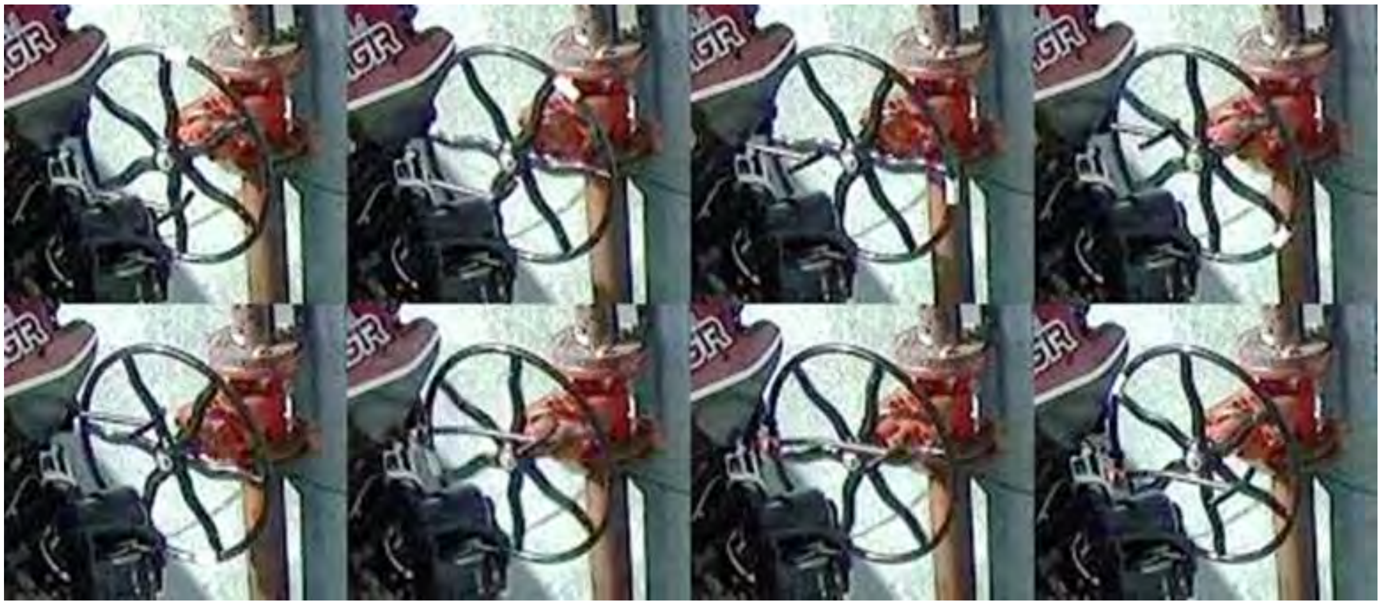


Figure 37. Team ViGIR during the Valve Task in the DRC Trials.

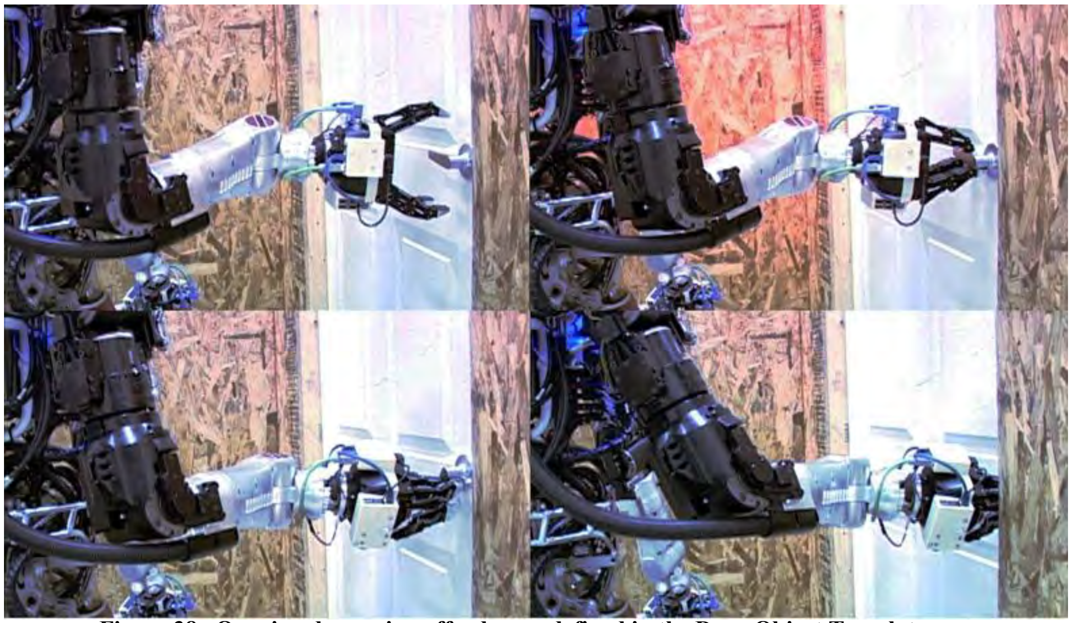


Figure 38. Opening door using affordances defined in the Door Object Template.

Upper Left: Final grasp pose. Upper right: Final grasp posture. Lower left: Using counterclockwise turn affordance with 60 degree. Lower right: Using push affordance with 0.05m.

During our Post-DRC experiment season, we tested the Object Template approach in manipulation tasks such as opening the door, turning the valve, and the surprise task of the cord plug. We performed these tests in two different ways: an operator commanded all the actions of the robot (pre-grasp, grasp, and affordance execution) as shown in Appendix E, and letting a behavior control all the actions of the robot (with the exception of object recognition and Object Template alignment) as shown in Appendix H.

An additional advantage of the Object Template approach presented here is that the operator has the ability to use objects in a different way than how they were designed. As described in [5] included in Appendix E, improvisation is an ability that can increase robustness while attempting manipulation tasks in post-disaster environments.

For more information, see Appendix E and our video playlist[42] that includes all manipulation experiments:

4.2.3. Footstep Planning

This section provides a brief overview of experiments using our footstep planning framework during DCR Trials and Finals. Detailed results of the DRC Trials can be found in [1] and [4] included in Appendices A and F.

Section 3.2.5 presented an integrated footstep planner which has been evaluated successfully during the DRC Trials. The only falls were due to operator error or hardware issues; but the footstep planner performed as expected. The novel ground contact estimation allows overhanging steps which significantly improves planning performance for the terrain task; therefore, it took only a few minutes and very few interaction steps by the operator to cross the pitch ramp[43] and the chevron hurdle[44] during our terrain task run at the DRC Trials.

Although the planner has worked very well for us, it took a lot of time to tune all parameters for a good performance. Many experiments were required to determine the limits of the walking controller and even more experiments to discover all special cases. This motivates further investigation how to simplify this process.

As discussed in Section 3.2.5 the footstep planner is also required to solve navigation problems like walking through narrow doorways. Unfortunately, operator error caused a fall during this task at the DRC Trials, but a video[45] of the robot walking autonomously through a very narrow doorway without any collisions using our footstep planner is available.

These examples show that planner is capable of solving navigation problems as well as generating feasible plans within seconds. Unfortunately, it is still too slow for online replanning when the robot is already walking; here, we need a result in less than a second to be able to inform the walking controller about the new step sequence in time. For this reason a walking monitor was implemented which can trigger a soft stop if it detects any issues during step plan execution. The problem of replanning efficiency will be a topic for future work.

[42] https://www.youtube.com/playlist?list=PL7v9EfkjLswJQn2yZE3qv5sECx-qZbeXO
[43] https://www.youtube.com/watch?v=7Qv__bLa3j4
[44] https://www.youtube.com/watch?v=vAtqVKGWvFM
[45] https://www.youtube.com/watch?v=BlUfl5iSAkU

Summarizing, the planner is capable of utilizing existing black box walking controllers and generating feasible step plans in rough terrain scenarios in short time. But it is still not working flawlessly. Especially in rough terrain scenarios the quality of the generated plan highly depends on the quality of the perceived environment. If the perceived data is too noisy or even incomplete due to obstruction, information needed by the planner is too inaccurate. In case of noisy data, foot placement cannot be determined correctly; if the world model is incomplete, the planner cannot take into account unseen obstacles that may lead to colliding foot placements. As it cannot be guaranteed that the world model is correct and complete, we have never used the planner in a fully autonomous manner even though this would be possible through behaviors. Therefore, the operator is responsible for validating the footstep plan (e.g. through camera images) before permitting execution.

For the convenience of the operator the footstep planning system has been integrated into the OCS with different layers of abstraction. At the highest level of abstraction, the operator is supposed to trigger planning using a template or dragging a goal pose using an interactive marker (see Figure 39). The only needed interaction with the planning system consist of a dropdown selection box where the operator can switch between different planner parameter sets e.g. 2D vs. 3D planning (see Figure 40). Advanced features are hidden in the settings menu where you can change basic footstep planner parameters e.g. time budget and the behavior of footstep editing mode (see Figure 41). If the operator decides to manually adjust step placement, he can simply activate the edit mode by double clicking on the desired step. Afterwards an interactive marker appears which the operator can use to move and freely change the step placement (see Figure 28 in Section 3.2.5). Depending on the selected *edit step mode* in the settings menu, the planner will automatically adjust the moved step according to the underlying terrain. In any mode the planner indicates with a colormap from green to red how feasible the new step placement is for the walking controller, where red warns about violated constraints. If the entire planning system is failing for some reason, the operator has access to all advanced footstep planning features as well as detailed parameters (see Appendix F) through special widgets. In such worse case scenarios, the operator is even able to generate manually patterns of foot placements using the pattern based generation mode (see Figure 29 in Section 3.2.5).

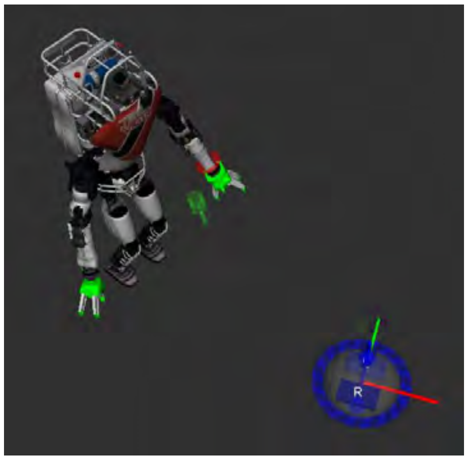


Figure 39. Interactive marker to define goal of the step plan request.

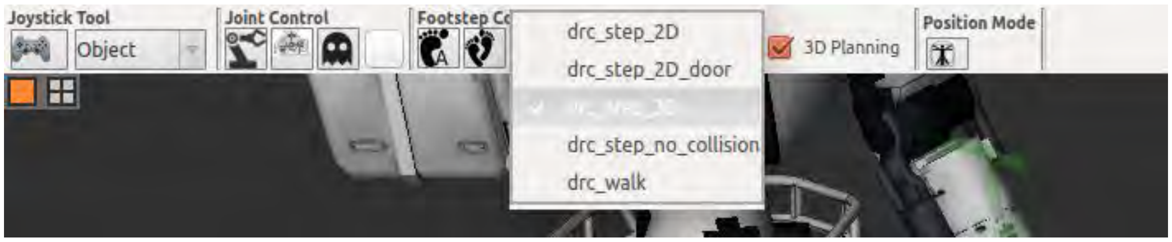
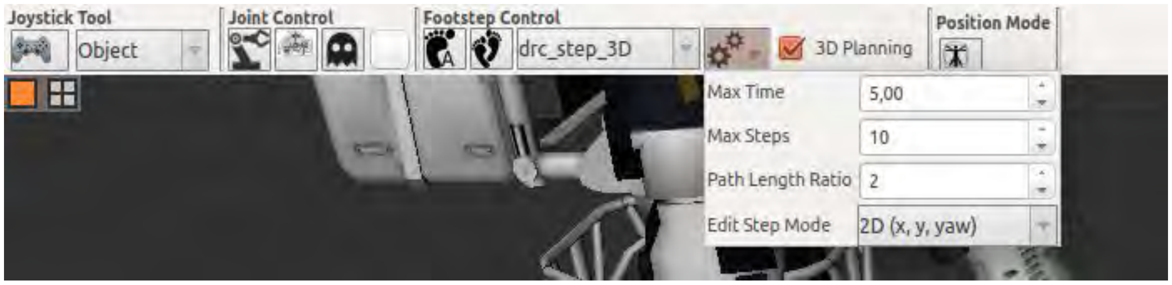


Figure 40. Drop down box to select a predefined parameter set.



Max Time 5,00
Max Steps 10
Path Length Ratio 2
Edit Step Mode 2D (x, y, yaw)

Figure 41. Menu granting access to the most important planner parameters.

At the DRC Trials the operator had to request and refresh manually the terrain model when the robot has to travel across rough terrain. A goal for the finals was to disburden the operator from all low-level tasks like this one. For this reason we have enhanced the terrain generator by the capability to create and update automatically the terrain model on-line which is demonstrated in the Appendix F.

Our efforts of refactoring the footstep planner to a footstep planning framework has already showed results, but it is still an ongoing work. We have been able to provide the footstep planning framework to Team Hector and Team VALOR. After implementing the mandatory hardware interface and defining the correct parameters, the entire footstep planning framework presented in Section 3.2.5 became available for them. Therefore, the robots ESCHER and THOR-Mang used our footstep planning approach and the OCS with their own walking controllers. Unfortunately, hardware issues at the DRC Finals kept them from showing their full locomotion planning potential during DRC Finals.

The entire footstep planning framework is already available as open-source code under GitHub:

- https://github.com/team-vigir/vigir_footstep_planning_msgs
- https://github.com/team-vigir/vigir_footstep_planning_basics
- https://github.com/team-vigir/vigir_footstep_planning_core
- https://github.com/team-vigir/vigir_terrain_classifier
- https://github.com/team-vigir/vigir_pluginlib
- https://github.com/team-vigir/vigir_generic_params

4.2.4. Behavior Control

Team ViGIR created behaviors for some of the tasks in the DRC Finals. Specifically, we had "Open Door," "Turn Valve," and "Cut hole in Wall" behaviors[46]. For the driving task, we had a behavior for positioning the robot for car entry and then for driving ("ATLAS Vehicle Checkout"). We did not attempt the vehicle egress task, therefore we did not create a behavior for it. Moreover, we did not create behaviors for the uneven terrain and stairs, since those tasks did not involve complex sequences of locomotion and object manipulation.

In addition to the task-specific behaviors, we had behaviors for performing the initial ATLAS checkout upon startup as well as for calibrating the hydraulic joint offsets. For example, the latter ("Praying Mantis Calibration") was employed when ATLAS was placed outside the door area (as part of the requested reset) after the driving task (see Figure 42). This behavior drives the hydraulic joints to their limits in order to measure the encoder offsets and properly calibrate those joints. Performing this calibration was crucial for accurate manipulation; using a pre-defined behavior speeded up the checkout, and reduced errors.

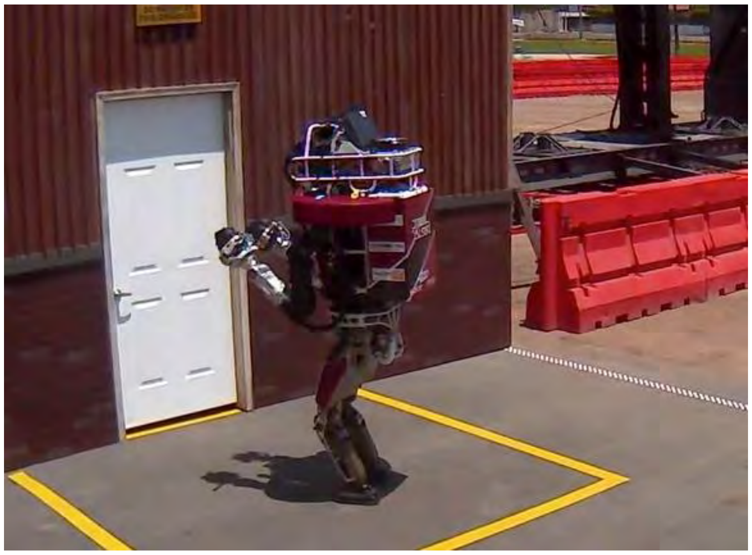
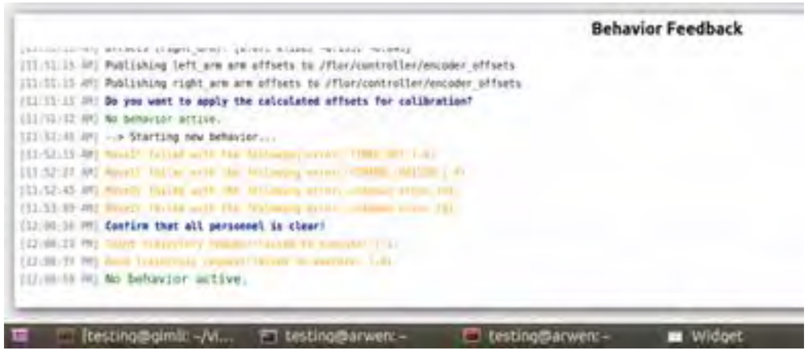


Figure 42. ATLAS executing the "Praying Mantis Calibration" behavior

DRC Finals

On Day 1 of the DRC Finals, due to the unexpected communication issues mentioned in Section 2.3.3.5, action requests originating from the Behavior Engine (deployed on the field computer) were not being serviced by the corresponding action servers (deployed on one of the onboard computers). Examples include footstep execution and motion planning for the arms (Figure 43). Even the "Praying Mantis Calibration" (Figure 42) did not work as expected and thus the hydraulic joints were not calibrated. To conclude our summary of Day 1, the contribution of behaviors to our performance was negligible.


Behavior Feedback

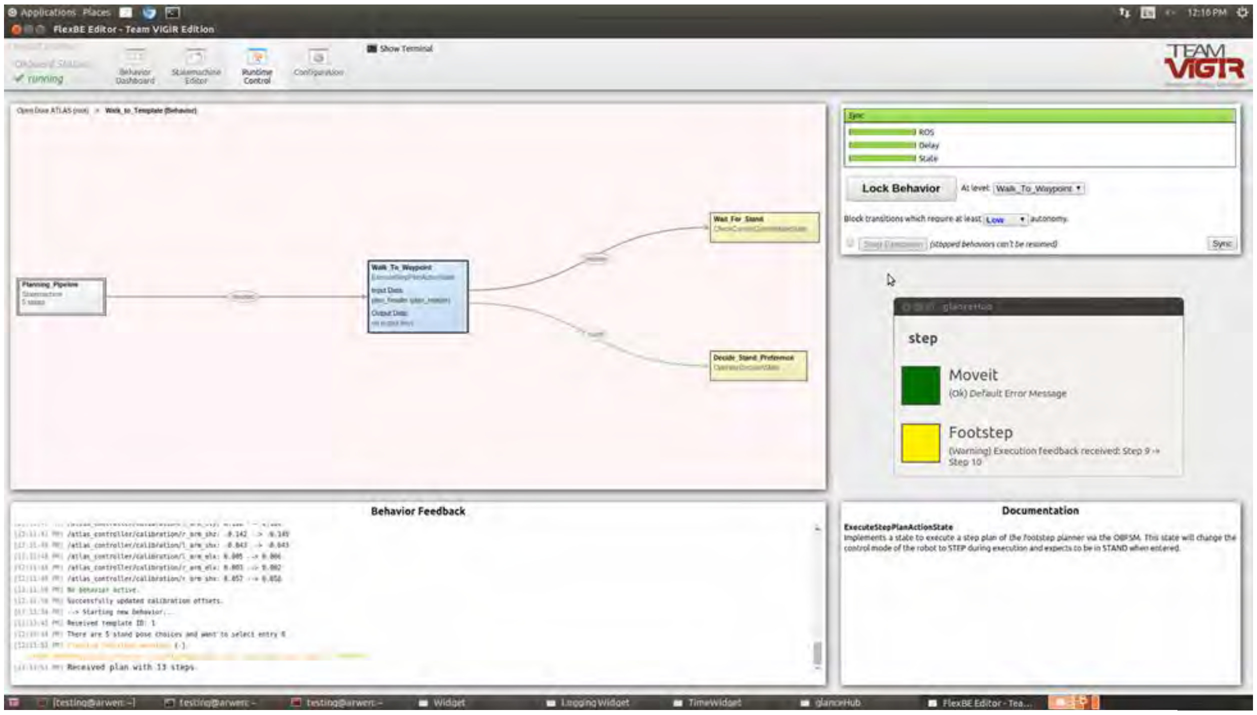
```
[illegible]
[11:51:15 AM] Publishing left_arm arm offsets to /flor/controller/encoder_offsets
[11:51:15 AM] Publishing right_arm arm offsets to /flor/controller/encoder_offsets
[11:51:15 AM] Do you want to apply the calculated offsets for calibration?
[11:51:32 AM] No behavior active.
[11:52:48 AM] --> Starting new behavior...
[11:52:15 AM] [illegible]
[11:52:27 AM] [illegible]
[11:52:45 AM] [illegible]
[11:53:09 AM] [illegible]
[12:00:30 PM] Confirm that all personel is clear!
[12:00:22 PM] [illegible]
[12:00:39 PM] [illegible]
[12:00:59 PM] No behavior active.
```

Figure 43. Behaviors errors on DRC Finals Day 1

Between our two runs, we moved the Behavior Engine deployment to an onboard computer, in an effort to circumvent the unexpected communication issues. Thus, on Day 2 of the DRC Finals, behavior execution was working as expected (Figure 44 and Figure 45). Based on our experience with opening the

[46] The state machines corresponding to behaviors mentioned in this section can be found in Appendix H.

door using the “Open Door” behavior, we hypothesize that the “Turn Valve” and “Cut Hole in Wall” behaviors would also have executed as expected.



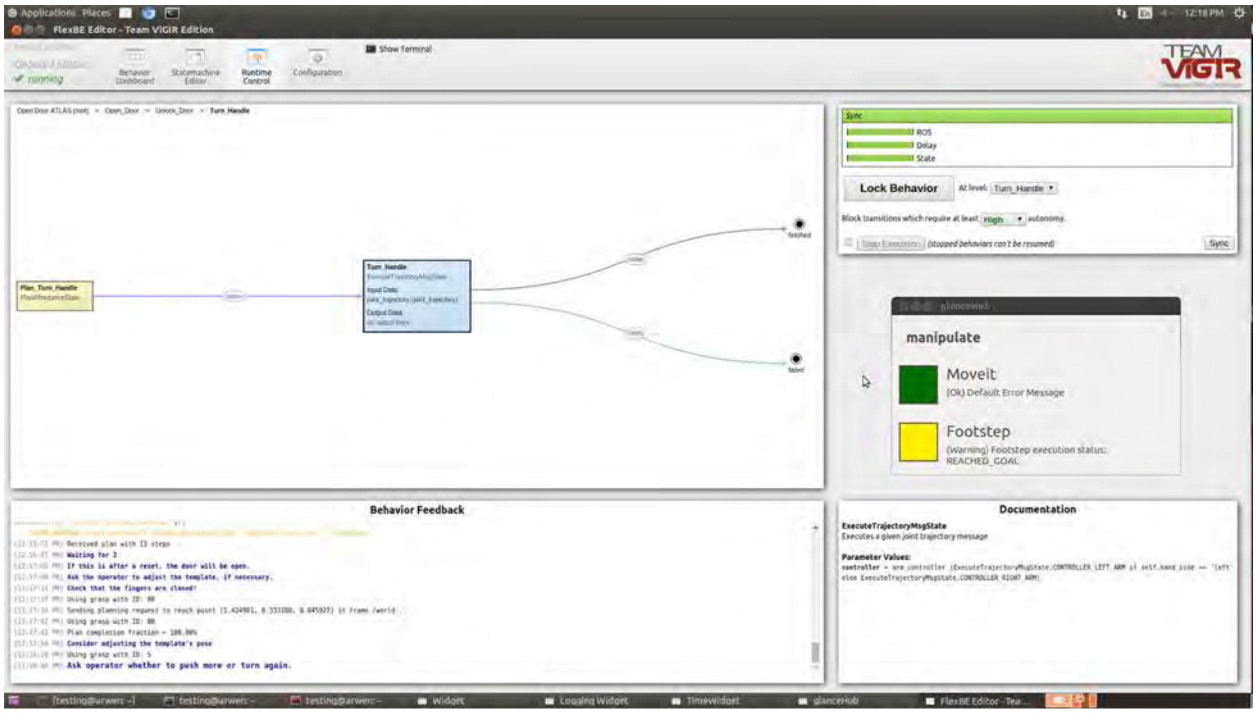
```
[illegible] /atlas_controller/calibration/r_arm_shz: -0.142 -> -0.145
[illegible] /atlas_controller/calibration/l_arm_shx: -0.043 -> -0.043
[illegible] /atlas_controller/calibration/l_arm_ely: 0.005 -> 0.006
[illegible] /atlas_controller/calibration/r_arm_ely: 0.003 -> 0.002
[illegible] /atlas_controller/calibration/r_arm_shx: 0.057 -> 0.056
[illegible] No behavior active.
[illegible] Successfully updated calibration offsets.
[illegible] --> Starting new behavior...
[illegible] Received template ID: 1
[illegible] There are 5 stand pose choices and want to select entry 0
[illegible] [illegible] (-).
[illegible] Received plan with 13 steps.
```

Moveit
(Ok) Default Error Message

Footstep
(Warning) Execution feedback received: Step 9 -> Step 10

ExecuteStepPlanActionState
Implements a state to execute a step plan of the footstep planner via the OBFSM. This state will change the control mode of the robot to STEP during execution and expects to be in STAND when entered.

Figure 44. The “Open Door” behavior successfully guiding ATLAS towards the closed door on Day 2


FlexBE Editor - Team ViGIR Edition
Behavior Dashboard
Statemachine Editor
Runtime Control
Configuration
Show Terminal
TEAM ViGIR
Open Door ATLAS (root) > Open_Door > Unlock_Door > Turn_Handle
Plan_Turn_Handle
Turn_Handle
finished
failed
Sync
ROS
Delay
State
Lock Behavior
At level: Turn_Handle
Block transitions which require at least High autonomy.
Stop Execution (stopped behaviors can't be resumed)
Sync
manipulate
MoveIt
(Ok) Default Error Message
Footstep
(Warning) Footstep execution status: REACHED_GOAL
Behavior Feedback
Documentation

```
[12:33:53 PM] Received plan with 13 steps
[12:36:47 PM] Waiting for 2
[12:37:05 PM] If this is after a reset, the door will be open.
[12:37:09 PM] Ask the operator to adjust the template, if necessary.
[12:37:11 PM] Check that the fingers are closed!
[12:37:17 PM] Using grasp with ID: 88
[12:37:38 PM] Sending planning request to reach point (1.424981, 0.333188, 0.045923) in frame /world
[12:37:42 PM] Using grasp with ID: 88
[12:37:42 PM] Plan completion fraction = 100.00%
[12:37:54 PM] Consider adjusting the template's pose
[12:38:28 PM] Using grasp with ID: 5
[12:39:40 PM] Ask operator whether to push more or turn again.
```

ExecuteTrajectoryMsgState
Executes a given joint trajectory message

Parameter Values:
```
controller = arm_controller (ExecuteTrajectoryMsgState.CONTROLLER_LEFT_ARM if self.hand_side == 'left'
else ExecuteTrajectoryMsgState.CONTROLLER_RIGHT_ARM)
```

Figure 45. The "Open Door" behavior in process of turning the door handle on Day 2

Post-Finals Lab Experiments

In order to validate the efficacy of the task-level behaviors, we carried out the three DRC tasks, door, valve, and wall cutting, in the lab. However, a hardware issue with our ATLAS' left hip prevented it from walking or stepping. Therefore, we skipped the locomotion part of those tasks. This was the only difference in terms of behavior design between the lab experiments and the DRC Finals. In addition, we created a variation of the "Open Door" behavior in order to compare two strategies for turning the handle; pushing it from below with the fingers in the "fist" configuration (i.e., completely closed) vs grasping and turning it in a more human-like manner.

From our lab experiments, we have included a total of four demos in this report; two for the "Open Door" behavior (one for each turning strategy), one for the "Turn Valve" behavior, and one for the "Cut Hole in Wall" behavior. These demos are presented in detail in Appendix H.

4.2.5. Behavior Synthesis

Team ViGIR concluded early on that the DRC Finals rules encouraged, if not mandated, increased robot autonomy as well as interaction with the robot at a higher level of abstraction compared to the previous phases of the competition. To this end, we developed FlexBE (Section 3.2.6), which extends the SMACH Executive framework. It also adds a graphical user interface (GUI) (Section 3.1.3.4) for facilitating the creation of behaviors, i.e., hierarchical state machines, for our Boston Dynamics ATLAS humanoid robot.

Use of FlexBE's graphical editor resulted in significant productivity boosts in terms of development time and also provided basic syntactic verification capabilities. However, the development process was still manual, relatively slow, required an expert user, and provided no guarantee that the resulting behavior satisfied the implicit user specification. This motivated the use of techniques from the nascent field of formal methods in robotics. Specifically, we set out to automatically generate (synthesize) correct-by-construction state machines from an explicit user specification.

First, we create a formal mission specification, expressed in Linear Temporal Logic (LTL), by augmenting the high-level specification provided by the user (e.g. the final objective) with robot and context specific constraints (e.g., action preconditions) as well as initial conditions. We then synthesize a provably correct automaton from the LTL formulas using a freely available, off-the-shelf synthesizer. Finally, from the synthesized automaton, we generate instructions that FlexBE uses to instantiate the state machine, i.e., generate Python code. Figure 46 depicts the corresponding ROS packages and the nominal workflow.

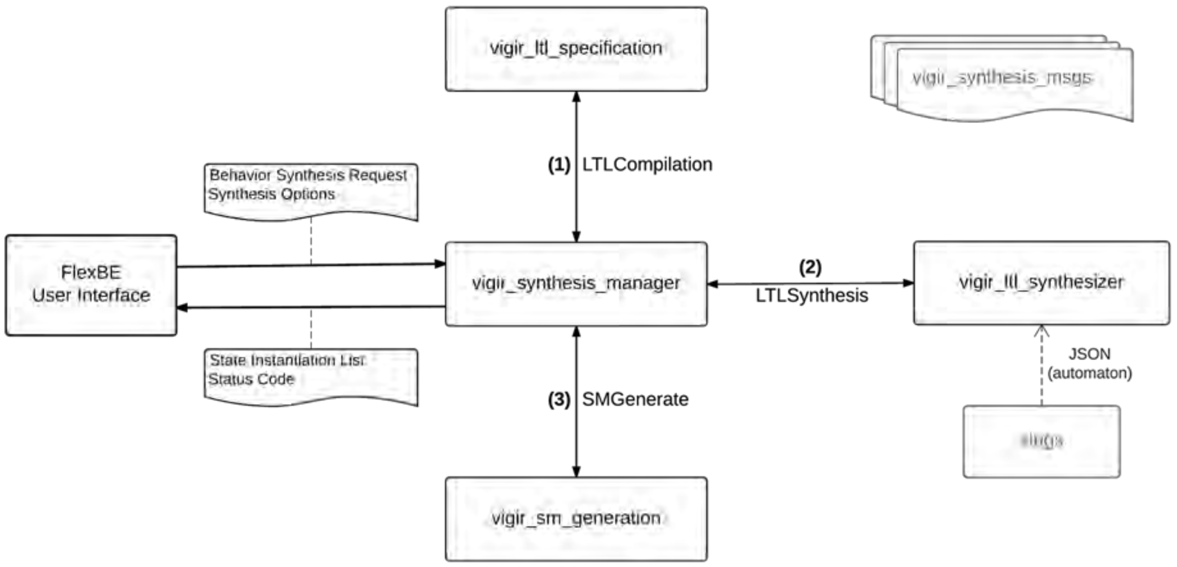


Figure 46. Behavior Synthesis ROS packages (vigir_behavior_synthesis) and nominal workflow.

As shown in Figure 46, the synthesis action server (vigir_synthesis_manager) receives a synthesis request from the user via FlexBE's GUI. Given the user's high-level specification, the server first requests a full set of LTL formulas from the LTL Compilation service (vigir_ltl_specification). The LTL Synthesis service (vigir_ltl_synthesizer) acts as a wrapper for an external LTL synthesizer. Upon request, it returns an automaton that is guaranteed to satisfy the LTL specification, if one exists. Finally, the server requests a state instantiation message from the State Machine Generation service (vigir_sm_generation). The resulting message contains instructions that FlexBE can use to generate Python code: an executable state machine that instantiates the synthesized automaton. The corresponding action, services, and messages are defined in the vigir_synthesis_msgs package.

The main theoretical contribution behind the Behavior Synthesis functionality is the modeling of actions with multiple possible outcomes (e.g. "completed", "failed", "preempted", etc.) in Linear Temporal Logic. We dub this the "Activation-Outcomes" reactive LTL specification paradigm. Its software implementation is part of the vigir_ltl_specification ROS package (see Figure 46). The theory behind Behavior Synthesis is presented in detail in Appendix I for the case of our ATLAS humanoid robot.

Behavior Synthesis has been integrated with FlexBE, which serves as a front-end to synthesis manager action server (see Figure 46). Developers do not have to start with an empty state machine when starting to create a new behavior or new parts of an existing behavior. Instead, they can provide a set of initial conditions as well as high-level goals to be achieved by this part of the behavior. Behavior synthesis will then draft a state machine that achieves these goals in a correct-by-construction manner. Developers can then further extend or modify the synthesized state machine, if desired, and also connect it to other parts of the behavior.

Synthesis works seamlessly with the process of runtime modifications to behaviors, resulting in powerful synergy effects. For example, it makes it much easier and faster for users to specify runtime changes since they only have to give high-level commands to the synthesizer instead of completely modeling the changes themselves. In addition, it could enable incorporation of even more powerful autonomous adaptation. In scenarios where the environment can be much better perceived by the robot, and the consequences of failure are considerably low, using a combination of behavior synthesis and runtime modifications will allow the robot to change its own behavior during execution depending on how the world changes. It will also achieve that in a provably correct manner, thanks to the strong guarantees of synthesis. This is a topic of future work.

Behavior Synthesis was not used during the DRC Finals for a number of reasons. First of all, the main developer of this functionality was also involved with the (manual) development of behaviors and states, which was deemed to be of higher priority. In addition, it was decided that this individual would be one of the four robot operators during the Finals, which imposed additional constraints on development time. Finally, there was a major technical reason for not employing Behavior Synthesis; the severe restrictions on communications during the Finals, which became apparent during the testing in South Carolina. Specifically, synthesizing a behavior on the operator's side and sending it to the robot for execution would result in prohibitively large packet sizes, which would be completely rejected by the network. Performing synthesis onboard could have circumvented this, because only small messages encoding the high-level objectives would travel over the degraded network. However, this would have been a major paradigm shift in terms of software architecture, since the FlexBE Editor (GUI), which performs the final step of Python code generation, is designed to run on the operator's side. This is another topic of future work in terms of development.

However, after the DRC Finals, we completed development of the Behavior Synthesis packages and performed a series of experiments in the lab. Appendix I describes these experiments in detail; Figures Figure 47 through Figure 49 depict one of the experiments.

In Figure 47, the user is in the process of specifying the initial conditions (STAND_PREP control mode) and goals ("look down", "pickup object") of the state machine to be synthesized. Clicking on the "Synthesize" button sends the Behavior Synthesis request to the corresponding action server (see Figure 46). Figure 48 shows the synthesized state machine.

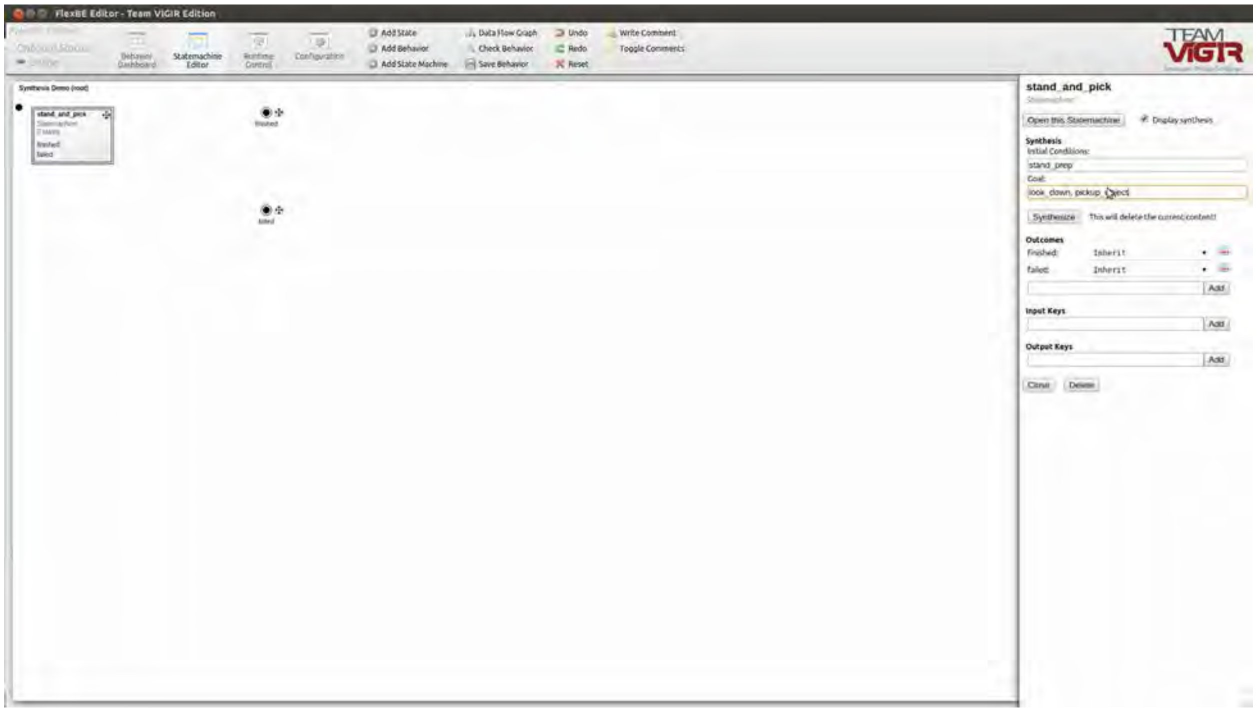


Figure 47. The FlexBE Editor's synthesis menu.

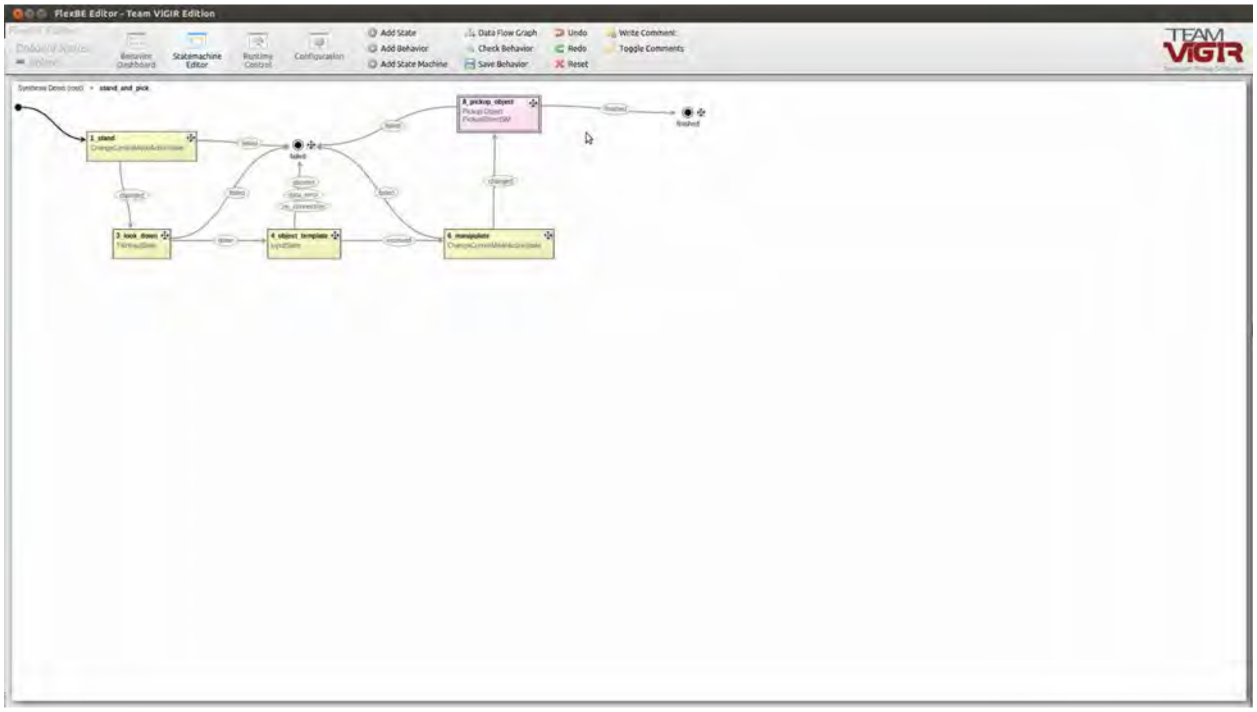


Figure 48. The synthesized state machine for pickup object.

In addition, the LTL Compilation process added additional constraints, such as the preconditions of executing the “pickup object” action: being in the MANIPULATE control mode and having an object template. In addition, since the initial condition was STAND_PREP and ATLAS needed to be in MANIPULATE, the synthesis process automatically added a state for transitioning from STAND_PREP to STAND in between as well.

Figure 49 shows the execution of the resulting state machine on the Atlas robot without modification. The user did have to manually choose which arm/hand side (left or right) Atlas should use to pick the object up. This is an artifact of the design of the state primitive (in this case, an embedded behavior), which could be changed to allow the user to set the arm/hand side as part of the specification (e.g. by inputting “pickup_object_right” in the “Goal” field; see Figure 48).

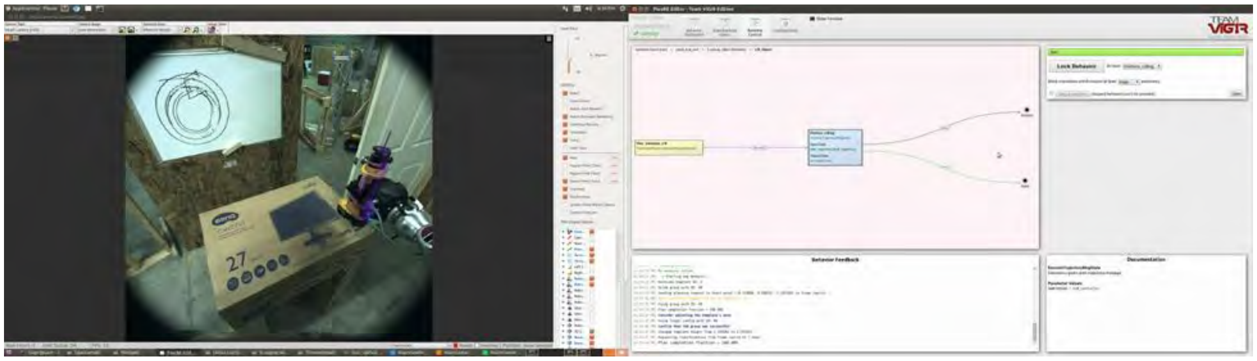


Figure 49. The synthesized state machine executed on Atlas.

This page intentionally blank.

5. CONCLUSIONS

This section discusses particular lessons learned, and presents our immediate plans for future research that builds upon the infrastructure that is now in place.

5.1. Lessons Learned

There are a number of lessons learned and improvements that can be made to individual components; these are left to the individual sections and appendices. In this section, we focus on team-level lessons learned that could have improved our performance, and on particular issues that we saw that DARPA may want to consider for future competitions.

5.1.1. Maintain Adaptability

With these types of competitions, especially ones under such tight schedules, the rules will change. Likewise, hardware delivery schedules will slip. It is important to plan for these changes, and to maintain adaptability in the system design. In spite of the challenges of functioning as a distributed team, this was a strength of Team ViGIR. While some resources were misspent in retrospect, overall the team defined a flexible architecture and adapted to changes in resources and schedule.

One lesson is the need to prune unproductive research branches quicker, and to avoid spending developer resources unnecessarily. This is complicated when operating with volunteer student resources, who have their own semester projects to complete.

5.1.2. Prioritize Infrastructure

Proper infrastructure is required.

The fidelity and completeness of the OSRF Gazebo-based drcsim was lacking, especially during Phase 2. This was driven both by development time pressures and a (perceived) lack of cooperation and transparency between BDI and OSRF. The issues, which were discussed in Section 4.1.3, severely impacted our team. While these issues were raised numerous times with both vendors and DARPA during Phase 1, we should have escalated them more; by Phase 2, the lack of progress became expected given the hardware development issues. In retrospect, we failed to escalate this issue sufficiently during the summer of 2014 when there was time to address the issue.

The second issue was with our internal software infrastructure for automated builds and testing. We used the Catkin build system from ROS, but lacked an integrated system for automated builds and simulation-based testing, which would have been helpful for ensuring overall software quality and ensuring a functional build for all parties. We tried a couple of times to set this up with part time student help, but this requires a significant level of expertise and focus to do correctly. Finding the right person for this job is critical, and something we failed to do with our resources.

Ideally, designing and developing this infrastructure should come before any development in a test-driven development framework. Adding such a system to a large complex system after the fact became a time consuming challenge, when developer time was at a premium. It is our position that having improved open source support for build and automated testing of these integrated systems would be greatly desired; this necessarily entails better simulation.

5.1.3. Separate Development and Testing

Our team struggled with having the same core group of developers working in design, software development, system testing, and operations. This resulted in overcommitted developers, and insufficient testing. Ideally, we would have had the designers testing the software that other people implement based on specifications; unfortunately, limited developer resources and the level of expertise required to develop the software prevented us from correcting this issue.

5.1.4. Force Early Integration

A continual challenge was the need to balance development and testing. This was made worse by the distributed nature of our team, and the split between OCS and onboard software development. In many cases the interfaces to the onboard software were evolving, which made integration with the OCS difficult; this caused developers to fall back into using simplified setups and engineering widgets to test their sub-system components. This led to stove-piping and last minute integration efforts after the interfaces were sufficiently mature.

For any given onboard module, the components needed to interface with our OCS and behaviors systems. Due to the distribution of expertise, we ended up with multiple streams of development that were coming together at the same time. Our intent was that module developers would be responsible for integration with behaviors; unfortunately, delays in development, delays in hardware availability for testing, and the distributed nature of our team conspired to push much of the integration onto our behaviors team. This led to rushed integration, duplication of effort, and insufficient testing of the integrated system.

The obvious answer is to maintain better accountability for deliverables, and strictly enforce test dates. This is challenging in any instances, and particularly so with a distributed team that depended on student developers using an imperfect simulation environment.

5.1.5. Require more openness from GFE Vendors

This is more of a DARPA program level lesson. As discussed above, the collaboration between BDI and OSRF was lacking, and insufficient resources were devoted to maintaining the simulation environment and releasing updates in a timely fashion. A more open and collaborative development arrangement was required.

5.1.6. Task difficulty

Overall, we felt the tasks were at an appropriate level of difficulty; however, in our opinion, the debris task missed the mark. The winning team and several other lightweight teams were able to push their way through the lightweight debris pile. As this was intended to be a manipulation challenge, it seems the task needed more interlocking parts to require manipulation and removal piece by piece.

5.2. Future Work

The work started under this DRC effort is continuing across our different sub-teams, both individually and in collaboration.

5.2.1. TU Darmstadt

Research in both humanoid and more conventional wheeled and tracked rescue robot systems will continue at TU Darmstadt. While teams at the DRC demonstrated impressive performance, there are significant research challenges that need to be solved before rescue robot systems are robust and mature enough to perform tasks of similar complexity to those in the DRC in a real disaster. The following research topics thus will be pursued:

- Perception and state estimation
  - Rich environment representations for supporting situational awareness/decision making of human operators facing previously unknown situations
  - Terrain classification (non-rigid, slippery terrain etc.)
  - Drift-free state estimation using internal and external sensing
- Human Robot Interaction
  - Tight integration between robot capabilities (planning), automated behavior synthesis and user interface tools for specifying tasks in complex and challenging environments
- Integration of heterogeneous robot platforms (such as bipeds, ground vehicles and/or UAVs) into a cooperating team
- Footstep Planning
  - Extend to adaptive level-of-detail planning to decrease planning time
  - Investigations in adaptive planner policies providing more safe plans and easier migration of new robots
  - Expand the footstep planning framework

5.2.2. Hanover

As long as the real robot platform Atlas is unavailable for us, we will use our existing control framework for a simulation based student lab, where students will understand the necessary steps of robot modeling and control design. We will extend our analytical robot model to complete upper body dynamics and finally full-body dynamics and try to implement full body (joint) impedance control and simple balancing control schemes. This will also be part of the student lab if it works in the gazebo simulation despite the aforementioned drawbacks.

If a humanoid robot platform would be available again, we will try to implement the control schemes mentioned above and would try to implement control for bimanual manipulation and cartesian impedance control.

5.2.3. Cornell University (Verifiable Robotics Research Group)

We want to improve our Linear Temporal Logic (LTL) -based Behavior Synthesis in a few ways. First, we want to allow the user to input richer high-level specifications in the behavior synthesis request; for example, to specify the robot's reaction to a dynamic, or even adversarial environment. This is already supported by the "back-end", i.e., the reactive LTL synthesis algorithm. It is a matter of facilitating the specification of such complex requirements by the user on a higher level, without having to write LTL formulas by hand. Furthermore, the more complex the specifications get, the more important it becomes to provide the user with feedback in cases of *unsynthesizable* specifications, ideally in natural or structured English. Our research group has already demonstrated this concept in different settings and we would like to apply such user-feedback techniques to behavior synthesis and integrate them tightly with the ROS-based behavior synthesis subsystem.

An aspect of Behavior Synthesis that we did not explore in depth in the context of the DRC is online synthesis and even re-synthesis on-the-fly. A simple version of the former concept, online synthesis, was demonstrated in Appendix J.2.4. However, we believe that a system could automatically invoke behavior synthesis during execution, by treating it as a state primitive, no different than footstep planning or closing the fingers. Only this state primitive would have the power to alter the structure of the active behavior itself, in accordance with some formal specification.

While our approach to behavior synthesis is, in principle, robot-agnostic, we have only demonstrated it on Team ViGIR's ATLAS humanoid robot. We want to facilitate the integration of other popular robotic platforms, such as the KUKA youBot mobile manipulator, by providing state primitives that will serve as building blocks for behavior synthesis.

Finally, a new, but related, research direction we plan to pursue in conjunction with Dr. David Conner, who has moved from TORC Robotics to Christopher Newport University, is "Capability Specification". Behavior synthesis relies on a developer mapping abstract symbols (used in LTL formulas) to the system's atomic capabilities (implemented in software). Currently, this requires system level expertise. We believe that annotating the software components, that is the ROS packages, with formal specifications of their capabilities, would allow behavior synthesis to automatically generate this mapping and any associated constraints (such as the pre-conditions and post-conditions of various actions). The team intends to explore ways to formalize these capabilities in a formal yet generic manner that is amenable to automatic generation of system level behaviors based on the capabilities of the deployed sub-systems.

6. REFERENCES

This bibliography includes documents written by the team during the course of this project; these documents are included in the appendices. General references are cited in the individual papers.

[1] S. Kohlbrecher, A. Romay, A. Stumpf, A. Gupta, O. von Stryk, F. Bacim, D. A. Bowman, R. Balasubramanian and D. C. Conner, "Human-Robot Teaming for Rescue Missions: Team ViGIR's Approach to the 2013 DARPA Robotics Challenge Trials," *Journal of Field Robotics, Special Issue: Special issue on DARPA Robotics Challenge (DRC),* vol. 32, no. 3, pp. 352-377, May 2015.

[2] S. Kohlbrecher, D. C. Conner, A. Romay, F. Bacim, D. A. Bowman and O. von Stryk, "Overview of Team ViGIR's approach to the Virtual Robotics Challenge," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Linköping, Sweden, 2013.

[3] A. Romay, S. Kohlbrecher, D. C. Conner, A. Stumpf and O. von Stryk, "Template-based Manipulation in Unstructured Environments for Supervised Semi-Autonomous Humanoid Robots," in *2014 IEEE-RAS International Conference on Humanoid Robots*, Madrid, Spain, 2014.

[4] A. Stumpf, S. Kohlbrecher, D. C. Conner and O. von Stryk, "Supervised Footstep Planning for Humanoid Robots in Rough Terrain Tasks Using a Black Box Walking Controller," vol. 32, no. 3, November 2014.

[5] A. Romay, S. Kohlbrecher, D. C. Conner and O. von Stryk, "Achieving Versatile Manipulation Tasks with Unknown Objects by Supervised Humanoid Robots based on Object Templates," in *submitted to 2015 IEEE-RAS International Conference on Humanoid Robots*, Seoul, South Korea, 2015.

[6] M. Schappler, J. Vorndamme, A. T¨odtheide, D. C. Conner, O. von Stryk and S. Haddadin, "Modeling, Identification and Impedance Control of the Atlas Arms," in *submitted to 2015 IEEE-RAS International Conference on Humanoid Robots*, Seoul, South Korea, November 2015.

[7] P. Schillinger, "An Approach for Runtime-Modifiable Behavior Control of Humanoid Rescue Robots," Darmstadt, Germany, 2015.

A. VRC AND TRIALS SYSTEM PAPERS

This section embeds [2] and [1] for easy reference.

Reference [2] provides a brief overview of the system and Team ViGIR's results in the 2013 VRC.

Reference [1] provides a system overview and details Team ViGIR's performance in each task at the 2013 DRC Trials.

Overview of Team ViGIR's Approach to the Virtual Robotics Challenge

Stefan Kohlbrecher[1], David C. Conner[2], Alberto Romay[1], Felipe Bacim[3], Doug A. Bowman[3], and Oskar von Stryk[1]

www.teamvigir.org [1] *Simulation, Systems Optimization and Robotics Group, Technische Universität Darmstadt, Germany*
[2] *TORC Robotics, United States* [3] *Center for Human-Computer Interaction, Virginia Tech, United States*

***Abstract*—With the DARPA Robotics Challenge (DRC), a call to an ambitious multi-part competition was sent out to the robotics community. In this paper, we briefly summarize the approach for addressing the Virtual Robotics Challenge (VRC) where software for control and supervision of a capable humanoid robot must be developed. Team ViGIR, comprising members from the US and Germany, leveraged previous robotics competition experience and a variety of open source tools, to achieve sixth place in the VRC out of 126 registrants, thereby advancing to the next round of the DRC and obtaining an Atlas robot.**

I. Introduction

The Fukushima Daichi nuclear plant disaster once again showed that disaster response and mitigation could improve significantly if more capable robots were available for use. This motivated the DARPA Robotics Challenge[1]. In the VRC a simulated humanoid robot has to perform a series of tasks inspired by real world needs. Details of the challenge and participation models can be found online. Seven top scoring teams in the VRC received an Atlas robot being developed by Boston Dynamics and proceed to the next round of competion. For developing and integrating the complete onboard and offboard software within the few months available until the VRC, we decided to reutilize and adapt available, preferably open source, software from within and outside the team.

II. System Overview

Team ViGIRs software uses ROS as middleware [1]. While the DRC simulator[2] was locked to use the ROS fuerte version during competition, ROS groovy was used for all team software to leverage recent software advancements only available for ROS groovy, for example the MoveIt! motion planning framework [2]. With developers spread over two continents, development was tracked using the Redmine project management system, frequent conference calls, and an on-site sprint meeting preceding the VRC competition.

III. Basic Capabilities

For the VRC, the team focused on developing the basic capabilities required for operation of a humanoid robot in the competition scenarios. Due to space constraints, other software capabilities like *footstep planning*, *behavior control*, *semi-autonomous grasping* and *open-loop motion editing* are not detailed in this paper.

[1]http://www.theroboticschallenge.org/
[2]http://www.gazebosim.org/wiki/DRC

Operator Control Station (OCS) The OCS is a set of graphical user interface components that allow the operator to control the robot and visualize information about its current state and the world surrounding it. The goal for OCS design was a customizable user interface with support for different widgets that can be changed and reorganized based on the task at hand. Three main components to the OCS were used in each task: a 3D perspective view widget that allows the operator to control end effectors directly, visualize 2D and 3D reconstructions of the environment, and plan robot motion; a top-down orthographic view widget that is used for navigation and to request more information about the environment; and a camera widget that allows the operator to request images with a requested resolution from every camera in the robot. The other OCS components available to the operator include grasp control, individual joint control, and planner configuration tools. The complete setup used in the VRC is shown in Fig. 1.

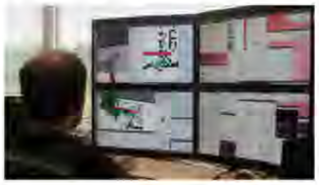


Fig. 1: OCS setup used in the VRC

Bandwidth-Constrained Communication Only very limited communication is allowed between the robot and the Operator Control Station (OCS). In the VRC, traffic was restricted by emulation software and teams could only transmit a limited amount of data to the robot (upload) and from the robot (download). The most strict scenario allowed only 7Mb of data download from the robot over the whole 30 minute mission time, making bandwidth management and careful selection of data to be transmitted a serious issue. The overhead of communication when using a single ROS Master to connect the onboard and OCS systems was prohibitive; therefore, a communications bridge using a compressed transport based on Google Protobufs and bz2 compression was developed. Bandwidth constraints were never exceeded during VRC missions.

Perception Due to bandwidth limitations, only limited amounts of perception data could be transmitted to OCS, so fusion of data and conversion to a low-bandwidth representation were very important. LIDAR point cloud data is fused using PCL tools [3] and the probabilistic octomap approach

[4], reducing data volume compared to conventional pointcloud storage. Octomap data can then be sliced in a region of interest over a given height range, resulting in a 2D occupancy grid map representation. Using this approach, map data for obtaining general situational awareness can be transmitted to the OCS with minimal bandwidth cost.

Manipulation of the environment was required for 2 of the 3 VRC tasks and thus a key capability. Limited bandwidth means that the transmission of a full 3D model of the environment is prohibitive, so collision free motion planning has to happen onboard the robot. To this end, the operator can specify a intended joint or cartesian endeffector target pose for the robot and motion planning including full 3D obstacle avoidance is performed using the MoveIt! motion planning framework [2] onboard the robot, leveraging full 3D obstacle information. Grasps for different kinds of object are template based and are generated offline using the GraspIt! [5] toolkit.

Motion Control A per joint PID controller using feedforward motion compensation was employed for simplicity and robustness. It can be used to follow joint trajectories and will also stitch trajectories for smooth transition between them. Keyframe based motions worked very well in simulation. Basic locomotion controllers were provided for optional use to teams.

IV. Task Performance

Approaches to the three VRC tasks and resulting performance are summarized. For each task, 5 instances with varying conditions were used and for each of those, up to 4 points could be scored depending on the degree of task completion.

Task 1 (Driving) After walking up to the vehicle using the capabilities presented in Section III, the driving task presented two challenges. Getting into the car was achieved by using a carefully designed motion that makes use of mechanical effects to reduce uncertainty in positioning the robot into a sitting posture in the car (Fig. 2a). Actuation of the car controls was difficult, as the seat was modeled as a sticky surface. When performing the required manipulation in contact with the environment, the robot would slip in the seat unpredictably. This resulted in loss of sitting posture in all runs.

Task 2 (Rough Terrain) Due to a bug in the optionally provided walk controller which only showed up during VRC, it was not possible to get into walking mode again when the robot has stood up after a fall. Therefore, after a fall during a task in the VRC, crawling was the only option left for locomotion. Open-loop key frame-based quadrupedal locomotion had been developed which was used in the VRC during the rough terrain task and whenever the robot had fallen. After walking up to the first gate, the robot encountered a simulated mud pit, which could not be traversed using the provided walking controller. Therefore, the robot was commanded into a sitting posture and backwards crawling was employed as the mode of locomotion for the remainder of Task 2 (Fig. 2b).

Task 3 (Hose Manipulation) The most challenging part of the hose manipulation task was aligning the hose with the standpipe after picking it up (Fig. 2c). While picking the hose up worked reliably using our approach, moving it towards the location of the standpipe was more challenging, as movement of the hose was unpredictable when dropped on the table for re-grasping with the other arm. For this reason, strategy was switched to walking with hose in hand during VRC, which allowed alignment with the standpipe in one run.

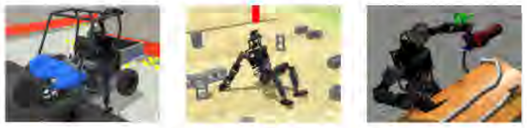


Fig. 2: VRC Tasks: a) Driving (left), b) Rough Terrain (middle) and c) Hose Manipulation (right)

V. Evaluation

From out of 126 Track B and C teams registered for the VRC, 26 passed qualification and 22 scored in the VRC. With a total of 27 points (Table I) Team ViGIR was ranked 6th behind 5th and 4th with 29 and 30 points and before 7th and 8th with 25 and 24 points. The distribution of points scored in the three tasks was quite similar for teams ranked 3rd to 8th.

	Run 1	Run 2	Run 3	Run 4	Run 5	Total (Max. 20)
Task 1	0	1	1	1	0	3
Task 2	4	2	4	4	4	18
Task 3	1	1	2	1	1	6

TABLE I: Team ViGIR VRC scores.

VI. Conclusion

In this paper, a short overview of Team ViGIRs entry to the Virtual Robotics Challenge is presented. The team is currently transfering and adapting the software developed for the VRC to the actual Atlas robot and extending it. It plans to provide as much as possible of their software developments as open source to foster progress in rescue robotics.

Acknowledgment

The authors would like to thank all team members who contributed and continue to contribute to the success of the team: Tony Angell, Florian Berz, Lindsay Blassic, Thorsten Graber, Jesse G. Hurdus, Alex Little, Johannes Meyer, Jochen Mück, Karen Petersen, Philipp Schillinger, Dorian Scholz, Jacob Sheppard, Alexander Stumpf and Ben Waxler.

References

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[2] S. Chitta, I. Sucan, and S. Cousins, "Moveit!" *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[3] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[4] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com. [Online]. Available: http://octomap.github.com

[5] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *Robotics & Automation Magazine, IEEE*, vol. 11, no. 4, pp. 110–122, 2004.

Preprint of the paper which appeared in Journal of Field Robotics,
Volume 32, Issue 3, pages 352-377, May 2015,
First published online 4 Dec 2014.

Human-Robot Teaming for Rescue Missions: Team ViGIR's Approach to the 2013 DARPA Robotics Challenge Trials

Stefan Kohlbrecher, Alberto Romay, Alexander Stumpf, Anant Gupta, Oskar von Stryk
Simulation, Systems Optimization and Robotics Group, CS Dept. Technische Universität Darmstadt
Hochschulstrasse 10, 64289, Darmstadt, Hesse, Germany
{kohlbrecher,romay,stumpf,gupta,stryk}@sim.tu-darmstadt.de

Felipe Bacim, Doug A. Bowman
Center for Human-Computer Interaction, Virginia Tech
2202 Kraft Drive, KWII Building (0106), Blacksburg, VA 24061-0106, United States
{fbacim,dbowman}@vt.edu

Alex Goins, Ravi Balasubramanian
Robotics and Human Control Systems Lab, Oregon State University
1891 SW Campus Way, Dearborn Hall, Corvallis, OR 97331, United States
goinsa@onid.orst.edu, ravi.balasubramanian@oregonstate.edu

David C. Conner*
TORC Robotics
2200 Kraft Drive Suite 2050, Blacksburg, VA 24060, United States
conner@torcrobotics.com

Abstract

Team ViGIR entered the 2013 DARPA Robotics Challenge (DRC) with a focus on developing software to enable an operator to guide a humanoid robot through the series of challenge tasks emulating disaster scenarios. The overarching philosophy was to make our operators full team members and not just simple supervisors. We designed our operator control station (OCS) to allow multiple operators to request and share information as needed to maintain situational awareness under bandwidth constraints, while directing the robot to perform tasks with most planning and control taking place onboard the robot. Given the limited development time we leveraged a number of open source libraries in both our onboard software and our OCS design; this included significant use of the Robot Operating System (ROS) libraries and toolchain. This paper describes the high level approach, including the OCS design and major onboard components, and describes our DRC Trials results. The paper concludes with a number of lessons learned that are being applied to the final phase of the competition.

1 Introduction

In the spring of 2012, the United States Defense Advanced Projects Research Agency (DARPA) proposed the DARPA Robotics Challenge (DRC)[1] to accelerate development and evaluation of disaster response robots that have the capability for early response and mitigation of disasters. This effort was partly motivated by the earthquake and tsunami that struck the Tohoku region of eastern Japan on March 11, 2011, and led to subsequent damage to the Fukushima Daiichi nuclear plant. The DRC is designed to mimic the tasks that might be required of a robot (Nagatani et al., 2013) to respond to the initial damage and avert subsequent catastrophes.

A major focus of the DRC is the development of an approach that leverages the complementary strengths and weaknesses of the robot system and human operator(s). While full bandwidth and update rate access to all sensory systems is available onboard the robot system, cognitive and decision making abilities of a human operator are still vastly superior for the foreseeable future. This is especially true for disaster scenarios, as only very limited assumptions about their structure can be made beforehand, in contrast to structured man-made scenarios like kitchens, where semantic knowledge of their structure can be leveraged for highly autonomous robots operating in them (Blodow et al., 2011). Team ViGIR was formed with a particular focus on developing the enabling technologies for the human-robot team.

There are a number of schemes for classifying the autonomy of a system(Bruemmer et al., 2002; Scholtz, 2003; Yanco, 2004; Desai and Yanco, 2005; Huang et al., 2007). In our system, the robot is never fully autonomous and therefore has a low score for *human independence*(Huang et al., 2007). The human members of the team function as *supervisors* who set high level goals, *teammates* who assist the robot with perception tasks, and *operators* who directly change robot parameters to improve performance(Scholtz, 2003); as these roles change dynamically during a set task in our system, we will use the term *operator* generically. Following (Bruemmer et al., 2002), we rarely operate in *teleoperation* where we directly control a joint value, and primarily operate in *shared* mode where the operator specifies tasks or goal points, and the robot plans its motions to avoid obstacles and then executes the motion only when given permission. Even when executing a footstep plan in *autonomous* mode, the operator still has supervisory control of the robot and can command the robot to stop walking at any time, and safely revert to a standing posture.

Team ViGIR entered the DRC as a 'Track B' team competing in the DARPA Virtual Robotics Challenge (VRC). Initially, Team ViGIR was composed of TORC Robotics[2], the Simulation, Systems Optimization and Robotics group at Technische Universität Darmstadt (TUD)[3], and the 3D Interaction Group at Virginia Tech[4]. With only 8 months from program kick-off to the first competition, the team focused on providing basic robot capabilities needed for the three tasks in the VRC. A short overview of our VRC approach is available in (Kohlbrecher et al., 2013).

While the tasks and requirements for the VRC were based on those anticipated in a real scenario, there were important differences: sensor noise was low and more predictable, simple friction models were used, there was no need for calibrating sensors or joint angle offsets for the robot, and the environments were known ahead of time. The dynamic model used for simulating the Atlas robot was available to the teams, enabling the use of model based control approaches without prior model/systems identification. In spite of these simplifications, Team ViGIR chose to take the long view and develop our human-robot interface (HRI)(Goodrich and Schultz, 2007) with an eye toward operation in more complex and unknown environments. The VRC results demonstrated that our approach and developed capabilities were competitive, enabling us to place sixth out of 22 qualifying teams that participated in the VRC competition. While some approaches, such as our technique for getting into the car, were not transferable to a real robot, the overall approach and interaction philosophy was readily transferable. This included our basic design concept for the operator control station (OCS), the footstep and manipulation planners, our approach to grasping, and our low bandwidth

[1]http://theroboticschallenge.org
[2]http://www.torcrobotics.com
[3]http://www.sim.informatik.tu-darmstadt.de/en/index/
[4]http://research.cs.vt.edu/3di/

communication software.

After our success in the VRC, Team ViGIR received the Atlas humanoid robot shown in Fig. 1a, which was developed by Boston Dynamics, Inc. (BDI). With only four months to modify our software and learn the capabilities and intricacies of the robot before the DRC Trials, Team ViGIR added representatives from the Robotics and Human Control Systems Lab at Oregon State University[5] to supplement our experience with real world grasping and manipulation. As a joint US-European team, with a nine hour time differential across the team, a crucial part of our successful participation in the DRC was establishing a workflow that enabled seamless cooperation over large distances. To achieve this, the team relied on the Redmine (Koch, 2010) open source project management application. Redmine provided both a wiki for documentation, an issue tracker as well as integration with the *Git* (Loeliger and McCullough, 2012) repositories containing source code. Frequent teleconferences were used to coordinate among team members.

As development time between initial kick-off and the DRC Trials was limited, development was split into reliable core modules that were crucial to the ability to perform the competition tasks and additional modules that could potentially improve performance but also carried a greater risk of failing due to environment conditions or limited time for testing.

There are a number of publications coming from DRC research, including other competitors in the VRC (Koolen et al., 2014; Tedrake et al., 2014), as well available summaries of DRC Trials results (Fallon et al., 2014a). There are also publications on specific system components like footstep planning (Deits and Tedrake, 2014; Stumpf et al., 2014), drift free state estimation(Fallon et al., 2014b), whole body motion planning (Dai et al., 2014), and user-guided manipulation (Alunni et al., 2013).

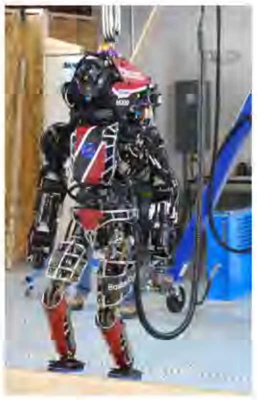
In this paper we present an overview of our system and some of our approaches that were developed, including some that were evaluated but could not be successfully integrated in time for the competition. Section 2 presents an overview of the system architecture with a focus on the hardware layout and communications. In Section 3, we discuss our philosophy and design of the the HRI. Perception and sensing of robot state and situational awareness is discussed in Section 4. Next, we discuss the robot control and planning systems in Sections 5 and 6, followed by a discussion of grasping in Section 7. The paper discusses our results at the DRC Trials in Section 8, offers lessons learned in Section 9, and concluding thoughts in Section 10.

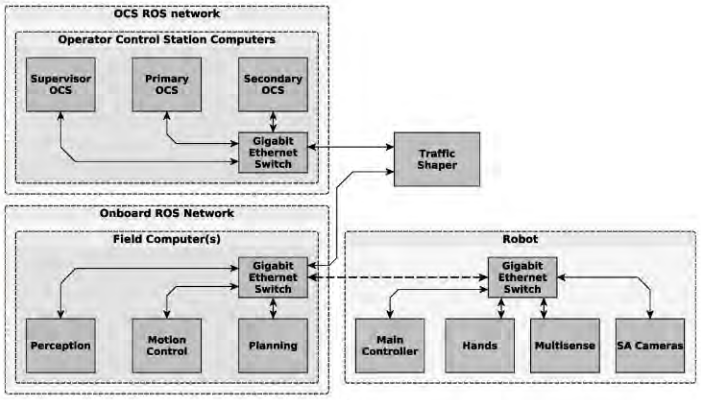
2 System Architecture

The system hardware is composed of three major sub-systems as shown in Fig. 1: the robot, the “onboard” computers, and the OCS hardware. Team ViGIR uses the Atlas robot that includes a robot control computer and Ethernet switch that handles communications with the sensors and robot hands. The “onboard” or “field” computers run the robot control and perception software developed by Team ViGIR. In the future, these field computers will be carried onboard the robot itself, but for ease of development leading up to the DRC Trials DARPA allowed these to be separate computers connected to the robot via a 10 GB/second fiber optic network connection. During the DRC Trials, the onboard computers were connected to the OCS computers via a 1 GB/second network connection that passed through a network traffic shaper; the traffic shaper introduced communication restrictions intended to mimic the effects of poor wireless communications. All operator interactions with the robot occurred through the OCS hardware, with commands sent to the onboard software via the traffic shaper connection.

Early on, the team chose to base the system software on the open source Robot Operating System (ROS)[6] (Quigley et al., 2009) to take advantage of the available libraries and infrastructure. The system communications use ROS as the middleware, with two separate ROS networks being used during competition. One ROS network handles communications between the OCS computers, while the second ROS network handles

[5]http://mime.oregonstate.edu/research/rhcs/
[6]http://www.ros.org


(a)



(b)

Figure 1: System overview: a) robot with tether connecting to the base station in the background, b) architecture diagram including the robot, Onboard, and OCS sub-systems. The dashed arrow shows the 10 gigabit fiber optic link between robot and field switch, while other arrows indicate 1 gigabit Ethernet links.

communications on the onboard computers. Section 2.4 describes the motivation for this choice, and the implementation of our Communications Bridge (CommsBridge) between the ROS networks. To be able to leverage latest open source software, we closely followed the ROS release schedule and used the ROS Hydro version during the DRC Trials. With ROS changing the underlying build system to *catkin*, the team uses a two workspace setup with one *rosbuild* workspace for packages using the legacy *rosbuild* build system and one workspace using the new *catkin* build system. Using this setup allows the most flexibility in using both old and new open source software for ROS.

The remainder of this section gives an overview of each major sub-system, with a focus on hardware layout.

2.1 Robot Hardware

The Atlas system is a hydraulically actuated anthropomorphic robot developed by Boston Dynamics Inc. (BDI). The robot has 28 actuated degrees of freedom (DOF), stands 1.88m tall and weighs approximately 150kg. There are 6 DOF per arm and leg, 3 DOF for torso motion, and one DOF for head pitch motion. For the DRC Trials, the robot was used in a tethered configuration as shown in Fig. 1, with the tether providing electrical power, coolant flow, and a 10 gigabit Ethernet connection .

The robot is equipped with a number of sensors. The main external sensor is a Carnegie Robotics[7] Multisense SL sensor mounted as the head. This sensor uses both a Hokuyo UTM-30LX-EW LIDAR mounted on a slip ring for continuous rotation and a stereo camera system that performs stereo disparity computation on a onboard FPGA. Both sensors are calibrated against each other using a calibration approach supplied by the manufacturer. Additionally, there are two situational awareness (SA) cameras mounted below the head with high field of view fisheye lenses. The robot provides a pose estimate based on an internal IMU and internal joint sensing.

For the DRC Trials, the robot was supplied with three options for hands (see Fig. 2a, 2b, 2c): a 3 fingered robotic hand with 5 DOF from iRobot[8], a 4 fingered robotic hand with 12 DOF and stereo camera system from Sandia National Labs[9], and a simple hook hand developed by BDI.

[7]http://www.carnegierobotics.com
[8]http://www.irobot.com
[9]http://www.sandia.gov

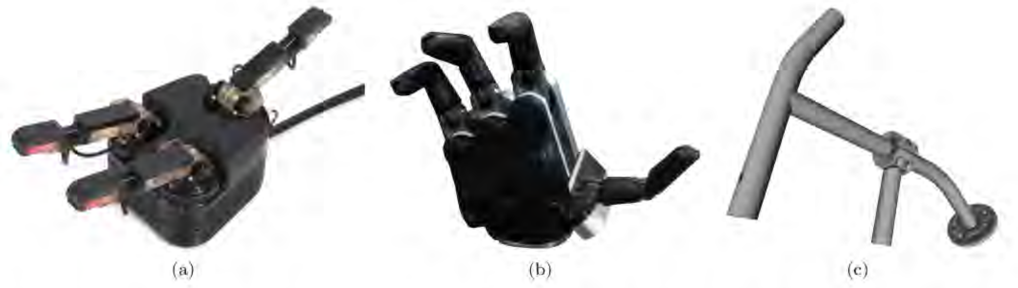


Figure 2: Robotic Hands: (a) iRobot (Odhner et al., 2013). (b) Sandia (Sandia National Laboratories, 2014). (c) Hook.

2.2 Onboard System

Onboard software has to provide the basic skills of the robot system required in the DRC, characterized by locomotion and manipulation in challenging environments, and employing a constrained bandwidth connection to human operators. Onboard software is thus developed with a focus on two requirements. The first requirement is operator independent onboard processing for autonomous and semi-autonomous functionality and the second is providing sensor data and interfaces to remote operators that allow them to gain situational awareness and interact with the robot over the constrained bandwidth link.

For the DRC, there is no strict limitation on computing power; during the DRC Trials, the onboard software ran on field computer machines located in the team garage. Our setup uses three desktop computers (Intel Core i7 3770, 3.4 GHz, 16GB RAM). Software modules are distributed among these machines according to functionality, with one machine each dedicated to motion control, perception, and planning. As the robot performs low-level closed-loop joint control using its internal main controller computer, a hard real-time computing system is not required for onboard software; a standard 64-bit Ubuntu 12.04 kernel is used for the Team ViGIR software.

Leveraging proven best practices within the ROS ecosystem, the robot is modeled using the URDF format and standard tools like the *tf*[10] library are used to provide a basic robot setup and model that could directly be used by many higher level components, like the motion planning system. The robot model is constructed at runtime using the *xacro*[11] macro toolset, to allow for flexibility (e.g. for choosing from available hand options on a per arm basis).

2.3 Operator Control System

The OCS consists of three computers (Core i7 3770 3.4GHz, 16GB RAM) arranged as two auxiliary operator stations, which used two 27" 1080p monitors side-by-side powered by a single-GPU NVIDIA GTX660, and the main operator interface station, which used four 27" 1080p monitors powered by a dual-GPU NVIDIA GTX690. The main station is placed in between the two auxiliary stations. The auxiliary station on the left is used as the main point of connection between the onboard and OCS computers, and allows the operators to monitor communication status at all times. The main station is used by our primary operator to plan tasks based on the 2D and 3D visualizations and issue commands to the robot. Finally, the right auxiliary station is used by our secondary operator to request and validate existing perception data used by the primary operator. The setup used in competition can be seen in Figure 3.

[10]http://wiki.ros.org/tf
[11]http://wiki.ros.org/xacro

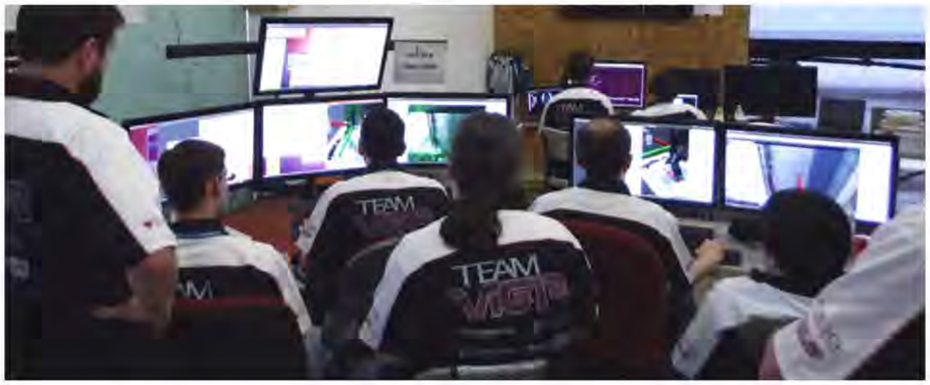


Figure 3: Multiple operator OCS setup used during the 2013 DRC Trials.

Since we use ROS, we are able to take advantage of the several existing tools that it provides, mainly *rviz*[12] and *rqt*[13]. In the development of our main widgets (described in section 3.3), we use a combination of *librviz*, *Qt*, and *Ogre*. Leveraging the existing tools in librviz for visualizing 3D data communicated via ROS was very important given the short amount of time to implement the system before the DRC Trials. All 3D views in the OCS use existing and customized versions of rviz plugins (e.g., adding support to our own methods for picking geometry), in addition to completely new plugins that implement some of our OCS' unique features (e.g., templates). For the development of simple 2D widgets, we use rqt extensively; this allowed us to quickly prototype widgets during development that act as windows for specific controllers on the onboard side (e.g., footstep controller parameters).

2.4 Bandwidth-Constrained Communication

During the DRC Virtual Challenge and DRC Trials, a network traffic shaper was used to modulate the available communications between the robot onboard software and the OCS. This was intended to mimic the communication restrictions likely during a disaster response scenario. During the VRC, communication connectivity was good, but there was a strict bandwidth and total data budget that varied for different tasks. During the DRC Trials, the communications alternated every minute between good comms (1,000,000 bits/s, 50 ms latency) and bad comms (100,000 bits/sec, 500 ms latency). During bad communications, the system buffered excessive packets introducing significant delay in data transmission if the system attempted to transmit too much data. As only limited bandwidth between the OCS and onboard software is available, the capability to leverage this link to transfer useful data for the human operator to gain timely situational awareness is crucial. Additionally, during the initial VRC design there were discussions of dropping packets, therefore Team ViGIR designed the communications system to be tolerant of dropped packets and broken communication links.

As stated above our team chose to use ROS for our communications middleware. The ROS system uses a publisher/subscriber model with a centralized *roscore* to coordinate communications between ROS nodes. This is not suitable for use with the communications challenges defined above as the system cannot tolerate loss of communications to the centralized *roscore*. For this reason, the team chose to use two separate ROS networks for the onboard and OCS software and develop a custom communications bridge (CommsBridge); this setup was used during the initial VRC and throughout the DRC Trials.

The CommsBridge uses both TCP and UDP to communicate between the OCS and onboard software. For

[12] http://wiki.ros.org/rviz
[13] http://wiki.ros.org/rqt

an OCS command going to the onboard software, the OCS-based CommsBridge module subscribes to the relevant ROS topic, sends the data across the network, and republishs the data as a ROS topic on the onboard side; for the reverse, the onboard software subscribes to perception, state, and status messages, sends the data across the network, and republishs for use by the OCS. The CommsBridge is robust to communication dropouts, and automatically reconnects as needed if a process is restarted or communications are dropped. As the same topic names are used on both sides, the setup allows seemless testing as a single ROS network if desired.

For high-rate robot state information from the onboard software to the OCS, the CommsBridge uses UDP packets and a custom message packing that uses scaled integer representations in lieu of most floating point values, and custom data compression based on Google Protocol Buffers[14]. The data for robot position in the world, all joint positions (including fingers), and other state data is sent as a single UDP datagram. As only the latest robot state data is relevant for operator situational awareness, the CommsBridge system is tolerant to dropped packets. All control calculations, including planning and trajectory interpolation, are performed on the onboard side; thus, the operator does not require continuous velocity updates. The operator's situational awareness is based on intermittent position updates.

For operator commands, data sent only on transition, or large data that is not tolerant to dropped packets the CommsBridge uses TCP-based communications. The team developed a templated message handler that uses the message type, topic name, and direction (e.g. OCS_TO_FIELD or FIELD_TO_OCS) and automatically defines the appropriate communications for any valid ROS message. The approach uses the open source blob_tools[15] developed by TU Darmstadt to serialize the messages into "data blobs" and compress using the standard bzip2 compression library. On receipt of the TCP message, the data is uncompressed, deserialized into the appropriate data structure, and republished on the appropriate message topic.

Image data is handled by a special CommsBridge node denoted ImageBridge. On the onboard side, raw camera images are communicated using the ROS image transport package[16], and handled by special crop-decimate nodes that permited the operator to request variable resolution images. The operator can also request video data at specified frame rates using Theora[17] compression. Generally, the full image or video frame is displayed in a greatly reduced resolution while a defined region of interest (ROI) can be requested at higher resolution as shown in Fig. 4. A separate ImageBridge is created for both the full and cropped images for each camera feed. The ImageBridge forwards OCS image requests that specify the desired ROI, resolution, and frame rate to the appropriate crop decimate node, and sends the compressed image structures using the blob_tools package. ROS CameraInfo messages used by the image transport protocol are only communicated across the ImageBridge on change, but are automatically republished on the OCS side to permit proper visualization of the image in 3D representations.

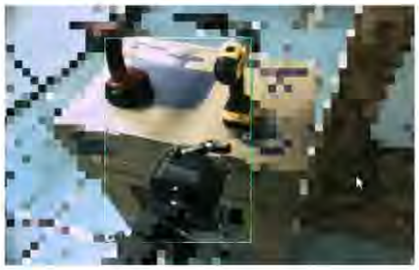


Figure 4: OCS Image view showing variable resolution views for grasping drill during wall task practice.

[14]https://code.google.com/p/protobuf/
[15]https://github.com/tu-darmstadt-ros-pkg/blob_tools
[16]http://wiki.ros.org/image_transport
[17]http://wiki.ros.org/libtheora

3 Human-Robot Interaction

A key goal in our system design was to combine robot capabilities with the rapid and finely tuned perceptual and decision-making abilities of human operators in order to complete the challenge tasks quickly and efficiently. In order to achieve this goal, based on the principles of human-robot interaction (HRI) (Goodrich and Schultz, 2007), we designed a powerful User Interface (UI) that allowed the operators to communicate with and control the robot, maintain situational awareness, and make quick decisions. However, there are several challenges involved in designing a UI that provides this level of interaction between operator(s) and the robot.

3.1 Design Challenges

With an overwhelming amount of sensor data and information coming from the robot, our first major challenge was to decide what sort of information operators needed, as well as to determine when and how to display and interact with that information. Given ideal conditions for processing and communications, it would be easy to provide the user with all the processed and raw data coming from the robot at all times. However, it would also be difficult to understand and manage that information. Allowing the user the flexibility to select the data that is needed to perform each task (e.g., 2D map data to perform navigation or point clouds to perform fine hand manipulation), was one of our main goals.

Another challenge in designing the UI for the DRC was the limited bandwidth that was provided for communication between onboard computers and the operator interface. This meant that while there was a large amount of data being generated by the robot, only a small portion could be sent over to the operator station. This aspect of the DRC played an important role in the design of our UI, since we would not only have to provide flexibility in what data should be displayed at any given time, but also how much of that data would be transmitted (e.g., controlling resolution of images/point clouds).

In addition, because of the 3D nature of the tasks and data collected by the robot, operators would spend a large amount of time looking at and interacting with 3D visualizations. The location of the robot with respect to other objects has to be known when navigating in a real environment. The positions of the hands and fingers have to be accurately displayed when performing fine manipulation tasks. All tasks in the DRC required users to perform complex spatial judgments, and all of them required multi-DOF interactions with the robot and the environment. Viewing 3D data on desktop displays and interacting with 3D environments through 2D input devices is inherently difficult (Van Dam, 1997), so designing both the visualizations and interactions was a major challenge.

Finally, while the DRC tasks served as a reference and benchmark for our solution, we strived to provide maximum generality in our UI solution. Custom design of our UI for the specific tasks that we had to accomplish during the DRC Trials was a possibility (since we had the complete task specifications ahead of time), but we knew that this approach would not work in the DRC Finals, where tasks were less well-defined and in real-world scenarios, which might be completely unknown.

3.2 Design Goals

To overcome these challenges, we designed a set of graphical UI components (widgets) that allowed the operator to control the robot and visualize information about its current state and the world surrounding it. The main goal was to build a customizable UI with various widgets that could be changed and reorganized based on the task at hand, providing variable amounts of data and control, allowing operators to perform under any situation specified in the challenge. In order to achieve this, we first created rough and then detailed information design and interaction design storyboards for each of the tasks. This included scenarios and sketches (Rosson and Carroll, 2002) for the UIs with the features we envisioned, which helped us to determine requirements for completing each task with a minimal amount of bandwidth. Prototyping and weekly meetings to evaluate the usability and functionality of the UI helped us to improve on the design.

In addition, while our initial design ideas would allow a single person to control every aspect of the robot and the tasks, we ultimately decided to design for multiple operators during the DRC Trials due to the complexity of the tasks and the limited amount of time to complete them. As shown in (Murphy and Burke, 2005), the use of multiple operators can increase overall robot system performance in search and rescue environments.

3.3 OCS Overview

A detailed description of our Operator Control Station (OCS) design is not possible here; we focus on two major features and the primary views/controls we developed. The first major feature is the use of 3D templates. Templates (3D models of important objects/features in the environment) allow the primary and secondary operators to annotate perception data with semantic information. For example, if the operator sees a known object in the point cloud (e.g., a tool), he can insert a template representing that tool in the 3D view at that location, thus informing other operators and the onboard systems about that object (Figure 5a). Based on the location of these known objects, the operator can then plan manipulation of the object using grasps and positions that maximize grip strength or reachability.

A second major feature, the "ghost" or simulation robot, is a transparent duplicate of the ATLAS robot visualization. The ghost robot allows the primary operator to plan and validate motions before executing them with the physical robot (Figure 5b). It is also color coded to give the operators feedback about the internal state of the onboard systems, such as collision checking for motion planning, to prevent unexpected actions. Both of these features can be used in and visualized at any of the views described below.

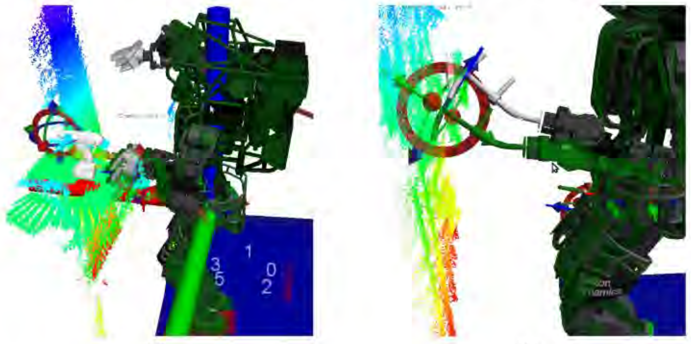

(a) (b)

Figure 5: Two major OCS features: (a) Templates. (b) "Ghost" robot.

The OCS used by the main operators contains three main components: *main view*, *map view*, and *camera view*. The *main view* is a widget that is primarily used for visualization of 3D data and fine manipulation control. It allows the operator to control end effectors, visualize the 2D and 3D reconstructions of the environment, annotate these visualizations with templates, and plan robot motion by controlling the ghost robot. The single 3D view can also be split into four 3D visualizations with different points of view and settings (orthographic/perspective), to facilitate spatial judgments and aid depth perception. The *map view* is a top-down orthographic view widget that is used for navigation and to request more information about the environment. The operator can select a region of interest in the environment by clicking and dragging to create a box selection, and then choose what type of data is needed (e.g., grid map, LIDAR/stereo point clouds). Fine control over the amount of data being requested helps in reducing the amount of information transmitted over the network and shown on the screen. Finally, the *camera view* allows the operator to request single images or video feeds with varying resolution from every camera on the robot, with up to four images displayed at a time. Three-dimensional data can be overlaid on the images to validate the sensor data and prevent errors due to drift in position/orientation estimation.

The OCS also contains specialized components for controlling specific commands or to give flexibility to overcome unknown situations. The special widgets included the grasp control interface for each hand, which gave the operator access to pre-defined hand poses for templates placed in the environment, as well as widgets to provide individual joint control and planner configuration tools. The widgets and layout used in a manipulation task can be seen in Figure 6.

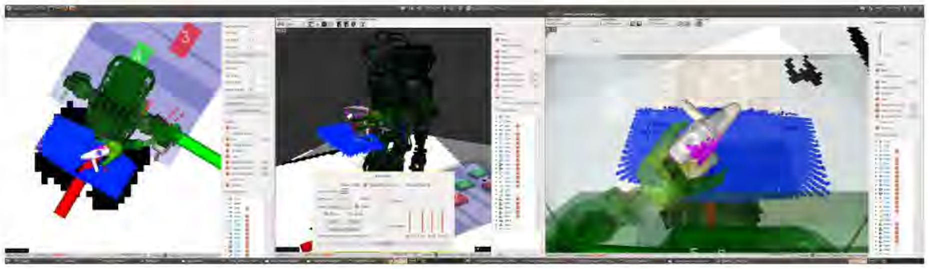


Figure 6: Overview of OCS layout used during a manipulation test, with the map view on the left, the main view in the center with grasp control popup, and the camera view on the right. All major controls are readily accessible through hotkeys or toolbar buttons that hide or display individual widgets; additional improvements are currently being incorporated to move the realization toward the initial concept and build upon lessons learned.

Finally, as mentioned in Section 3.2, one of our main goals was allow two or more operators to interact with the robot system using separate stations. In order to achieve this, we use a model-view-controller software pattern (Krasner et al., 1988): the model is instantiated as a set of ROS nodes that hold all the information about templates, the robot, images and user data; the views display the data to the operators; and the controllers allow the users to interact with and change the model. Thus, it is possible for multiple OCS instances to be active simultaneously. Since the two main operator tasks are requesting sensor data and commanding mission actions, we decided to separate these tasks per operator. While a primary operator evaluates perception data and controls physical actions of the robot, the secondary operator can then be responsible for other tasks, such as managing and requesting sensor data to provide situational awareness to the primary operator. While our team only had two active operators during the DRC Trials, our OCS implementation supports the addition of more operators that can supervise the actions of the main operators or even perform parts of the tasks themselves.

When designing OCS components, we followed a few guiding principles based on both best practices in HRI and our own usability evaluation of the UI used in the VRC. Our design process included weekly meetings and constant evaluation of the OCS. We wanted to minimize mouse movement by providing the ability to perform actions directly in the 3D environment, as well as providing keyboard shortcuts for the most used actions, such as requesting images or applying motions to the robot. All views can overlay multiple sensor data in same 3D environment to provide situational awareness, and the visibility of each piece of data could be toggled on/off to reduce visual clutter. Finally, we wanted to reduce the need for the operator to process detailed streams of text information. While detailed logs and complete information about errors and the state of the system is available at all times, we decided to create a widget that informs the operator about the state of different controllers at a glance, with a simple color scheme (green - successful, yellow - processing data, red - failed). This allows to quickly check the state of onboard systems.

During the limited development time prior to the DRC Trials, a number of features were being developed in parallel with the OCS development. The use of *rqt* and *rqt_python* allowed rapid development of a number of widgets required for proper configuration of specific components in our system (e.g., configuration of our footstep planner parameters, and the glance hub mentioned above); unfortunately, a significant number of these widgets were not properly integrated into our design (e.g., selection of pre-canned poses, grasp control, and ghost robot configuration and control) due to limited resources. Thus, the version of our OCS used during the DRC Trials suffered from a major issue: clutter caused by the use of too many windows and buttons (as seen in Figure 7). While some of these windows were necessary to provide access to required functionality, the resulting clutter caused our OCS to have a very steep learning curve, and caused a big gap between our original intent with its design and the actual implementation. Moreover, while no formal usability studies were conducted to verify how big of an impact this had in operator performance, it is clear that many of these extra windows fail to comply to best practices in HCI and HRI (e.g., too many buttons that look exactly the same on the pre-canned poses widget). Because of this, properly integrating all existing features into our OCS design is one of our major goals going forward toward the DRC Finals.

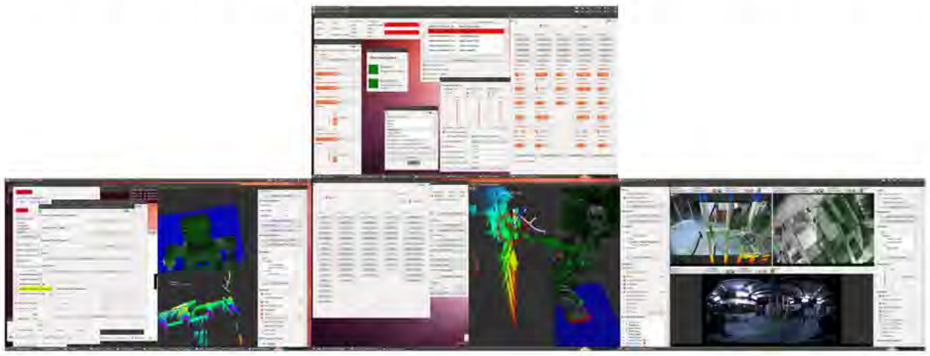


Figure 7: Overview of widgets used in the 2013 DRC Trials, with the map view on the left screen, the main view in the center, the camera view in the lower right, and all other OCS widgets spread throughout the available screen space. Compare this cluttered version with our current version shown in 6.

4 Perception

The perception system has to provide perceptual information to two different sinks: the onboard software components and the OCS. For onboard use, bandwidth constraints are not relevant and therefore information from all sensors can be integrated at full bandwidth. Data transfer to the OCS on the other hand is severely constrained, therefore the perception system must support selective data query and transfers.

4.1 Worldmodel Server

The Worldmodel Server component preprocesses, collects and aggregates sensor data and makes it available to both onboard and OCS system components. Leveraging established open source tools like Point Cloud Library (PCL) (Rusu and Cousins, 2011) and Octomap (Hornung et al., 2013), the worldmodel server allows queries of information about the environment with flexible level of detail and bandwidth consumption.

As described in Section 2.1, three dimensional sensing is provided by the sensor head, providing point cloud data both via the Hokuyo UTM-30LX LIDAR and an integrated stereo camera system. As the stereo camera system has a smaller field of view and is sensitive to lighting conditions, LIDAR data is used as the main source for creating a 3D geometry model of the environment onboard the robot. To achieve this, the planar scans of the LIDAR have to be preprocessed and aggregated, resulting in full 3D point clouds. The following preprocessing steps are employed: First, scan data is filtered for spurious measurements commonly called shadow points or mixed pixels that occur at depth discontinuities (Tuley et al., 2005) using the shadow point

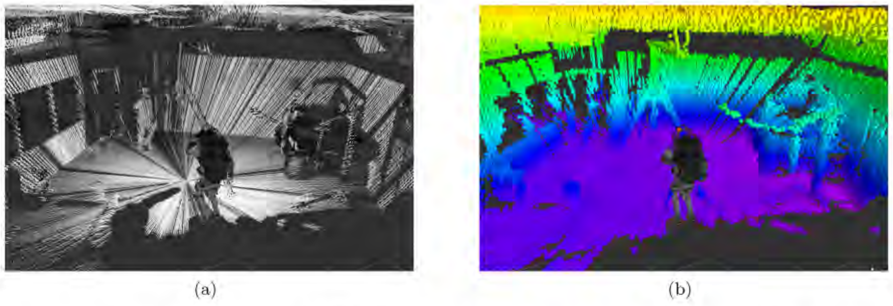


Figure 8: Sensor data processing: (a) LIDAR data with intensity information. (b) Resulting octomap representation.

filter available for ROS [18]. The filtered scan is then converted to a point cloud representation. During this process, the rotational motion of the LIDAR on the slip ring is considered and high fidelity projection is employed, transforming every scan endpoint separately. In a last step, parts belonging to the robot have to be filtered out of LIDAR data. To increase robustness against errors in kinematics calibration, a specialized robot geometry model uses simplified and enlarged collision geometries for self filtering purposes. LIDAR scans are saved to a ring buffer along with snapshots of coordinate frames used within the system. By employing this method, aggregate point clouds relative to different coordinate frames can be provided on request. A ROS API allows querying the world model via both ROS topics or services and flexibly retrieving region of interest point cloud or octomap data relative to various coordinate frames.

The primary onboard 3D geometry model is created using Octomap, a volumetric, probabilistic approach using an octree as a backend. Using this approach, the environment representation maintained onboard can be updated efficiently and in a probabilistically sound way. Even in case of changes in the environment or drift in state estimation, the environment model is updated accordingly and maintains a useful representation.

The Octomap environment model provides the main geometry representation and is used for multiple purposes. Using ray casting, distances to geometry can easily be determined. This feature can be used from within the OCS to perform ray cast distance queries against onboard geometry. In this case, only the ray cast information has to be transmitted to the robot and the distance information is transmitted back, utilizing very low bandwidth. The capability to request regions of interest (ROIs) of the environment model allows to transfer small ROI geometry over the constrained connection on operator demand and also makes geometry available to other modules, like the planning system. Similarly, it is possible to request 2D grid map slices of the Octomap representation, aggregating 3D data into a 2D grid map. Using compression, this representation is very compact and often sufficient for operators to gain situational awareness or for motion planning.

4.2 Object Detection and Pose Estimation

To perform manipulation tasks, object identification and pose estimation is required. For the DRC, we use a library of geometry templates for different objects as described in Section 3.3. Using the UI, the operator can insert a template into the 3D representation of the environment and adjust the 6DOF transform of the template. Using both camera imagery and 3D point cloud data from LIDAR and stereo camera, the template can be aligned to this sensor data. This 6DOF pose data is transmitted to the onboard systems and used by the planning and grasping control systems (see Section 7).

[18]`http://wiki.ros.org/laser_filters#ScanShadowsFilter`

An automatic object detection and pose estimation system using the Object Recognition Kitchen (ORK)[19] framework as a backend and adapting the LINEMOD (Hinterstoisser et al., 2011) approach for DRC tasks has been developed within the team. This was not used during the DRC Trials due to time constraints and challenges in adjusting the system to work under outside light conditions. We intend to further explore automatic pose estimation during the next phase.

4.3 Kinematics Calibration

Joint angle sensors of the Atlas robot exhibit a joint angle bias offset as well as a scale factor offset with respect to the true joint state. Without calibration of these effects, the joint kinematics configuration as measured by the joint sensors and the true configuration differ, making accurate manipulation difficult. For kinematics calibration a proven bundle adjustment approach previously used on the PR2 robot (Pradeep et al., 2014) available via the *calibration*[20] ROS package is adapted for use with the Atlas robot. To perform calibration, a calibration checkerboard is attached to the arm end effectors. A dataset comprising multiple camera views of the checkerboard using different arm joint configurations is then recorded. Due to the bundle adjustment approach employed by the calibration system, the transform between checkerboard and end effector does not have to manually specified, but can be estimated in a first calibration step, keeping the other arm kinematics parameters to be estimated afterwards fixed. A second calibration step is then employed to jointly optimize for the checkerboard transform as well as the kinematics parameters. Fig. 9 shows calibration results for the right arm. The reprojection error is significantly reduced after calibration. The cause for the remaining error will be investigated in future work.

5 Motion Control

Although the use of hydraulics in heavy machinery is quite common, there are relatively few examples of hydraulically actuated humanoid robot systems in comparison to the large number of electrically actuated robots used both for research and in commercial applications. While advantageous in terms of bandwidth and force magnitudes that can be generated, hydraulic actuation poses significant challenges for control of the Atlas robot system. Given limited development time, Team ViGIR chose to focus on position/velocity based joint control techniques using the BDI interface.

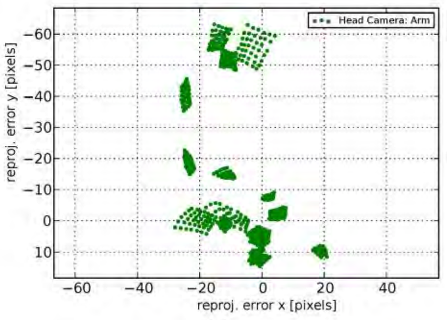
5.1 Friction identification

Friction effects in joints have significant impact on control performance, for example when applying torques for gravity compensation. In hydraulic actuators, friction is known to have significant impact (Lischinsky et al., 1999) and multiple approaches of different complexity can be used to compensate for it.

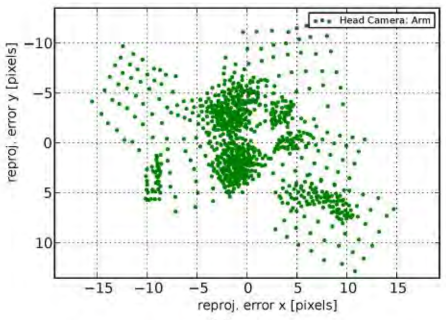
Fig. 10a-e shows the Stribeck curves for five of the six left arm joints. To measure friction, each joint was moved individually while leaving the remaining joints in a fixed position to avoid any coriolis and centrifugal effects. Using a PI-controller, each joint was moved with various, but constant velocities. In periods of constant velocity, the joint torque was measured via the built-in pressure differential sensors in the joints and averaged to cancel out sensor noise. Using model-based gravity compensation, gravity effects were partly compensated. All curves differ in shape, clearly showing that friction identification has to be performed for each joint separately. The error bars in the plots show the standard deviation for the measured friction, indicating high variance in some measurements.

In a second experiment, the valve current was commanded to constant values. The result of this experiment is shown for the left elbow joint in Fig. 10f. Although constant current values are applied, measured torques

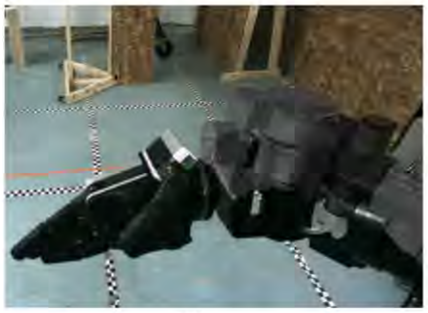
[19] http://wg-perception.github.io/object_recognition_core/
[20] http://wiki.ros.org/calibration



(a)



(b)


(c)

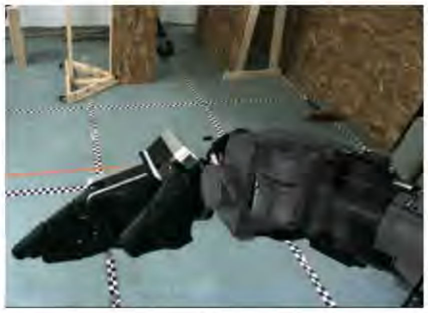

(d)

Figure 9: Kinematics calibration. Scatter plot of checkerboard corner reprojection error between forward kinematics based calculation of checkerboard corner positions and detections in camera images: (a) Before kinematics calibration (b) After calibration of joint angle offset biases and scale factors. (c) Example of reprojection of forward kinematics based arm configuration before calibration (d) after calibration

show a slope, illustrating the need to model actuator dynamics. The asymmetric response of joint actuation is also readily visible, with the same absolute value of applied current resulting in different measured torques, depending on sign of the applied current.

From the results of this section, we concluded a simple friction model based on the current joint velocity is not sufficient for effective friction compensation as the dynamic hydraulics effects must be taken into account. Friction forces have a tremendous impact on a pure torque based control policy; this led us to abandon attempts at model-based control for the DRC Trials as described in Section 5.4.

5.2 Control Modes

We chose early on to focus development on operator interactions and manipulation, while leveraging the basic capabilities provided by the Boston Dynamics API. These include behavioral control modes for maintaining balance in STAND and MANIPULATE modes, as well as basic WALK and STEP modes. Team ViGIR interfaces with these robot control modes through a custom RobotController program using an operator selectable set of control modes that determines what commands are accepted and processed by the robot control interface. While BDI supplied a USER mode that permitted complete control of all 28 joints in the robot, we chose not devote significant resources to developing whole-body controls prior to the DRC Trials.

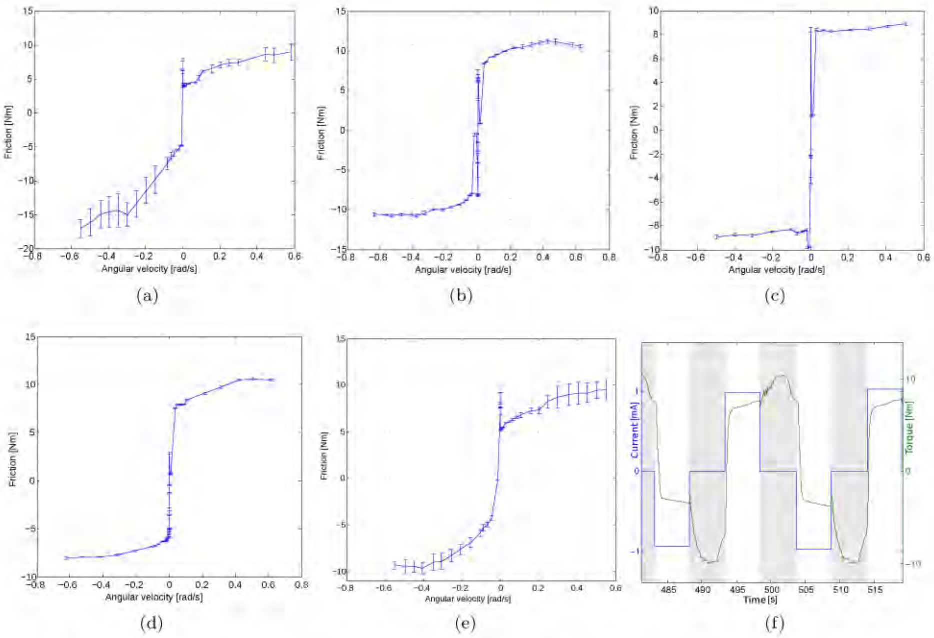


Figure 10: Friction analysis: (a)-(e): Experimentally determined Stribeck curves for left arm joints. (f) Valve current and torque for the left elbow joint. Only the parts of the plot with white background show torque sensor measurements with applied constant current. Dark background parts indicate motion of the joint using position control.

Thus, the USER mode was only used during the ladder task for the 2013 DRC Trials competition; although it was used more extensively during the VRC.

The WALK and STEP modes allow the onboard control system to send a sequence of footsteps defined in a world frame shared by the robot pose estimate. The RobotController accepts a footstep plan and sends the next four steps to be executed to the robot, which executes the steps using the robot's BDI-developed software to maintain balance. These control modes also provided the operator limited ability to command the upper body joints in the torso, arms, and head using joint commands.

The robot accepts joint position[21] or trajectory[22] vectors for joints grouped according to the appendage. The left/right arm/leg commands specify six joints each for 24 joints total, the torso specified three joints, and a separate head command accounted for the full 28 degrees of freedom for the robot. During the DRC Trials, we mainly depended on BDI behaviors for lower body control and balance; the leg appendage chain control was only used during the VRC and Trials ladder task. Coordinated motion of multiple appendages is obtained through the use of trajectories with a common time reference.

Commands to the robot hands are governed by the grasp controller, and are completely independent of the main robot control interface.

[21] http://docs.ros.org/api/sensor_msgs/html/msg/JointState.html
[22] http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectory.html

5.3 Joint Control

The robot API permits control of individual joints of the robot upper body in all behavioral modes, and lower body joints in USER mode. The joint control API permits control of position, velocity, and force based on a proportional-integral-derivative (PID)-based control law that included some feedfoward terms[23]. We predominantly used position-based PD control with a force term used for gravity compensation and a feed-forward term based on commanded velocity.

The software system has two basic modes with position and trajectory joint angle commands. As the hydraulic actuators are capable of significant forces and fast motions in response to step changes in commanded position, the operator interface position commands are converted to trajectory commands from current position to desired positions over a variable time interval. Upon receipt of a new trajectory command by the controller interface, the commanded trajectory is stitched to existing trajectory commands, and smoothed. The smoothing process recalculates the internal joint velocities at each desired joint position in the defined trajectory vector so that the trajectory can be represented as a sequence of cubic spline segments that match position and new velocity at each trajectory point.

During execution of the trajectory, the controller evaluates the cubic spline for the relevant section based on current time and publishes the desired position and velocity for each joint. As the main controller of the robot performs closed loop control with hard real-time guarantees, a dedicated standard Kernel Ubuntu machine is used for the RobotController, sending joint command updates at 250-300 Hz to the robot main controller. Once the trajectory end point in time is reached, the controller holds the current joint position with zero desired velocity. If robot joints are under control of an internal robot control mode, such as leg joints during stepping, the robot control interface tracks the current actual joint position so that mode transitions are bumpless[24].

In general, the per joint control response was excellent. Difficulties in manipulation were due to kinematics calibration errors and kinematic limitations of the arm design. The former difficulties were partially overcome by improved calibration as described previously (cf. section 4.3); The latter difficulties require additional improvements to the physical arm and the inverse reachability and inverse kinematics system in the next phase (cf. section 6.2).

Another difficulty that was not addressed in this phase was the dynamic coupling between arm and torso motions due to the massive arms. The Rigid Body Dynamics Library[25] (Felis, 2012) is used to calculate the forces required to support the robot in a quasi-static manner for gravity compensation, but full inverse dynamics was not attempted at this stage. Preliminary testing, as previously discussed (see Section 5.1), showed that the model based force calculations were significantly undervalued due to model inaccuracies and the high friction inherent in the hydraulic actuators. In lieu of model-based inverse dynamics, the controls focused on sending smooth arm trajectories with relatively slow motions to minimize the impact of dynamic coupling, and used relatively stiff PD control on the torso joints to minimize perturbations.

5.4 Whole Body Control

In the BDI USER control mode all joints can be controlled by the operator. Using this cabability for whole body control (WBC) was identified as a promising approach for improving manipulation performance preceding the VRC competition. Using established methods (Sentis, 2007) as a foundation, a whole body force control framework based on the RBDL dynamics library (as already used for gravity compensation,

[23]The exact form of the control equation and parameters is proprietary to Boston Dynamics, and is governed by non-disclosure agreements.

[24]'Bumpless' is a term of art in the industrial control community used to denote a continuous transition of control output between automatic and manual control modes (c.f. http://www.mathworks.com/help/simulink/examples/bumpless-control-transfer-between-manual-and-pid-control.html).

[25]http://rbdl.bitbucket.org/

cf. section 5.3) was developed. Using this approach, arbitrary operational points (any points on the robot links) can be controlled in task space. Using a hierarchy, tasks are projected into the joint space Jacobian null space of higher priority tasks, guaranteeing that higher priority tasks are not violated.

The pelvis of the robot was modeled as an unactuated 6 DOF free floating base. Balancing was obtained by controlling the position of the robot center-of-mass (COM) and by projecting every motion in the constrained space of the supporting contact limbs. Using this approach, the simulated robot was able to stand statically stable in dual stance and also perform stable single stance motions in simulation, as shown in Fig. 11a.

To support manipulation tasks and to achieve compliance while being in contact with obstacles, a hybrid position-force policy similar to the one described in (Khatib, 1987) was used. Fig. 11b shows an example of a hybrid control application, with the robot pushing a cinderblock across a table.

While showing very good performance in simulation, the developed WBC system relies on a accurate dynamics model of the robot which was available for simulation in the drcsim simulator [26], but not for the real Atlas system. This can degrade performance of the used operational space approach (Nakanishi et al., 2008). As time constraints made detailed system identification infeasible, this WBC approach was not used in the 2013 DRC Trials.

6 Motion Planning

Due to high latency and low available bandwidth, approaches that require constant operator input and supervision for motion generation are both slow and potentially unsafe. For this reason, automated planning is employed onboard the robot system, allowing the planner to leverage the availability of all sensor data with no latency. Ideally, the operator just has to specify a goal pose, with the planning system generating a viable and collision free plan to that pose. Separate planning systems have been developed for locomotion as well as manipulation.

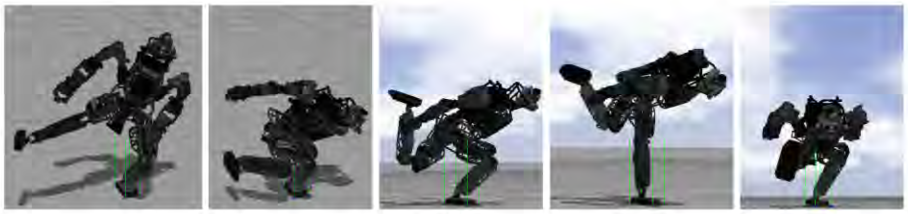
6.1 Footstep Planning

As described in Section 5.2 the Boston Dynamics API provides the capability for the robot to follow footsteps. Given a sequence of footsteps it generates and executes smooth trajectories for each foot placement which allows to decouple planning and execution level, also known as contacts-before-motion planning. For this purpose our approach is based on the already existing *footstep_planner* ROS package[27] which expands a graph and applies an Anytime Repairing A* (ARA*) search algorithm to obtain a sequence of footsteps (Hornung et al., 2012). For 3D planning on rough terrain, a suitable 3D world model, as well as suitable states, actions, transitions, and cost functions must be defined.

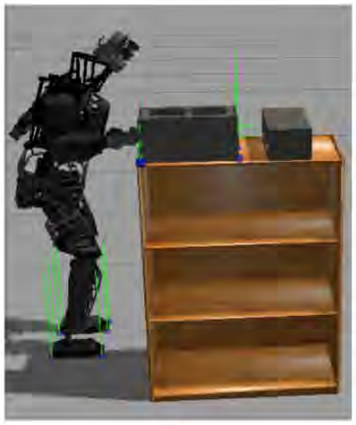
6.1.1 3D World Model

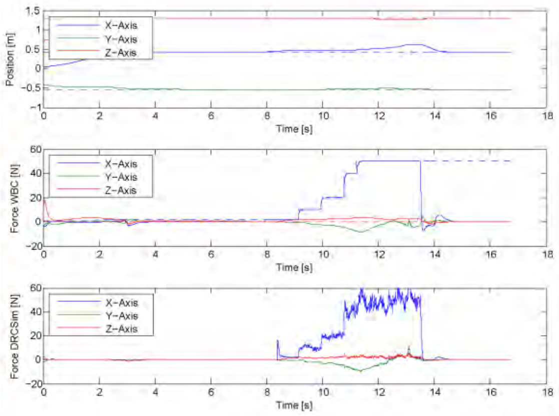
For 3D footstep planning a suitable world model is required which provides all data in an efficient way. The first step is pre-filtering the LIDAR point cloud to reduce noise in LIDAR data. Afterwards surface normal estimation[28] based on Principle Component Analysis (PCA) is applied (see Fig. 12a). The resulting normals are used to determine foot attitude for each step. A height map (see Fig. 12b) is generated which allows determining the foot height for each step and estimation of the ground contact percentage (see subsection 6.1.5 for further details).

[26]http://gazebosim.org/wiki/DRC
[27]`http://wiki.ros.org/footstep_planner`
[28]For more details see `http://pointclouds.org/documentation/tutorials/normal_estimation.php`



(a)





(b)

(c)

Figure 11: Whole Body Control: (a) Single leg stance motions (b) Atlas pushing cinderblock using hybrid position/force control. Along the X axis, a force control law is used, while the Y and Z axes are governed by position control (c) Position/force plot for the right hand. Solid lines are the sensed values; dashed lines are the commanded values. The upper plot shows the hand position, the middle plot the forces estimated by the WBC framework and the lower plot shows the forces provided by the simulated force torque sensors of drcsim. Starting at approximately 8 seconds, the robot begins pushing the block with stepwise increasing force, overcoming friction when applying over 50N. At the 13 second mark, the pushed block is stopped by the other block behind with no uncontrolled forces acting on the robot system.

6.1.2 States, Actions and Transitions

The foot state s is defined as global position and attitude in 3D space by $\mathbf{s} = (x, y, z, \phi, \psi, \theta)$. Here, ϕ, ψ and θ are the roll, pitch and yaw angle of the foot. The z, ϕ and ψ values are constrained by the underlying terrain, so all actions a can be given as a displacement vector $a = (\Delta x, \Delta y, \Delta\theta)$ relative to the stance foot s_0. All available actions are defined in the set of footstep primitives denoted as A. The original approach uses a manually defined set A, but in rough terrain scenarios the foot placement has to be flexible for the robot to be able to step over unstructured debris on the floor. Atlas is a human-sized robot having a large variety of possible footstep primitives, therefore it is not feasible to define such a set A manually. For this reason we define a reachability region $\mathcal{R}$ by a polygon next to the stance foot s_0 from which the footstep primitive set A is discretely sampled. Finally we define all possible transitions (s_0, s) by a function $s = t(s_0, a), a \in A$.

6.1.3 Cost Functions

In the original approach (Hornung et al., 2012), half steps are used for evaluation of cost functions. We found this representation to be insufficient for modeling cost functions of human-sized robots. Ignoring the

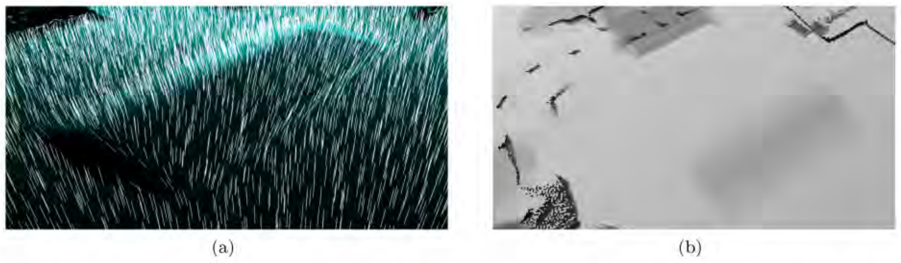


Figure 12: Most important parts of 3D world model: Normal estimation (a) and height map (b) of a ramp

source of the swing foot may lead to non-smooth footstep sequences, causing a fall. For this reason, in this work all cost are evaluated for full steps with s and s' denoting the source and target configuration for the same foot respectively.

The footstep planner has to minimize multiple, possibly competing costs e.g. shortest path and minimal fall risk. For this reason a hierarchical cost function was designed, modeling each criterion in a different layer. This approach was realized by using the Decorator Pattern (Gamma et al., 1994), enabling the flexible composition of cost functions. Let $c_i(s, s', c_{i+1}(s, s', ...))$ denote the cost function of the i-th layer given the result of the next layer's cost function. This recursion is stopped by the last layer which just evaluates $c_n(s, s')$. Each layer receives the result of the next layers and may reuse this value for internal purposes. In most cases cost functions are designed to just accumulate cost from the next layer, thus in common cases the resulting total cost may be illustrated as a weighted sum $c(s, s') = \sum_{i=1}^{n} w_i c_i(s, s')$. Using the 3D world model, the planner is able to generate sequences of footstep plans over rough terrain. Figure 13a shows a solution for the pitch ramp in which the soles of the feet are well aligned to the underlying 3D world model.

6.1.4 Risk Measurement

Using the reachability region $\mathcal{R}$ implies the risk of including bad footstep placements that lead to falls. For this reason a quantity denoted *risk* is required additionally to the cost. We are using cost and risk to distinguish between effort and feasibility of a step. Merging them in a single value would not prevent the planner from using bad steps in a plan when only a small number of step options are available. Analogous to evaluation of cost, each cost function computes the risk simultaneously and forwards it to the next layer. Feasibility of a step is finally determined based on the accumulated risk.

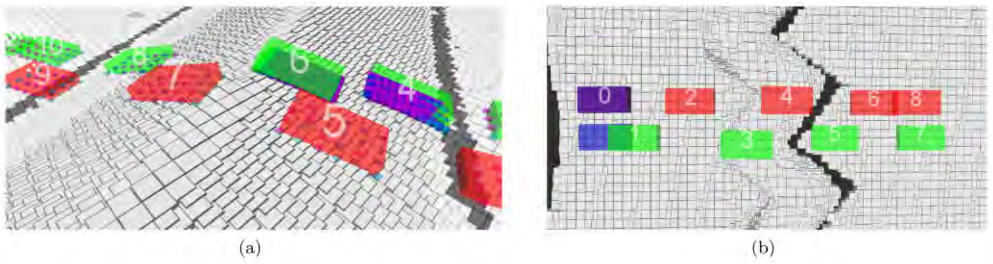


Figure 13: Example solutions for 3D planning in rough terrain: (a) Solution for walking over the pitch ramp showing footsteps aligned with underlying geometry. (b) Plan for traversing over the chevron hurdle with some footsteps on the hurdle having less than full ground contact.

6.1.5 Ground Contact Estimation

The Atlas robot is not capable of stepping over a cinder block sized obstacle, so this task can be solved only by stepping on top of such obstacles. For ground contact estimation, edge detection based on grid maps generated from point cloud data was implemented first. While performing as expected, occupancy grid maps have the drawback of wasting space by encircling each obstacle with non-traversable cells. As a result, collision checks with occupancy grid maps are too strict because they enforce placing each foot avoiding obstacles completely. This results in longer paths and the ARA*-planner takes longer computational time trying to minimize this path length. The resulting plan in figure 14a is obviously an ineffective solution to travel across the pitch ramp. An alternative approach therefore estimates the percentage of ground contact for a given foot configuration which allows usage of foot configurations with less than 100% ground contact. The planner is thus capable of planning for overhanging steps while keeping them collision free without the usage of discretized occupancy grid maps, improving the result on the pitch ramp significantly (see figure 14b). In figure 13b another example is illustrated, where the steps 3 and 4 are placed on cinder blocks with portions of the feet overhanging the blocks, enabling to generate a straight footstep plan over them.

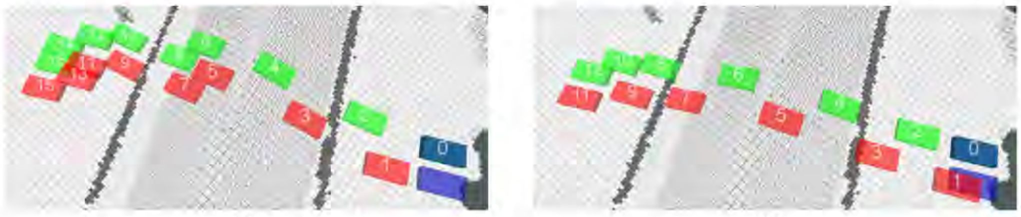

(a) (b)

Figure 14: Comparison of different collision check approaches: (a) Occupancy Grid Map (b) Ground Contact Estimation

6.2 Manipulation

For manipulation, motions need to be generated to move manipulators into desired configurations for grasping or other tasks. As it can reduce operator workload considerably, a desirable capability is automated collision avoidance, both with regards to self-collisions of the robot (e.g. arm coming in contact with torso) and collision with regards to collisions with environment. When performing manipulation in contact with the environment, planned motions must not lead to high internal forces acting on the robot. As described in section 5.4, force control was not a viable approach during the DRC Trials due to the limited time for proper system identification; for this reason, contact tasks were performed using position control with gravity compensation, making collision free and precise kinematic planning indispensable. As high latency limits the usefulness of otherwise promising approaches for teleoperation of end effectors (Leeper et al., 2013), we primarily employ planning to goal configurations. The Atlas arm kinematics proved to be challenging for manipulation, as the joint limits and composition of rotational DOFs in the arms result in limited reachable workspace. An important part of planning is selecting a pose of the robot pelvis relative to objects of interest in a way that allows the end effector to reach the object. For this, the Simox[29] open source inverse reachability approach described in (Vahrenkamp et al., 2013) has been integrated with our system.

We developed a custom manipulation planning package based on the MoveIt! (Chitta et al., 2012) framework available for ROS by using the low level MoveIt! API to provide additional specialized functionality required for the Atlas robot and DRC tasks. The system enables planning to goal joint configurations and to goal end effector poses. Two planning modes are available: The default mode is unconstrained planning, with joints free to move to reach the goal. Optionally, motion can be constrained to follow a Cartesian path between the start and goal end effector pose. In this case, waypoints are generated based on linear interpolation between

[29]http://simox.sourceforge.net/

start and goal position and orientations for waypoints are generated using spherical linear interpolation (Slerp) (Shoemake, 1985) between start and goal end effector quaternions. More complex constrained motions such as circular motion for turning a valve are generated by concatenating multiple short linearly interpolated Cartesian paths as shown in Fig.15.

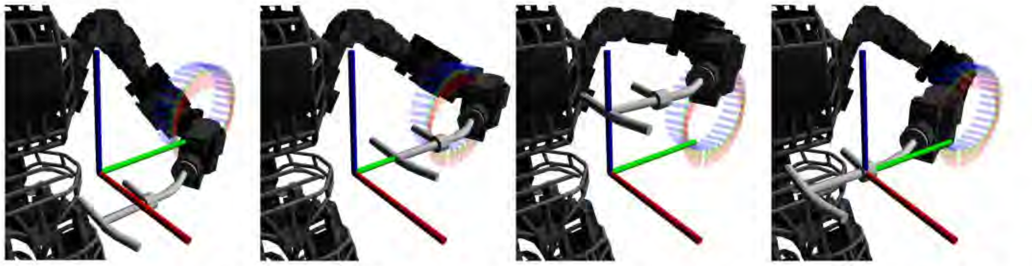


Figure 15: Clockwise circular path. The Hook rotates around the X axis(red) while the interpolated poses are shown for the last joint of the arm.

For obstacle avoidance, a region of interest of the worldmodel octomap as described in section 4.1 is used. As contact with the environment is required in many of the DRC tasks, collision checking between end effectors and the environment can optionally be switched off. For example, collision avoidance is needed to safely bring the robot hand into a position to pick up the drill. In order to grasp the drill, collisions between the palm and fingers of the hand and the drill handle must be allowed. With the challenging outdoor scenarios of the DRC, noise in sensor data that leads to geometric artifacts, preventing successful planning due to spurious collisions cannot be ruled out completely. To cope with such situations, collision checking with the environment model can also be disabled for the complete robot geometry; in this case the ghost robot changes color to warn the operator. Motion planning can be performed using either 6 DOF with the arms only, or by including the torso joints and using up to 9 DOF. As the 9DOF planning mode tends to result in higher control error or oscillation in some joint configurations,the operator can lock a selection of torso joints to restrict the planning space.

7 Grasping

For tasks requiring grasping operations, we use human-in-the-loop grasping and rely on the operator input to make crucial decisions. This strategy was chosen since for most objects, there are only one or two ways in which the object is to be grasped in order to complete a specific task. Thus, task specific grasps can be generated off-line and stored for future use, therefore removing the necessity for a robust online grasp generator. Each of the pre-generated grasps are associated and linked to a particular 3D geometry model (template) that the user can load in order to execute the stored grasps. After completion, the user can visually inspect the resulting grasp to ensure that the grasp executed correctly before proceeding with the rest of the task.

7.1 Template Grasps

All of the objects in the DRC Trials competition were well defined; therefore, we chose to create a 3D geometry template for all relevant objects, matching sizes and shapes. Each template contains information related to the object such as mass, center of gravity, and nominal orientation. Grasps are created off-line using the GraspIt! toolkit (Miller and Allen, 2004), taking into account parameters such as hand type, task, object size and relative location of the object in the environment. Once a reliable grasp is created, it is given an index number and the relevant data stored with respect to the template center of mass. The recorded data includes grasp and pre-grasp positions as well as a desired relative pelvis pose for the robot to maximize

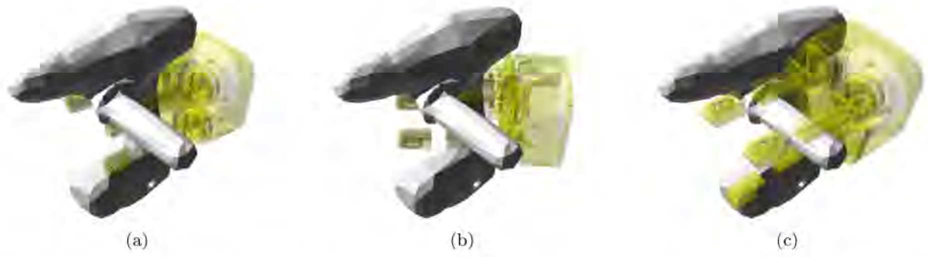


Figure 16: Precomputed Grasps visualized relative to drill template: (a) Front grasp. (b) 45 degrees grasp. (c) Handle grasp.

reachability. The pre-grasp location is typically created by moving the position of the hand approximately 10 cm away from the object in the direction normal to the palm. By creating a set of grasps offline, the cognitive burden on the operator is reduced because the number and types of choices are reduced.

During the task, the operator selects and loads the appropriate template into the main 3D view window. The operator aligns the template with perception data in the 3D view (see Section 4.2). Upon inserting the template, the associated grasp information is loaded and the operator is able to select and view all of the stored grasps. Stored grasps are visualized by showing a translucent hand overlaid on the template (Fig. **??**) to inform the user which grasp is currently selected.

7.2 Grasp controller

After placing the template and selecting the grasp, the grasp can then be tested for reachability using the ghost robot (Fig. 17a). If the grasp is reachable, the IK solution is displayed (Fig. 17b). If no solution is found, the ghost robot remains unchanged in its previous state. The user can then manipulate the robot pose, grasp orientation, template location, or torso constraints until the desired grasp is achievable. Once a feasible configuration is found, the user can send a footstep plan to move the real robot to the ghost robot pose. (Fig. 17c).

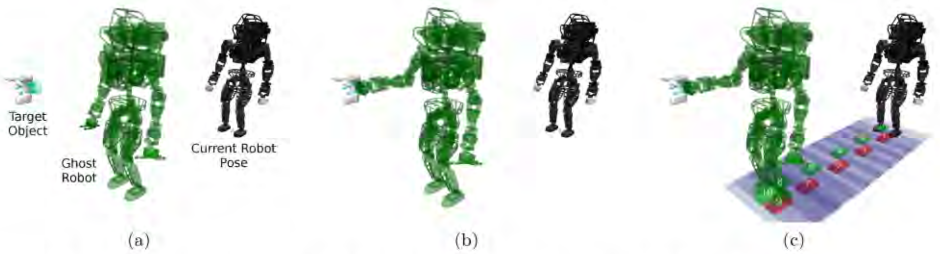


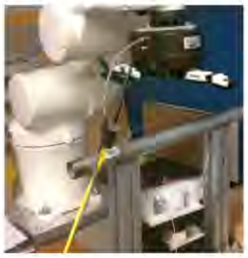
Figure 17: (a) Testing reachability, (b) Reachable position found and (c) Footstep plan to ghost pose.

Once the robot is correctly positioned with respect to the object, the user can then send the grasp to the grasp controller. The grasp controller then transitions through a series of states from the initial approach to the final grasp state. The grasp controller first enters the "approaching" state where it plans a collision free path from the current joint configuration to the IK solution for the pre-grasp position using the motion planner described in Section 6.2. Once the pre-grasp position is reached within a margin of error, the controller transitions into the "surrounding" state, moving the hand into the final grasp position near the object. When the final grasp position is reached, the controller switches to the "closing" state, closing the fingers around the object so that all fingers make contact simultaneously. When the grasp is completed, the grasp controller switches to the "monitoring" state where it maintains the grasp and signals the user that the grasp is completed. If at any point during the process an error is detected, the controller will make the appropriate corrections and state transitions in order to complete the grasping process. Additionally, the user has the ability to abort the grasping process if there is a problem that cannot be handled automatically by the grasping controller. If this occurs, adjustments to the grasp must be manually corrected by the operator until the final desired grasp is achieved. This primarily was an issue with the wall task, since correct placement of the hand was crucial in order to operate the drill.

After a grasp is executed, the user can visually inspect the grasp to make sure that it matches the desired grasp. Since the final grasp may not match the initially planned grasp due to small movements of the object while grasping or slippage of the object after grasping, the user can modify the planned grasp to match the actual one by using a "stitch template" feature. Because variations between the planned and final grasp can affect manipulation quality, the object pose with respect to the hand must be accurately estimated. By rotating the template relative to the hand and positioning it to match the actual object orientation, the modified template transform can be used to update the actual grasp location. All future manipulations of the template will use this new transform, thus "stitching" the object to the hand and aiding the user in manipulation. This very important for the wall cutting task, as the cutting bit pose is used for planning cutting operations.

7.3 Hardware validation

Prior to selection of either the iRobot or the Sandia hands for use during competition, the robustness of the grasps produced by each hand was evaluated. Each hand was placed on the end of a six degree of freedom robot with a sample object anchored to the table below using a steel cable and force sensor (see Fig. 18a and Fig. 18b).


(a)

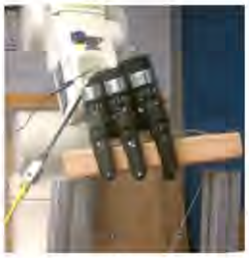

(b)

Figure 18: Testing setup with object anchored to table by a cable and force sensor. (a) iRobot hand (b) Sandia hand

The end effector was moved into a human selected ideal grasp location above the object. A series of grasp tests were then performed where the hand was commanded to the ideal grasp position with an increasing incremental offset to mimic position errors. After grasping, the arm was moved upward slowly until the object was dropped and the maximum force exerted prior to the grasp slipping was recorded. After analyzing the results, it was found that the iRobot hand was nearly twice as strong as the Sandia hand and was less likely to break. Despite the downside of reduced dexterity, we selected the iRobot hand because of its strength, reliability, and robustness.

During practice with the iRobot hands on the wall task, we noticed a significant amount of vibration while using the drill. It was determined that this was due to the underactuated nature of the hand and the lack of having distal phalanges in contact with the drill (see Fig. 19a). Based on a static analysis of the fingers in contact with the drill, it was determined that for a fixed finger cable excursion length ΔL, the contact forces on the object would be reduced if the distal joint angle $\Delta\theta_2$ is increased and the proximal joint angle $\Delta\theta_1$ is decreased (see Fig. 19b), resulting in the loss of grip force and the object slipping.

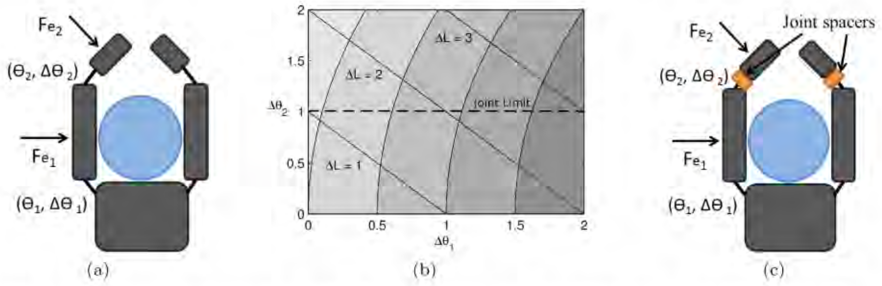


Figure 19: (a) Grasping configuration with distal phalanges not making contact with object ($F_{e_2} = 0$). (b) Underactuation force response for given grasp configuration where darker region is higher contact force F_{e_1} with object. Increasing cable excursion length ΔL increases overall contact force given that there is a joint limit for joint two. (c) Image of the finger spacers added to create joint limit to increase grip strength.

This relaxing of the grip could be overcome by increasing the cable excursion length, but would result in permanent joint damage as the required force in the grip to restrain the drill would cause the distal joint to flex beyond the safe limit. To increase grip force in the proximal phalanges without damaging the distal joints, we added joint spacers which introduced a joint limit on the distal link (see Fig. 19c). With the spacers added, the distal joint would increase until the limit was reached, resulting in further cable length changes increasing $\Delta\theta_1$ and thus increasing the grip force F_{e_1} on the object without risking damage to the finger joints.

8 DRC Trials Results

In this section, we describe the tasks of the DRC Trials, how Team ViGIR approached each task, and the results during the Trials. The approaches were based on lessons learned during testing with the Atlas robot, and an assessment of current system limitations. Supplemental video recordings of task performance for all tasks is available online[30]; we recommend viewing these after reviewing the descriptions below. After descriptions of individual tasks, we discuss lessons learned based on a timeline schematic that provides further insight into performance at the Trials.

8.1 Overview

The DRC Trials took place at Homestead-Miami Speedway on December 20-21, 2013. Teams competed in a total of 8 tasks, with only one attempt at each task allowed per team. The time limit for each task was 30 minutes. There was a maximum score of 4 points per task. Points could be scored by completing pre-defined subtasks. For instance, turning one of the three valves in the valve tasks was worth one point each. In case of robots getting stuck or falling, teams could call for an intervention, allowing restarting the robot at the start of the current subtask and incurring an automatic 5 minute penalty. If a team scored 3 points without a intervention in between, a fourth point was awarded. See (Krotkov, Eric, 2013) for a comprehensive overview of the DRC Trials rules. Team ViGIR performed five tasks (door, debris, hose, valve, wall) on the first day and two tasks (terrain, ladder) on the second. The driving task was not attempted by the team. Due to a

[30]http://www.youtube.com/playlist?list=PL5Ku_0Pgk5eKbi9dYWrNoX_j4t6tOr9d5

hardware issue, the robot had a major repair in the night before the first competition day, which limited our ability to perform kinematics calibration before competition start.

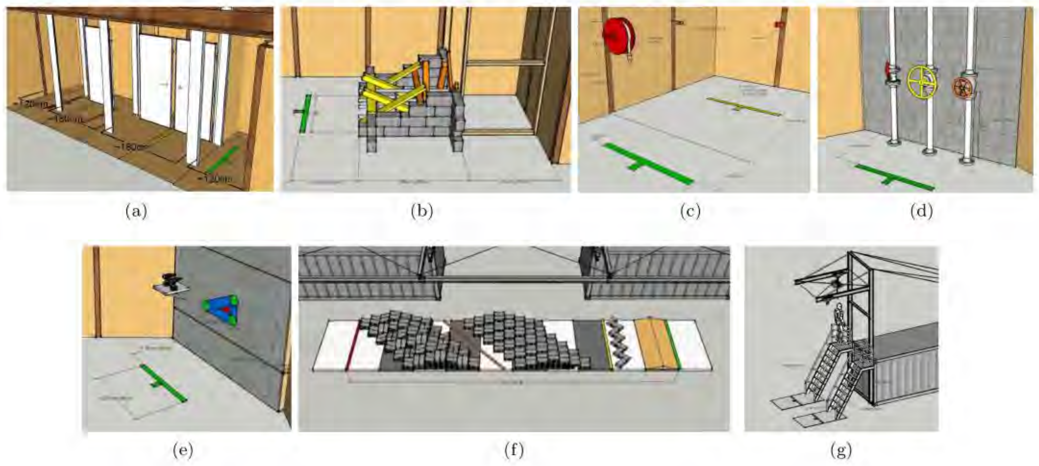


Figure 20: Schematics of Tasks attempted by Team ViGIR at the DRC Trials: (a) Door (b) Debris (c) Hose (d) Valve (e) Drill (f) Terrain (g) Ladder. All schematics provided by DARPA.

8.2 Team ViGIR performance at the Trials

8.2.1 Door

The Door task consisted of crossing through three different doors. The first door was a "push" door, the second was a "pull" door and the third was a "weighted pull" door (see Fig. 20a). A point was given for crossing each door. Since turning a door handle requires low dexterity no robotic hand was used. The robot was equipped with a hook on each hand which was then used to turn the handles and move the doors (see Fig. 21a).

To open the door, our standard proceedure is to estimate the door handle pose using an object template. Based on this template, footstep plan is computed to move the robot into a manipulation position. Once reached, the IK positions of the arm to turn the handle are validated using the ghost robot and once the operator is satisfied with them, the robot arms are moved. With the door opened, the next step is to calculate a footstep plan taking the robot from the current pose into a pose across the door automatically. During lab-based testing prior to the DRC Trials, we made many successful attempts to walk through the door using autonomous and manual planning. During the DRC Trials door task, automated planning failed as happened occassionally during testing as the doorway is very narrow and noise in sensor data can lead to it appearing blocked. The operator switched to semi-autonomous planning for this reason, selecting footstep plans manually. Sending a manually selected footstep plan, the robot touched the door frame with the right hook hand, leading to a fall of the robot. A similar failure occurred in a second attempt after an intervention was called; at this point, time ran out and no points were scored in this task.

8.2.2 Debris

The Debris task consisted of having to remove debris from a doorway, in order to be able to walk through it. The debris consisted of 10 pieces of balsa wood of different dimensions and an aluminium truss (see

Fig. 20b). A point was given for removing sets of 5 pieces of debris and the last point was given for crossing the open doorway. The robot was equipped with a hook and an extended iRobot hand (see Fig. 21b).

Using an object template, the intended robot location with respect to the first debris piece is specified and is used to obtain a footstep plan to send the robot to this pose. Pre-Trials testing showed that removing debris pieces one by one was time consuming and error prone, as accidentally dropping a debris piece could introduce long time delays. When the grasp on the first piece failed when trying to remove it and it fell back into the debris pile, the operator team changed strategy and attempted to pull the whole truss out of the way using the hook hand. Even though more than 5 pieces of debris were moved, they were not located completely outside the rectangle specified in the rules before time elapsed; no points were obtained in this task.

8.2.3 Hose

The Hose task consisted of having to take a hose and attach it onto a wye by screwing it to a threading coupling. The hose was mounted on a reel separated from the wye by several meters so the robot had to grasp the hose and walk with it towards the wye (see Fig. 20c). A point was given for walking past a defined line with the hose, a second point was given for touching the hose coupling to the wye and the third point was given for attaching the hose to the wye. The robot was equipped with an iRobot hand to grasp the hose and align it with the wye, and a hook to turn the threaded coupling (see Fig. 21c).

Using the hose template and a footstep plan, the robot was sent to a position where the hose was picked up using the iRobot hand. While holding the hose, a footstep plan was then calculated to send the robot into a position in front of the wye. We scored the first two points, but were unable to attach the hose to the wye. The hook was used to turn the coupling a couple of times. Given that attaching the hose requires high accuracy, small misalignments caused the threads not to engage so that the hose fell to the floor when released. The primary challenges were the limited field of view of the cameras, the lack of tactile feedback and the self-balancing MANIPULATE mode (see Section 5.2) that prevented the robot from applying a constant force between the hose and the wye. According to video analysis, Team ViGIR was the fastest Atlas team to score the second point in this task.

8.2.4 Valve

The Valve task consisted of opening three different valves. The first valve was a lever valve, the second was an 18-inch diameter rotary valve and the third was a 9-inch diameter rotary valve (see Fig. 20d). A point was given for closing the lever valve 90 degrees and the other points were given for closing the rotary valves 360 degrees. The manipulability behaviours described in Sec. 6.2 provided the capability of turning the valves in one single circular movement of the arm. Since using robotic hands would imply a series of grasping/release step actions, the robot was equipped with a hook for each hand (see Fig. 21d).

Using the valve object template and footstep plans, the robot was sent into a position in front of each valve. The lever valve was turned using linear cartesian motion, while the rotary valves were turned using the circular path planning capability as shown in Fig. 15. Team ViGIR scored four points in this task including one point per valve and the bonus point, which was awarded for completing all three valve tasks without intervention.

8.2.5 Wall

The Wall task consisted of a half inch thick drywall panel with a right triangle pattern traced on the front which must be cut out (see Fig. 20e). The robot was set up with the iRobot hand on the left arm and a hook hand on the right arm (see Fig. 21e). Finger spacers were added to all but the trigger finger to increase grip strength but allow full motion of the trigger finger (see section 7.3). This was done so as to maximize the force applied to the drill and to minimize the amount of vibrations in the drill while cutting laterally. The goal was to cut out the red triangle in the middle without cutting into any of the gray area surrounding the triangle. Cuts must be continuous from one green circle to the next. Two templates were created, one of the drill and the other of the triangle pattern to be cut out. The locations to stand relative to each template was calculated and stored prior to the trial. The Dewalt DCD980M2 drill was selected for the task for the reason that it would be easier to turn on and kinematically easier to perform the cutting operation within the field of view of the Multisense sensor. Since limited reachability of Atlas arm was already determined to be an issue, we opted to use the drill which would minimize this risk.

We scored zero points due to a slip of the grasp while performing the first vertical cut. Since the cut was being made downward, the resulting force was greater than the grip force could withstand, causing the hand to slip and lose grasp of the drill trigger. After the loss of the grasp on the trigger, we were unable to recover in time to complete the task. Loss of grip on the trigger could have been mitigated by starting the cut from the bottom and proceeding upward; however, selection of the other drill tool would have eliminated this issue and perhaps performed better as demonstrated by several of the other teams.

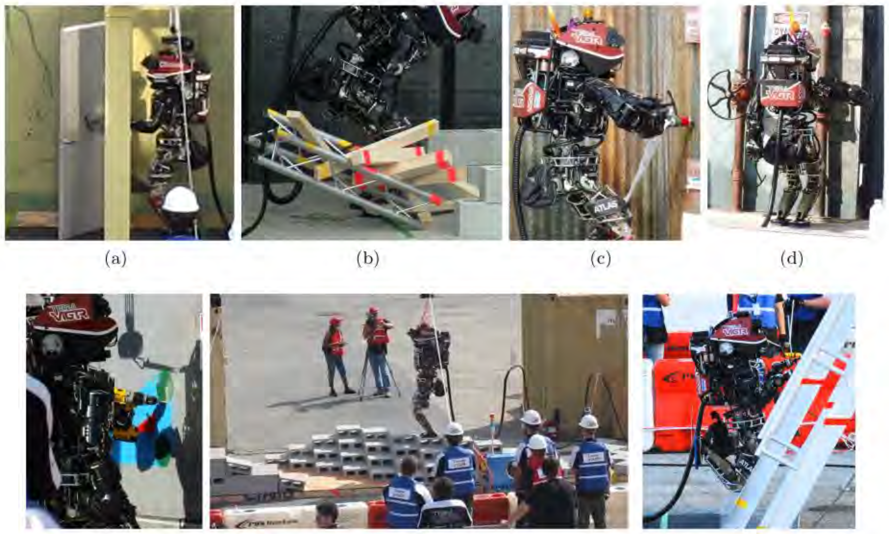

(a) (b) (c) (d) (e) (f) (g)

Figure 21: Participation in the DRC Trials: (a) Opening the first door (b) Pulling the truss out in the debris task (c) Attempting to connect the hose to the wye (d) Rotating the first valve (e) Using the drill (f) Stepping over cinderblocks (g) Robot on ladder with all support points above the floor.

8.2.6 Driving

The Driving task consisted of using the robot to drive a Polaris Ranger vehicle through a slalom course. During the Trials, the robot could be placed into the vehicle at the start, but had to drive the length of the course to receive a single point. Two additional points were earned by having the robot climb out of the vehicle, and walk across the finish line. Doing both parts successfully would have earned the team four points.

Given limited practice time, and the risk of hardware damage to the robot in this multi-contact situation, the team chose not to pursue the driving task and therefore scored zero points for this task.

8.2.7 Terrain

The Terrain task consisted of three parts with different requirements and difficulties (see Fig. 20f). The first part consisted of a pitch ramp and a chevron hurdle. The robot walked autonomously over the pitch ramp and adapted the foot attitude to the underlying ramp. The robot also autonomously planned and executed stepping over the chevron hurdle without any issues, and therefore scored one point.

The second part consisted of stairs with flat surfaces. The planner failed in this situation due to noise in 3D world model data. The flexibility of our UI allowed the operator to manually plan single steps up each block, and the robot successfully achieved the top step (see Fig. 21f). Unfortunately, during the initial step down the robot fell due to a weakness of the knee joint in this configuration. During our hurried second attempt after an intervention, the right robot foot caught a neighboring block on the step down and the robot fell a second time. In the end, we earned only one point from this task.

8.2.8 Ladder

The Ladder task consisted of having the robot ascend a ladder (see Fig. 20g). A point was given for having all the contact points above the first step, the second point over the fourth step and the last point with all contact points above the landing. Due to the high weight of the robot, it was equipped with a hook on each hand to be able to support itself while stepping (see Fig. 21g).

Using the ladder template, the robot location relative to the first step was previously calculated and used to obtain a footstep plan to send the robot into this position. The first point was obtained using the hooks to support the robot from the fourth step, then bending the knees in USER mode such that the knees rested on the first step, and thus had all the contact points above the first step. During a second try, the robot was commanded to do a walking step with a defined height. Although the movements of the robot looked promising, the robot fell while transitioning its weight to the upper foot.

8.3 Overall Trials Performance

Considering Fig. 22 and analyzing performance across all tasks, general observations can be made. The speed at which a task was performed was generally determined by the speed and proficiency of the primary OCS operator, who in turn relied on OCS and onboard systems. The secondary OCS operator took over the tasks of sensor data acquisition and template alignment, offloading work from the primary operator and accelerating operations due to parallelization between operators. The robot system was moving less than 50% of total task time, indicating that the bottleneck for task execution was on the operator side, with the robot waiting for commands most of the time. In only a few cases was the opposite true, with operators waiting for the robot to finish motion execution.

Analyzing task performance at the Trials, the following reasons for the operators being the bottleneck have been observed: The performance of robot state estimation, while sufficient for locomotion, was not reliable enough for performing automated grasping in the competition. Both internal (kinematics calibration) and external (pose estimation) state estimation errors contributed to the operator needing to close the end-effector positioning feedback loop manually, adjusting end-effector poses. A contributing factor was the previously mentioned robot repair in the night directly preceding the competition, which precluded proper calibration. As manipulation in contact was required for nearly all tasks and the team wanted to reduce the likelihood of failures and resulting interventions using position control, the speed of trajectory execution was reduced considerably compared to what the robot hardware is capable of.

The overall approach described in this paper proved to work well; however, the previously mentioned issues made task execution very slow. The task failures were due to insufficient operator training (door, debris), bad decisions about procedures (wall), or hardware faults (terrain), and were not fundamentally due to the

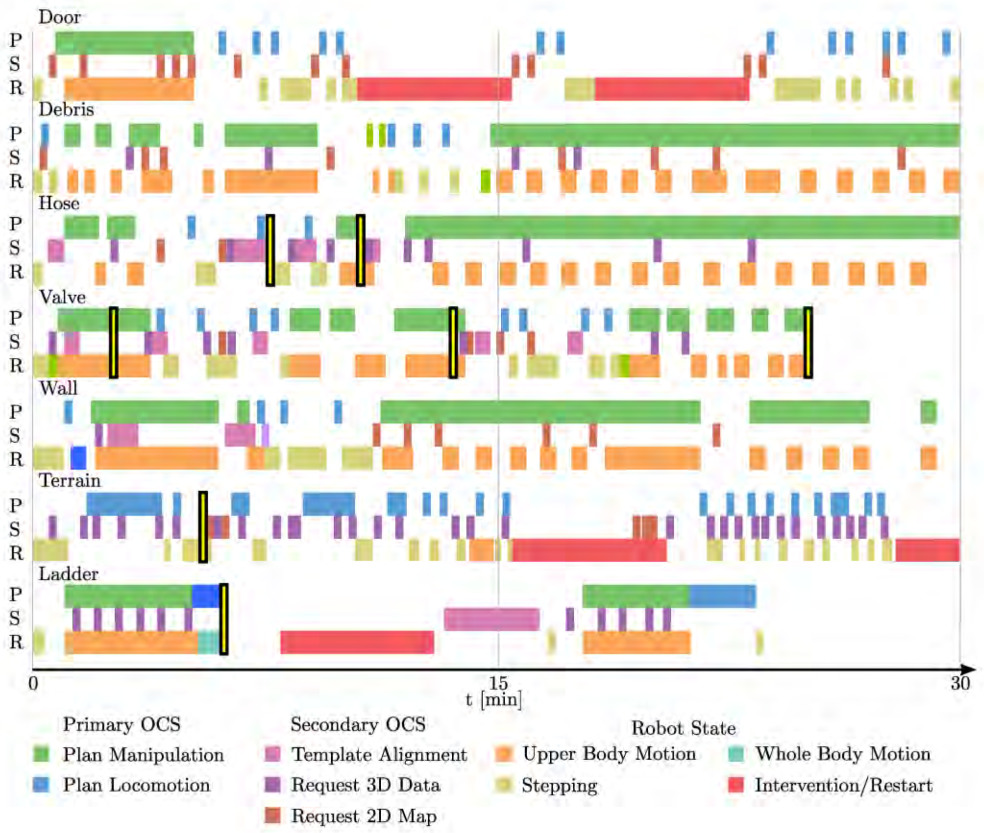


Figure 22: Overview schematic showing the time and task distribution among Primary OCS (P) , Secondary OCS (S) and robot state (R). Scoring events are marked using yellow bars with black border.

approach or software. In future work the following shortcomings noted in the Trials will be addressed, with the goal of increasing reliability and speed of task execution:

- Reliable and fast kinematics calibration is required for reliable autonomous manipulation.
- Drift compensation for robot pose estimation is required for locomotion as well as manipulation.
- Friction compensation will be investigated to enable low gain position control with gravity compensation or full force control.
- Automated pose estimation and grasp planning for objects of interest indicated by the operator(s)
- Whole body planning to leverage the high number of DOF and extending the reachability of the system.
- Whole body control to address the driving and ladder tasks.

9 General Lessons Learned

Beyond the technical discussion of performance at the Trials in the previous section, a number of lessons were learned over the course of the project. These include both successes that we can build upon, and failures that taught important lessons as we move forward to the next phase.

Team Coordination is the most important issue to be tackled in an extremely time constrained and complex project such as DRC participation. With Team ViGIR members being spread over more than 9000km, setting up tools and procedures was crucial. Using the Redmine project management system for issue tracking and documentation and coordinating via teleconferences weekly and additionally whenever needed, the team successfully tackled this challenge.

Complex Software Projects require all members of the involved team(s) to have easy access to software in order to be able to build and test software as well as follow conventions during development. Within Team ViGIR, great care was taken to guarantee this at all times by standardizing OS and ROS versions used; the installation, update and compilation processes were simplified using scripts. The power of the distributed version control system *Git* was utilized for our own software development as well as to quickly fork and extend upon existing open source software and use feature branch based development.

Simulation is an important tool to be able to validate and test complex robotics software. With an international team, this becomes even more crucial, as access to the real robot system is limited. The robot API available for drcsim and for the real Atlas robot diverged due to rapid development of the latter when the real robot became available. This meant that drcsim was only of limited use preceding the DRC Trials, a situation that caused development delays for the team.

Operator Training is crucial to overall system performance. Due to time and resource constraints as well as required expertise, the operator team was made up of core team members involved in onboard and OCS software development. This restricted the amount of time that could be dedicated to training. Given the infrastructure that is now in place, we plan to put an emphasis on operator training and practice in the next phase.

Multiple Operators Approach worked great as a "Wizard of Oz" speech interface, in which the main operator says, "May I have a point cloud?" and secondary operator does it. We plan to extend these functionalities to include new ways of communication and collaboration between operators, such as allowing the operators to mark and annotate point clouds. We intend to significantly increase the amount training and practice time where the operators function in defined roles prior to competition.

Interaction for Manipulation is a major challenge, as 6 DOF poses have to be specified in a fast and intuitive way. Using keyboard and mouse as a baseline approach, we will evaluate advanced methods for interactions as future work.

Understanding Complex 3D Structures was hard, especially because it was difficult to judge depth. We plan to evaluate the use of displays that provides higher levels of display fidelity (objective level of sensory stimuli (Slater, 2003)), such as a head-mounted display with stereoscopy and head-tracking, and how it can be integrated into the workflow of completing tasks with multiple operators.

Operator Control Station Layout is crucial for enabling interaction of the operator with the robot system. For the primary operator, our four screen setup worked well during competition; however, this setup could not be easily replicated on other machines for operator training.

User Interface design must be simple and intuitive to facilitate operator training. During development, numerous engineering widgets were created for testing and specialized controls. While the ability to rapidly prototype interfaces using *rqt* was crucial for enabling us to complete development, the resulting interface was overly cluttered and did not satisfy our design criteria. A major focus during the next phase is on streamlining the interfaces, and integrating control functions.

Multiple Perspectives are crucial to success. The limited FOV of the robot head and lack of a yaw joint on the neck limited perception. We found the cameras in the Sandia hands useful for providing another point of view during testing, but chose to use different configurations at competition for several reasons. We intend to revisit additional sensing leading up to the DRC Finals.

10 Conclusion

In this work, we present an overview of Team ViGIR's approach for solving complex rescue tasks using teaming between a robot system and human operators. The DRC scenarios are especially challenging because of severe bandwidth constraints and the use of complex human tools by a humanoid robot system. This paper provides an overview of many (but not all) of the important components in our onboard and operator control system. We describe our performance in the DRC Trials competition on a per task basis and related video data is linked. Lessons learned are discussed both from a single performance issue and from a "bigger picture" human factors/project management view point.

We plan on releasing all Team ViGIR software not bound by non-disclosure agreements as open source after the DRC Finals to facilitate knowledge transfer and comparison with work by other researchers.

11 Acknowledgments

This project was supported by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Lab (AFRL) contract FA8750-12-C-0337.

Team ViGIR would like to thank DARPA and its support staff for a well run Trials, the Open Source Robotics Foundation (OSRF) for their simulation support, and Boston Dynamics, Inc. for their support with the Atlas robot. The team would also like to thank the contributors and maintainers of the Robot Operating System (ROS), MoveIt!, GraspIt!, Octomap, rviz, rqt_gui, calibration, footstep_planner, Simox and Rigid Body Dynamics (RBDL) libraries.

In addition, the co-authors would like to thank our other team members who provided support during development: TU Darmstadt - Karen Petersen, Johannes Meyer, Dorian Scholz, Philipp Schillinger, Jochen Mück, Michael Stahr, Omid Pahlevan-Sharif, Mike Smyk, Sebastian Gauert; Virginia Tech - Tania Hamid, Lindsay Blassic, Jacob Shepphard, Alex Little, Tony Angel; Oregon State - Jordan Meader, Zhifei Zhang, TORC Robotics - David Anderson, Adam Bowling, Janette Brown, Kevin Collins, Jesse Hurdus, Andrew Lycas, Chris Sammet, and Ben Waxler.

References

Alunni, N., Phillips-Grafftin, C., Suay, H. B., Lofaro, D., Berenson, D., Chernova, S., Lindeman, R. W., and Oh, P. (2013). Toward a user-guided manipulation framework for high-dof robots with limited communication. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pages 1–6. IEEE.

Blodow, N., Goron, L. C., Marton, Z.-C., Pangercic, D., Ruhr, T., Tenorth, M., and Beetz, M. (2011).

Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4263–4270. IEEE.

Bruemmer, D., Dudenhoeffer, D., and Marble, J. (2002). Dynamic autonomy for urban search and rescue. In *In Proc. 2002 AAAI Mobile Robot Workshop*, Edmonton, Cananda.

Chitta, S., Sucan, I., and Cousins, S. (2012). MoveIt! *IEEE Robotics Automation Magazine*, 19(1):18–19.

Dai, H., Valenzuela, A., and Tedrake, R. (2014). Whole-body motion planning with simple dynamics and full kinematics. *Under review*.

Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization. *Under review*.

Desai, M. and Yanco, H. (2005). Blending human and robot inputs for sliding scale autonomy. In *Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop on*, pages 537–542.

Fallon, M., Kuindersma, S., Karumanchi, S., Antone, M., Schneider, T., Dai, H., Perez D'Arpino, C., Deits, R., DiCicco, M., Fourie, D., et al. (2014a). An architecture for online affordance-based perception and whole-body planning. *Under review*.

Fallon, M. F., Antone, M., Roy, N., and Teller, S. (2014b). Drift-free humanoid state estimation fusing kinematic, inertial and lidar sensing. *Under review*.

Felis, M. (2012). RBDL - the rigid body dynamics library. `http://rbdl.bitbucket.org`. Accessed: 2014-03-07.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Goodrich, M. A. and Schultz, A. C. (2007). Human-Robot Interaction: A survey. *Found. Trends Hum.-Comput. Interact.*, 1(3):203–275.

Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V. (2011). Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 858–865. IEEE.

Hornung, A., Dornbush, A., Likhachev, M., and Bennewitz, M. (2012). Anytime search-based footstep planning with suboptimality bounds. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 674–679. IEEE.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, pages 1–18.

Huang, H.-M., Messina, E., and Albus, J. (2007). Autonomy levels for unmanned systems (alfus) framework volume ii: Framework models version 1.0. NIST Special Publication 1011-II-1.0, NIST.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53.

Koch, M. D. (2010). Utilizing emergent web-based software tools as an effective method for increasing collaboration and knowledge sharing in collocated student design teams.

Kohlbrecher, S., Conner, D. C., Romay, A., Bacim, F., Bowman, D. A., and von Stryk, O. (2013). Overview of Team ViGIR's approach to the Virtual Robotics Challenge. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–2. IEEE.

Koolen, T., Smith, J., Thomas, G., Bertrand, S., Carff, J., Mertins, N., Stephen, D., Abeles, P., Englsberger, J., McCrory, S., van Egmond, J., Griffioen, M., Floyd, M., Kobus, S., Manor, N., Alsheikh, S., Duran, D., Bunch, L., Morphis, E., Colasanto, L., Ho Hoang, K.-L., Layton, B., Neuhaus, P., Johnson, M., and Pratt, J. (2014). Summary of team IHMC's Virtual Robotics Challenge entry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE.

Krasner, G. E., Pope, S. T., et al. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49.

Krotkov, Eric (2013). DRC Trials Rules. `http://www.theroboticschallenge.org/files/DRCTrialsRulesRelease7DISTAR22157.pdf`. Accessed: 2014-07-22.

Leeper, A., Hsiao, K., Ciocarlie, M., Sucan, I., and Salisbury, K. (2013). Methods for collision-free arm teleoperation in clutter using constraints from 3D sensor data. In *IEEE Intl. Conf. on Humanoid Robots*, Atlanta, GA.

Lischinsky, P., Canudas-de Wit, C., and Morel, G. (1999). Friction compensation for an industrial hydraulic robot. *Control Systems, IEEE*, 19(1):25–32.

Loeliger, J. and McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.".

Miller, A. and Allen, P. K. (2004). GraspIt!: A versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122.

Murphy, R. R. and Burke, J. L. (2005). Up from the rubble: Lessons learned about HRI from search and rescue. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 49, pages 437–441. SAGE Publications.

Nagatani, K., Kiribayashi, S., Okada, Y., Otake, K., Yoshida, K., Tadokoro, S., Nishimura, T., Yoshida, T., Koyanagi, E., Fukushima, M., et al. (2013). Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots. *Journal of Field Robotics*, 30(1):44–63.

Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757.

Odhner, L. U., Jentoft, L. P., Claffee, M. R., Corson, N., Tenzer, Y., Ma, R. R., Buehler, M., Kohout, R., Howe, R. D., and Dollar, A. M. (2013). A compliant, underactuated hand for robust manipulation. *International Journal of Robotics Research*.

Pradeep, V., Konolige, K., and Berger, E. (2014). Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *Experimental Robotics*, pages 211–225. Springer.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3.

Rosson, M. B. and Carroll, J. M. (2002). *Usability engineering: scenario-based development of human-computer interaction*. Morgan Kaufmann.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE.

Sandia National Laboratories (2014). Sandia hand. `http://www.sandia.gov/research/robotics/advanced_manipulation/Sandia_Hand.html`. Accessed: 2014-03-07.

Scholtz, J. (2003). Theory and evaluation of human robot interactions. In *HICSS '03 Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*.

Sentis, L. (2007). *Synthesis and Control of Whole-Body Behaviors in Humanoid Systems*. PhD thesis, Stanford University, Stanford, USA.

Shoemake, K. (1985). Animating rotation with quaternion curves. *ACM SIGGRAPH computer graphics*, 19(3):245–254.

Slater, M. (2003). A note on presence terminology. *Presence connect*, 3(3):1–5.

Stumpf, A., Kohlbrecher, S., Conner, D. C., and von Stryk, O. (2014). Supervised footstep planning for humanoid robots in rough terrain tasks using black box walking controller. *Under review*.

Tedrake, R., Fallon, M., Karumanchi, S., Kuindersma, S., Antone, M., Schneider, T., Howard, T., Walter, M., Dai, H., Deits, R., et al. (2014). A summary of team MIT's approach to the Virtual Robotics Challenge. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Tuley, J., Vandapel, N., and Hebert, M. (2005). Analysis and removal of artifacts in 3-d ladar data. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2203–2210. IEEE.

Vahrenkamp, N., Asfour, T., and Dillmann, R. (2013). Robot placement based on reachability inversion. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1970–1975. IEEE.

Van Dam, A. (1997). Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67.

Yanco, H. A. (2004). Classifying human-robot interaction: An updated taxonomy. In *Proc IEEE SMC*, pages 2841–2846.

B. SYSTEM HARDWARE MODIFICATIONS

Hand Hardware and Robotiq Modifications

In an attempt to gain better vantage points for the various manipulation tasks, we affixed one small, low-resolution camera to the palm of each Robotiq hand facing outward from the middle of the paired fingers. The cameras offered views that proved useful for object contact verification, driving obstacle avoidance, and other task confirmations, but they, and the devices that supported them, were not as robust as was necessary for the robot's stature or for the tasks attempted. As the DRC finals drew close, hardware maintenance issues and low part availability rendered these cameras all but useless; they can be seen in the above pictures of the DRC Finals, but they are inoperable at this point.

In addition to palm cameras, we attempted to outfit the Robotiq hands with sets of tactile sensors to predict executed grasp quality and to provide operators with colored contact information in the OCS. Initially, we had planned to implement a machine learning algorithm that might predict, in real-time, the robustness of a grasp based on the number of finger contacts and the strength of each contact. The result could then be displayed to an operator or passed along to a behavior, which may decide to continue the task at hand or to replan and re-execute. Although much effort was put into these sensors and the machine-learning processing, the tactile hardware proved even less robust than the camera apparatus and necessitated removal prior to the DRC Finals.

The last duty attempted by the hand electronics was determining whether or not the team had successfully engaged the cutting apparatus for the Drill Task. For this, we made use of small USB microphones planted on the side of each hand and monitored their average volume levels after initiating the Drill Task. During testing we found that the cutting tool produced a loud enough response when activated that we could readily detect it via the microphone. The microphone system was fully implemented by the time of the DRC Finals, but was also not used.

All of the aforementioned electronics were powered by a 24V line split off of each of Florian's arms and relied on Ethernet for communication. The 24V line was run through a variable step-down DC-DC voltage converter and fed into a Raspberry Pi 1 B+ and a small three-port Ethernet switch. Custom cases were designed and printed for each component (camera, raspberry pi, Ethernet switch, and DC-DC converter) to safeguard them from physical shock and electrical conductors. The raspberry pi ran the Debian-based Raspbian operating system and was outfit with ROS indigo.

The raspberry pi functioned as a command and control center and information relay, handling camera/tactile control/resetting and transmitting the captured information through the ROS framework to the proper listeners. The microphone and takktile sensors made use of the raspberry pi's built-in USB ports while the camera was attached to an onboard ribbon connector.

Relevant Faults

Much of the programming behind the components behaved as expected, but weak links in the chain of devices often caused failures. For the palm cameras, the ribbon cable connecting the camera in the palm to the raspberry pi mounted on the side of the hand was often sharply bent or punctured during operation. Initially, we had planned to encase the sensitive electronics in a guard around the hand, but this approach became cumbersome and was eventually discarded near the DRC Finals. As such, we could not produce a viable replacement protection and the camera cables became fragile equipment on a particularly heavy robot.

The tactile sensors suffered the interesting fault of having their communication wires ripped from their sockets regardless of their attached orientation. This caused communication issues on the sensors’ I2C buses often accompanied by a loss of data and a stalled state for each sensor. Efforts were made to programmatically reset the boards and continue on with the lost sensor, but our approaches were not robust enough for use in the DRC Finals.

C. OPERATOR STATION COMPONENTS

In addition to the UI components discussed in Section3.1.3, the OCS included a number of components that coordinated communications between the different operators and the onboard software. These non-UI components include:

- vigir_ocs_footstep_manager
  - Stores a stack of step plans (so we can undo/redo as needed)
  - Talks to the ocs footstep planner to plan footsteps based on local information only
  - Talks to the onboard footstep planner
    - Can talk directly to the planner to re-calculate ocs footstep plan based on data available onboard
    - Can use the onboard footstep manager to send minimal information onboard for planning in constrained communications
  - Receives information from the ocs/onboard planner, then creates and publishes visualizations
- vigir_ocs_template_nodelet (should have its name changed to manager)
  - Talks to the grasp widget and all the grasp components
  - Stores and publishes current templates
  - Handles template-related actions (add/remove/update)
  - Stores template/grasp information, affordances, template manipulation, etc (Alberto?)
- vigir_ocs_behavior_manager
  - Communication with behaviors
  - Handles requests, sends operator responses
  - Can handle multiple requests at the same time
  - "complexactionserver" (threaded action server)
  - from python to c++ and back to use python pickle for serialization
- vigir_ocs_global_hotkey
  - handles global (OS level) keyboard events and sends messages to the OCS views
- vigir_ocs_interactive_marker_server_nodelet
  - handles interactive markers added to the views
  - makes sure they are added correctly to all views
- vigir_ocs_robot_state_manager
  - singleton containing the robot state manager instances for the robot and ghost robot

These packages can be found in the vigir_ocs_common repository[47].

[47] http://github.com/team-vigir/vigir_ocs_common

This page intentionally blank.

D. ROBOT MODELING AND CONTROL

This appendix gives details about the different aspects of robot modeling, identification, and our impedance-based control approach. We start with a summary of the paper submission included at the end of Appendix D, continue with details about the integral torque controller in Appendix D.2, the friction identification experiments (Appendix D.3) and the examined friction compensation mechanisms (D.4), further experiments for arm dynamics parameter identification in Appendix D.5 and an evaluation of the compliance of the different controllers in Appendix D.6.

D.1. Summary of theoretical basics and basic experiments

A detailed summary of the theoretical approach and the experimental results of the joint impedance controller, the identification process and the disturbance observer can be seen in our submitted paper for the 2015 IEEE-RAS International Conference on Humanoid Robots, which is attached at then end of this appendix.

In Section I of the paper, we summarize the current state of the art of compliant control and impedance control for humanoid and hydraulic actuated robots. We come to the conclusion, that compliant control is absolutely necessary for humanoid robots in typical usage scenarios and that some promising approaches have already been researched in this field, which we combine in our control scheme for the Atlas robot.

In Section II.A of the paper, we give an overview of our kinematic, dynamic and friction model to describe the robot and the linear regressor formulation needed for feasible parameter identification. Section II.B continues with a description of our excitation trajectories for the identification based on Fourier series and polynomial functions. The parameter identification is done with a least squares approach weighted by sensor noise covariance. Section II.C explains the joint impedance controller approach and Section II.D. gives details about the formulation of the disturbance observer used for collision detection and model error compensation.

We begin our results in Section III.A of the paper with a description of the performance of our identification algorithm by comparing measured joint torques to the torques calculated from the identified model. This model accuracy was also experimentally validated by moving the arm in gravitation free mode, where a typical position teaching could be performed with only little drift in some poses. The high joint friction however helps to keep a position, if joint torque errors from model inaccuracies stay below the static friction.

This influence of the dynamic model is also shown in the first part of Section III.B. We show the abilities of the impedance controller compared to the existing tuned PD position controller: Our controller achieves a comparable position tracking and an improved velocity tracking.

Further we show the ability of the disturbance observer to qualitatively estimate the disturbance joint torque from model errors correctly. The ability to tune the impedance controller with different stiffness and damping coefficients is shown with a set of step response experiments.

D.2. Inner joint torque loop with integral feedback

In Section 3.2.1.3 Figure 19, it was pointed out that an integral controller is needed for the inner joint torque loop due to high steady-state-error of the proportional controller. Figure 50 shows the joint torque and position errors of the second arm joint for a complex trajectory with moderate velocity. With an increased integral gain, we could decrease the joint effort error about 90 % and decrease the joint position error for the hydraulic joints about 20 % in some movements and poses. The mean position error for this kind of trajectory could be reduced about 5% and for faster trajectories about 30%.

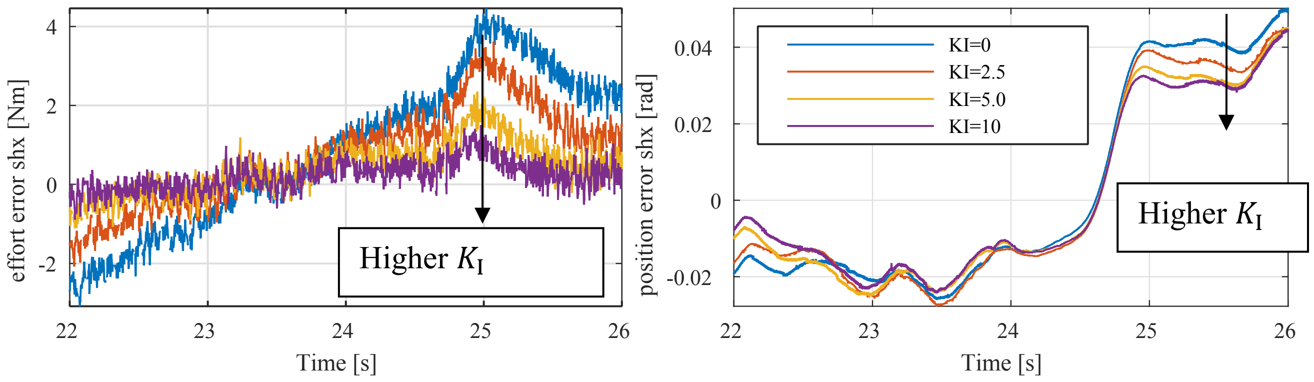


Figure 50. Torque and position error for different settings of integral inner torque loop

D.3. Friction identification

As already discussed in [1] and pointed out by other teams, the friction in the hydraulic valves has a strong influence on the quality of the arm control. Since the friction effects are located in the seals between the hydraulic pressure measurement and the actuated link, the measured torque always contains the friction. This especially influences the concept of joint impedance control, which normally assumes real joint torque measurements between gear friction of commonly used electric drives and the actuated link.

To identify joint friction, we executed trajectories with different constant velocities. Only with the model based controller and iteratively improved feedforward of dynamics and friction, we were able to run the trajectories smoothly without stick-slip-effect, which is shown in Figure 51 in comparison to the same experiment with the PD-position controller.

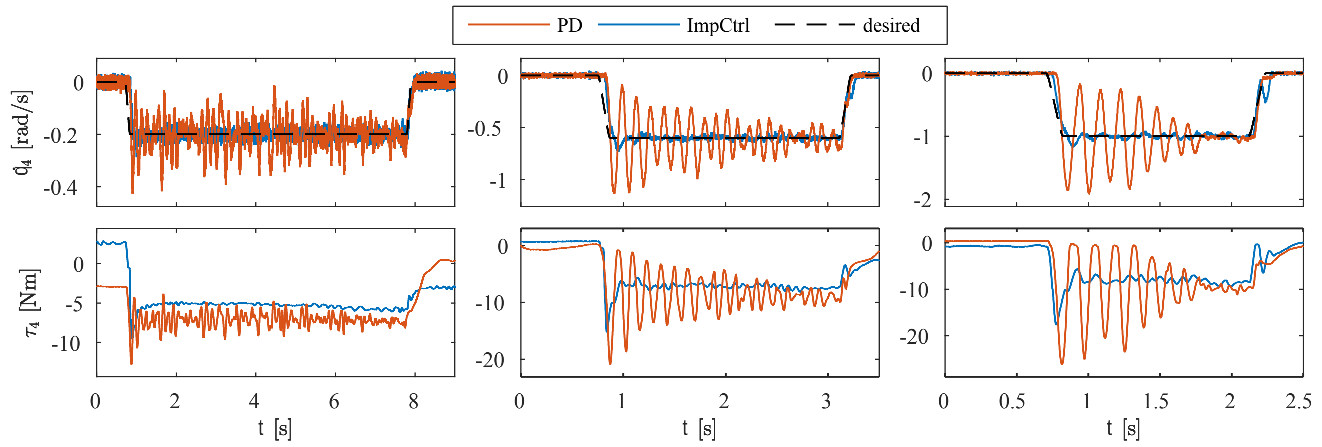


Figure 51. Velocity and joint torque plots for constant velocity trajectory tracking

The resulting friction curves with our viscous and Coulomb friction model are shown in Figure 52. The line marked as "mean" is calculated from a linear regression of the mean joint torque and velocity of the single experiments marked "exp.". The line marked as "raw" is calculated with a linear regression of all measured velocity and torque data points, which biases the friction identification, since trajectories with slow velocity take more time, produce more data points and are therefore weighted higher.

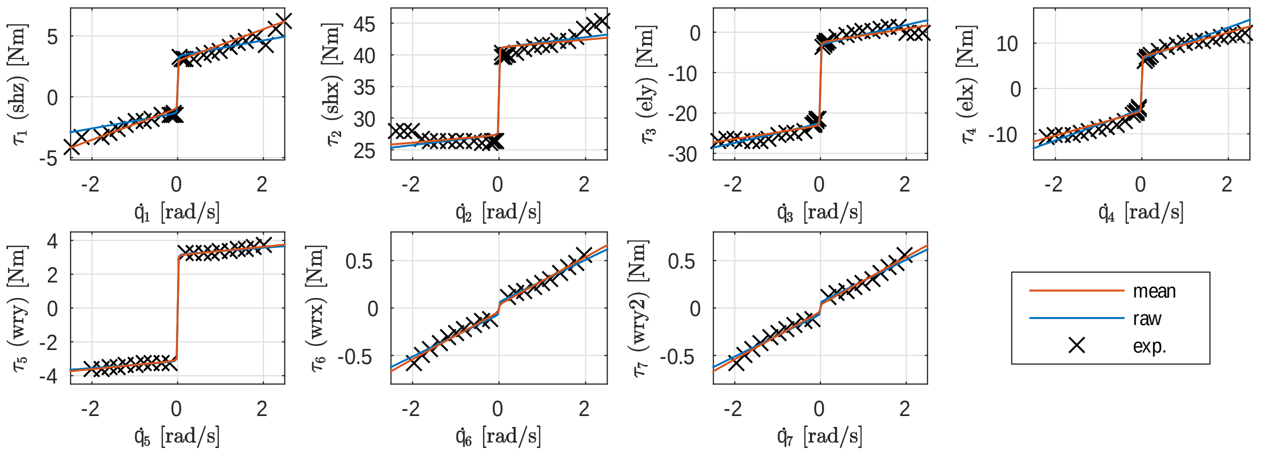


Figure 52. Joint friction diagrams from constant velocity experiments

D.4. Friction compensation and friction feedforward

We examined two different approaches for the friction compensation: Model based friction compensation with feedback of the measured velocity with

$$\hat{\tau}_{f,comp} = \mathrm{diag}(\hat{d}_v)\,\dot{q} + \mathrm{diag}(\hat{\mu}_C)\mathrm{sgn}(\dot{q})$$

and friction feedforward only as in

$$\hat{\tau}_{f,ff} = \mathrm{diag}(\hat{d}_v)\,\dot{q}_d + \mathrm{diag}(\hat{\mu}_C)\mathrm{sgn}(\dot{q}_d).$$

These terms are placed in the term $\hat{\tau}_f$ in Equ. (2) in D.7 included in this appendix.

Figure 53 shows the results of these two approaches compared to the impedance controller without compensation and the tuned PD position controller. An interval in a complex dynamic trajectory with moderate velocity is regarded with position, velocity, and effort of the third arm joint of the left arm.

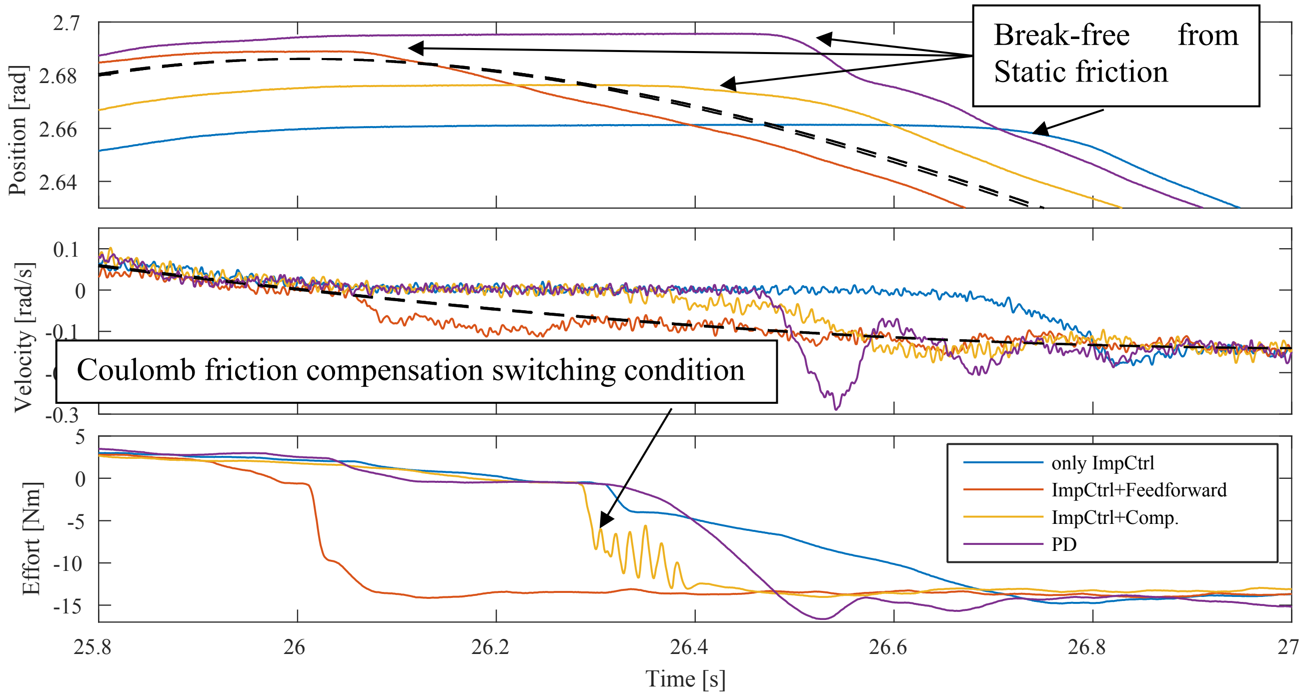


Figure 53. Comparison of mechanisms to cope with joint friction

With the friction feedforward control, the position tracking in intervals with low velocity is improved due to a better overcoming of the static friction. The position error decreased according to Table D-1 and is lower than with the PD position controller

Table D-1: Comparison of Cartesian errors with different friction handling modes

Mode	Mean Cartesian error [mm]	End Cartesian error [mm]
Only Impedance Controller	9.3	9.5
ImpCtrl. and friction feedforward	4.4	1.6
ImpCtrl. and friction compensation	7.9	4.9
Tuned PD position controller	13.1	3.7

The friction feedforward does not provide compliance in absence of a commanded velocity, since the arm friction is not compensated and the reaction force for low contact forces is the static friction. These low contact forces are not visible by the pressure sensors and therefore cannot be taken into account by the impedance controller. Since the main use-case of the impedance controller is to avoid falls after heavy collisions, we decided to use the friction feedforward with this drawback to compliance.

Also, our current implementation of the friction compensation cannot be set to the fully identified friction values from Figure 52 without having position oscillations with visibly high amplitude and low frequency. Therefore, the friction compensation compared above only uses friction coefficients reduced by ca. 50%. The oscillations probably result from the time delay and the switching between static and dynamic friction compensation.

D.5. Dynamic Arm identification

In addition to the identification results presented in [6], previously identified friction from Figure 7 in [6] was included to the robot regressor model. The aim was to reduce the parameter space from 59 to 45 unknowns and to improve the identification results by the implementation of more model based knowledge into the identification model. Assuming a robot arm model of

$$\boldsymbol{\tau}_{\mathrm{m}} = \boldsymbol{\Phi}\boldsymbol{\beta} - \boldsymbol{\tau}_{\mathrm{ext}}$$

from Eq. (7) of [6] (with $\boldsymbol{\tau}_{\mathrm{ext}} = 0$), the influence of a parametrized friction model with parameters $\boldsymbol{d}_{\mathrm{v}}$, $\boldsymbol{\mu}_{\mathrm{c}}$ can be incorporated by subtracting $\boldsymbol{\tau}_{\mathrm{f}} = \boldsymbol{\Phi}_{\mathrm{f}}\,(\boldsymbol{d}_{\mathrm{v}} \quad \boldsymbol{\mu}_{\mathrm{c}})^{\mathrm{T}}$ from both sides of Eq. (7). This effects the loss of friction related columns within the regressor formulation $\boldsymbol{\Phi} = \boldsymbol{\Phi}_{\mathrm{b}}$, which is represented by rigid body parameter only. The influence of friction to the motor torque $\boldsymbol{\tau}_{\mathrm{m}}$ can be written as $\boldsymbol{\tau}_{\mathrm{m,f}} = \boldsymbol{\tau}_{\mathrm{m}} - \boldsymbol{\Phi}_{\mathrm{f}}\,(\boldsymbol{d}_{\mathrm{v}} \quad \boldsymbol{\mu}_{\mathrm{c}})^{\mathrm{T}}$. The following procedure of the identification algorithm is kept equal as described in [6].

A comparison between the base parameter vector $\boldsymbol{\beta}_{\mathrm{hum15}}$ from [6], where friction was identified within the least squares optimization, and the base parameter vector $\boldsymbol{\beta}_{\mathrm{frct}}$, using single joint friction values, can be seen in Figure 54. The figure shows the model prediction to an unknown trajectory which should exclude the problem of self-fitting.

Similar performance in the torque prediction can be observed for the parameter vectors $\boldsymbol{\beta}_{\mathrm{hum15}}$ and $\boldsymbol{\beta}_{\mathrm{frct}}$ for the joints shz, shx, ely and elx. In shz improved results can be noticed by $\boldsymbol{\beta}_{\mathrm{frct}}$. However, both methods provide larger errors for this joint.

The following table shows the mean square errors between measured and modeled torques for the used parameter vectors $\boldsymbol{\beta}_{\mathrm{hum15}}$ and $\boldsymbol{\beta}_{\mathrm{frct}}$ for the hydraulic joints. Shz and shx show lower errors for parameter vector $\boldsymbol{\beta}_{\mathrm{frct}}$. Whereas, superior results are obtained in ely and and elx by $\boldsymbol{\beta}_{\mathrm{hum15}}$.

Table 2: Mean square errors at arm identification using different base parameter vectors

	Mean square error $\boldsymbol{\beta}_{\mathrm{hum15}}$[Nm²]	Mean square error $\boldsymbol{\beta}_{\mathrm{frct}}$ [Nm²]
shz	101.98	47.56
shx	29.63	25.62
ely	12.69	27.53
elx	14.18	31.22

As already mentioned in [6], the wrist joints do not seem to be identifiable by such global methods in case of this robot. Although friction was identified in single axis experiments, the predicted torques have no correlation with the measured torques for wry, wrx and wry2. The reasons can probably be found in small masses and inertias of the wrist elements, which effect a weak excitation of rigid body parameters, and in the use of current based torque measurement on actuator side, which are inferior to joint side torque measurements. The joint wry2 is not shown for illustration reasons because similar results to wrx and wry were achieved in which the model showed large errors.

The influence of Coulomb friction can be noticed within the plots by a clear step within the torque measurement which can only be explained by the signum function of the Coulomb term. Looking at shx these effects are described by the single axis identification of Fig. (7). In ely the peak of the Coulomb friction in $\boldsymbol{\beta}_{\mathrm{frct}}$ seems to be too high but the magnitude of the step height between modeled and measured torque matches. In this case the step is shifted by the terms of the rigid body model. Looking at elx a match as described in the previous example cannot be noticed, but different magnitudes in step height can be seen. Consequently, a single axis identification does provide correct Coulomb parameters in every case which is probably effected by time depending effects of the friction. For the remaining joints a statement cannot be made because no clear steps can be noticed.

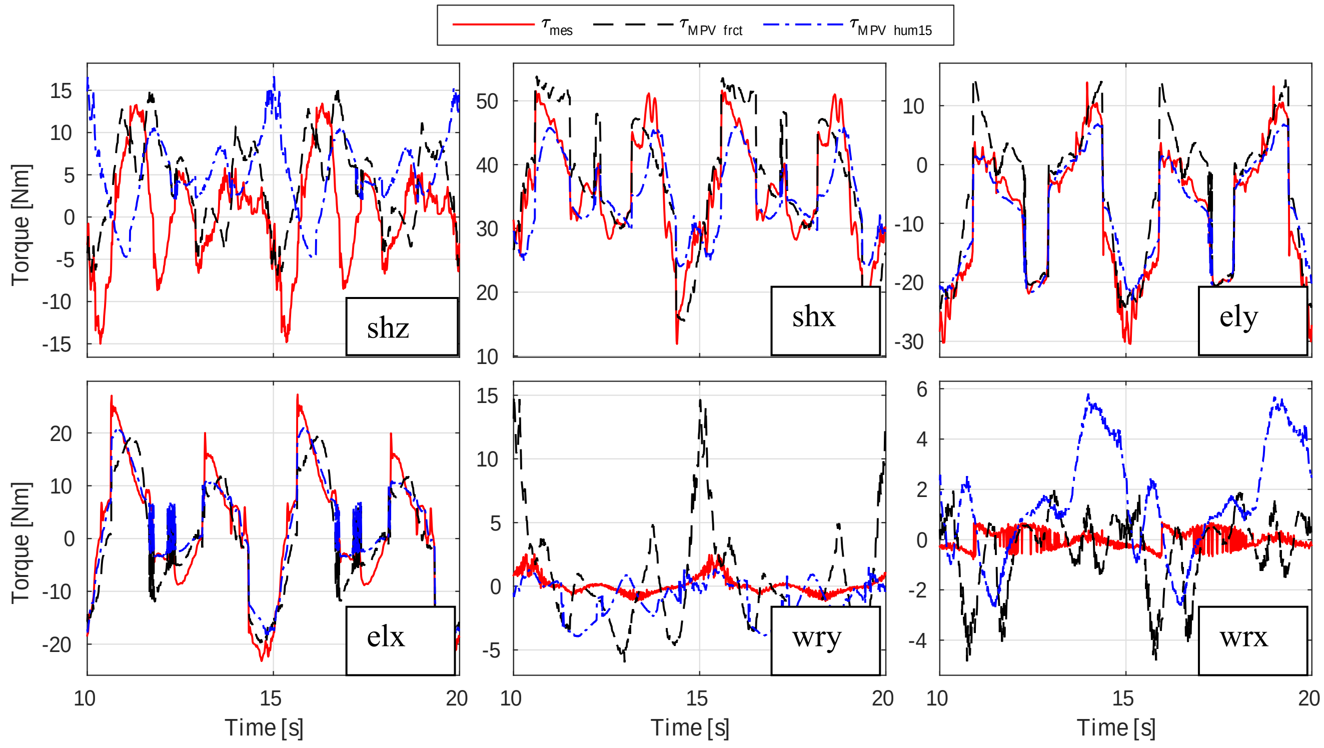


Figure 54. Measured and modeled torque for the left arm of ATLAS

As mentioned above, we discovered an inferior correlation between measured and modeled torques of shz in contrast to joints ely, elx. That is why we concluded a weak excitation of dynamic parameters which are related to potential energies of the arm in tilted poses. A cancelation of the related columns within the regressor formulation $\boldsymbol{\Phi}$ was not considered since a totally upright robot pose cannot be guaranteed completely for a humanoid robot. Therefore, we also executed the dynamic trajectories in two additional tilted poses shown in Figure 55 and implemented those results to the identification. The first identification results did not lead to an improved model correlation for shz. The reasons are subject of our ongoing research.

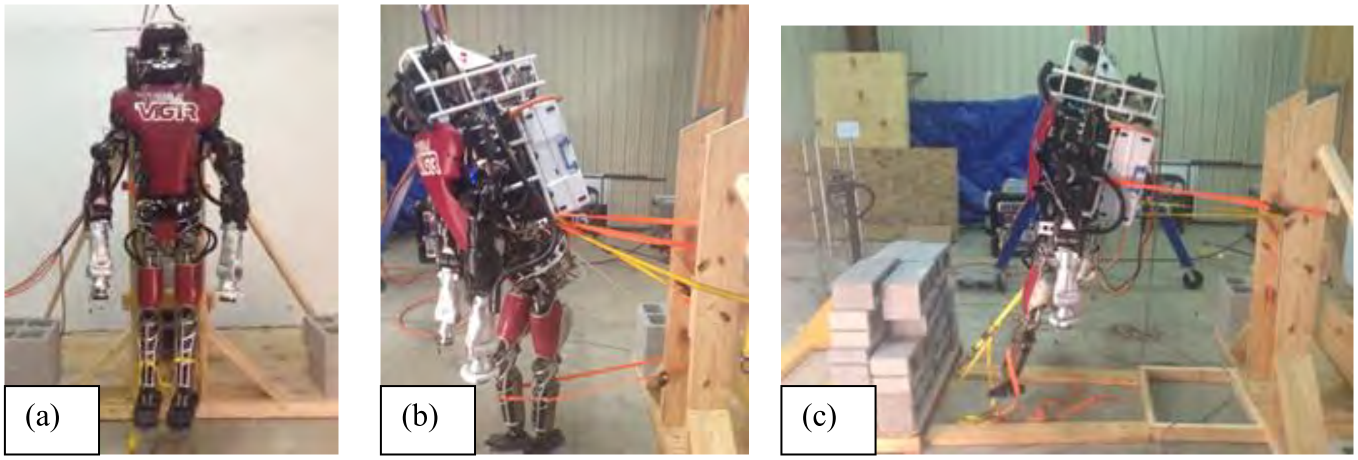


Figure 55. Different settings for the Robot with fixed upper body for arm identification

Finally, it can be concluded that the single axis identification of friction are a valid alternative method in contrast to a full identification of rigid body and friction parameters, but significant improvements could not be observed for the overall modeling accuracy by this approach. Possible explanations are probably time depending influences of friction within the robot joints.

The tilted orientation of the robot has not shown promising results yet. The identification of a robot arm does not seem to be an issue of covering arbitrary arm positions and robot orientations, but more a problem of finding those orientations which optimally excite all parameters. That is why the robot orientation should be taken into account for further trajectory optimizations within the identification procedure.

D.6. Compliance demonstration

In addition to the experiments mentioned in [6], we tested the compliance by placing an obstacle in the way of a typical grasping motion as depicted in Figure 56 and comparing the behavior in different manipulation modes: PD position controlled, impedance controlled with low stiffness and impedance controlled with high stiffness with and without collision detection.

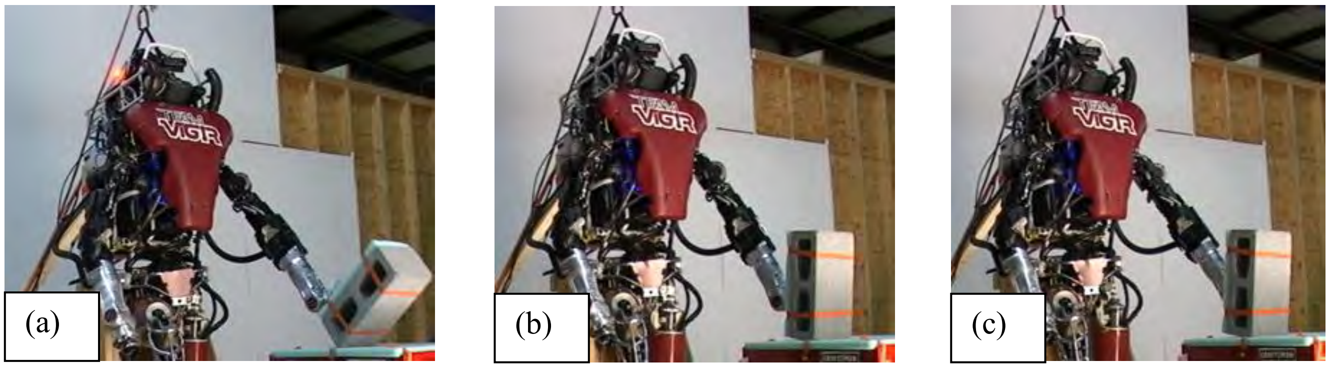


Figure 56. Experimental setup: High stiffness (a), low stiffness (b) and collision detection (c)

Figure 57 shows the measured values of force-torque-sensor, observed disturbance torque and joint position during a collision of the end-effector with a standing cinderblock using a Styrofoam protection.

With the PD position controller and the impedance controller set to a high joint stiffness of 300 Nm/rad, the end effector pushes the cinderblock out of the way and the collision forces reach about 70 N during the impact. If the robot was standing during this experiment and the collision force would be bigger, the robot would fall, as for example experienced in our tests before the finals. However both PD-position and stiff joint impedance controller achieve high position accuracies without the obstacle of respectively 6 mm and 4 mm at the end of the grasp motion. Figure 56-a depicts this result.

One mechanism to achieve compliant behavior is setting a low joint stiffness of 100 Nm/rad to the impedance controller. In our collision experiment the end effector pushes into the obstacle, but the collision force does not get high enough to push it away, so the arm gets stuck at the obstacle (see Figure 56-b). The position accuracy with low stiffness at the end of the grasping motion is about 9 mm and therefore only useful for safe transition motions, not for grasping motions (since this error increases significantly with attached hands and grasped objects).

Another mechanism to ensure a safe behavior after the collision is using the stiff impedance controller with collision detection based on the estimated disturbance joint torque. This approach currently allows a threshold for collision detection of about 10 Nm joint effort. After detecting the collision, the arm can be set into gravity-free mode, which is pointed out in the joint position plot in Figure 57 and can be seen in Figure 56-c.

The joint friction torque and our remaining dynamic identification model error is currently the limit for the collision detection threshold, since a wrongly estimated friction state in the disturbance observer could otherwise lead to a false collision alert.

Improving the identification of the dynamics model would allow decreasing the collision threshold further and detecting also minor collisions. With the current setting of the disturbance observer, it took about 300 ms to detect the collision with the cinderblock. With a higher observer Gain in Eqn. (22) of [6], a faster convergence of the disturbance observer can be achieved with the risk of overshoot in the observed disturbance torque exceeding the detection threshold.

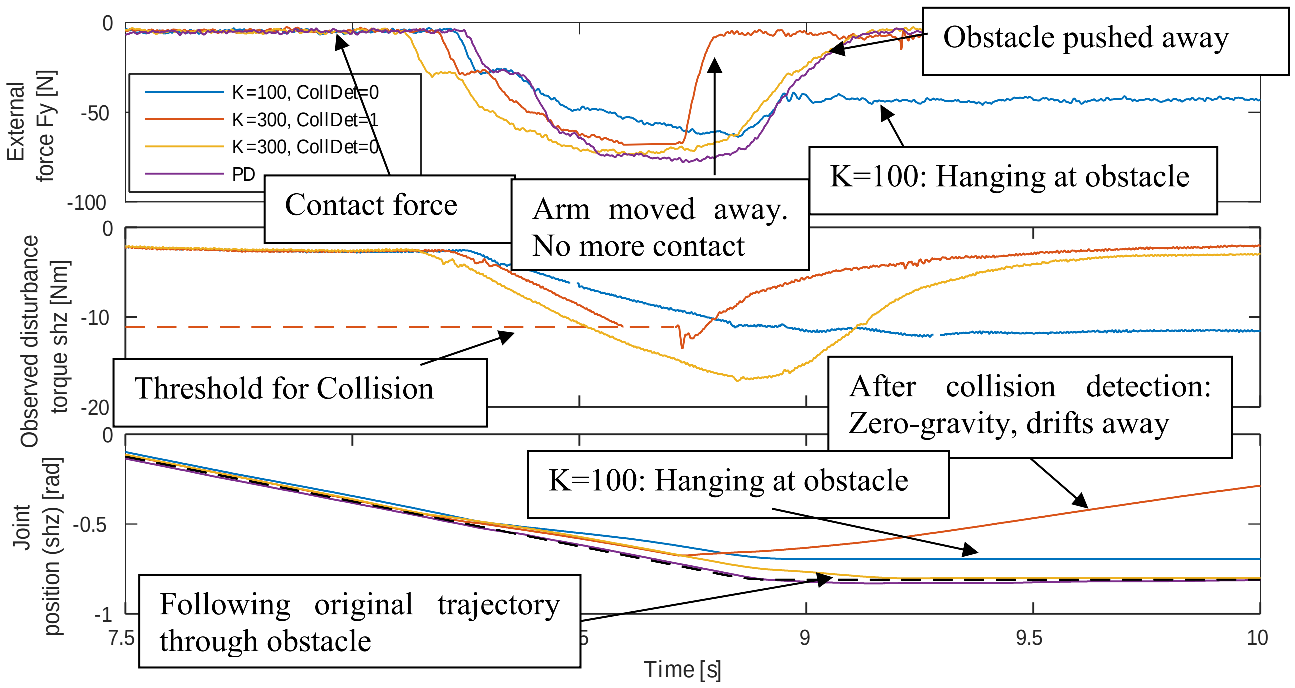


Figure 57. Typical measured forces, observed disturbance torque, and joint position

This page intentionally blank.

D.7. Humanoids 2015 Paper On Modeling and Control [6]

Modeling, Identification and Impedance Control of the Atlas Arms

Moritz Schappler[1], Jonathan Vorndamme[1], Alexander Tödtheide[2], David C. Conner[3], Oskar von Stryk[2], and Sami Haddadin[1]

***Abstract*— Compliant manipulation has become central to robots that are sought to safely act in and interact with unstructured as well as only partially known environments. In this paper we equip the hydraulically actuated, position controlled arms of the Atlas robot with model-based joint impedance control, including suitable damping design, and experimentally verify the proposed algorithm. Our approach, which originates from the advances in soft-robotics control, relies on high-performance low-level joint torque control. This makes it independent from the actual technology being hydraulic or electromechanical. This paper describes the approach to accurately model the dynamics, and design the optimal excitation trajectory for system identification to enable the specification of model-based feed-forward controls. In conclusion, the implemented controller enables the robot arm to execute significantly smoother motions, be compliant against external forces, and have similar tracking performance as compared to the existing position control scheme. Finally, unknown modeling inaccuracies and contact forces are accurately estimated by a suitable disturbance observer, which will be used in the future to further enhance our controller's performance.**

I. INTRODUCTION AND STATE OF THE ART

In the DARPA Robotics Challenge (DRC), the task of the Atlas humanoid robot developed by Boston Dynamics, Inc. as well as other platforms was to perform several manipulation tasks like steering a car, opening a door, milling a circle in a drywall or inserting a plug [18]. For achieving such tasks, high-performance and robust manipulation as well as locomotion skills are essential. One of the consequential core problems are unforeseen collisions, possibly along the entire robot structure, and uneven terrain. Both effects may cause the robot to accidentally fall or get directly damaged due to the collisions themselves. A solution to this problem is compliant behavior in order to safely and robustly absorb possibly harmful contacts. In order to handle reactions to external forces, force and compliance control were intensively researched [6], [27], [30], [35]. However, the presumably most common and successful approach is impedance control, which was introduced by Hogan in his seminal work [12]. Significant extensions towards compliant whole-body humanoid control for standing/walking control can e.g. be found in [13], [19], [25], [31]. Nonetheless, compliance control has most frequently been used in force-sensitive manipulation. In [7], [24], the DLR Justin system makes use of hybrid Cartesian and joint level impedance controller for the arms and hands. In [28] grasping applications of the the Armar-III humanoid robot, using a compliant Jacobian-based Cartesian velocity controller, were introduced. Bimanual compliant control for the COMAN robot was researched in [1] using a joint impedance controller with stiffness transformation from the task space. Generally, the aforementioned control concepts for *electromechanically* actuated robots are termed *soft-robotics control* and strongly rely on a high-performance low-level torque control loop.

In the context of *hydraulically* actuated humanoids, [16] investigated compliance control for a single hydraulic joint of the UT-$\theta 2$ humanoid. In [14] a compliant full-body force control framework was introduced and applied to the SARCOS humanoid. The controller was implemented as an internal loop for an overlying balancing controller. A first compliant manipulation approach for the Atlas robot using the concept of admittance control (position based impedance control) was introduced in [20].

Since soft-robotics control approaches (like impedance control) require very good joint torque measurement and accurate dynamics models, the modeling and identification of all significant mechanical influences is a central aspect in model-based robot control. Several well established identification methods were developed for fixed base robots [3], [4], [10], [21], [29]. In [33] an arm parameter identification of the iCub humanoid robot was done identifying inertial, friction and electric motor parameters using an additional 6 DOF (degrees-of-freedom) torque sensor in the arm. The identification problem for full humanoids and the associated problems regarding missing measurements has been thoroughly investigated in [5], [15], [22], [23], [34].

In this paper, we apply state-of-the-art modeling, identification, and control concepts from soft-robotics control to the 7-degrees-of-freedom arm of the Atlas humanoid system. The goal is to understand how these schemes apply to hydraulically actuated systems, validate their performance in comparison to the standard hydraulics position controllers that come with the system, and show possible next steps towards a fully elaborated whole body impedance manipulation framework for Atlas. The contributations of this paper are

1) the design and experimental evaluation of a model-based joint impedance controller with according state-dependent damping and feed-forward control for the hybrid hydraulic/electric Atlas arms,
2) the design and validation of a fully automatic identification scheme for a humanoid robot arm based on optimal excitation trajectories under the assumption of a fixed arm base, and
3) the initial investigation of an improved joint torque

[1]The authors is a member of the Institute of Automatic Control at Leibniz Universität Hanover, lastname@irt.uni-hannover.de
[2]The authors is a member of the Simulation, Systems Optimization and Robotics Group at TU Darmstadt, lastname@sim.tu-darmstadt.de
[3]The author is with TORC Robotics and the leader of the DRC team ViGIR, conner@torcrobotics.com

Preprint submitted to 2015 IEEE-RAS International Conference on Humanoid Robots.
Received July 7, 2015.

estimation using a model-based nonlinear disturbance observer.

Fig. 1 depicts the structure of the paper. Section II gives an overview of the modeling methods (II-A), the identification approach (II-B), and the control scheme (II-C). Section III provides our identification results (III-A) and the overall performance of our controller (III-B). Finally, Section IV concludes the paper.

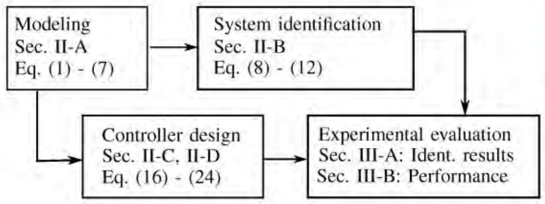


Fig. 1. Flow chart of steps with references to related chapters and equations

II. METHODS OF MODELLING, IDENTIFICATION AND CONTROL

A. Robot Dynamics and Joint Friction Model

The Atlas robot of version 5 consists of 24 hydraulic and 6 electromechanical joints, see Fig. 2. As mentioned above, we focus in this paper on modeling, identification, and control of the 7-DOF arms. The arms make use of a hybrid actuation concept: The first four joints (shoulder and elbow) are driven by hydraulic actuators. Joint torques are generated by pressure differences within the chambers of the actuator. The lower three joints (forearm and wrist) consist of an electric motor, gearbox, position encoder and a torque sensor (which however is not operational). Both hydraulic and electric joints make use of a hybrid position and torque control law provided by Boston Dynamics, Inc (BDI), the developer of the Atlas hardware. Following simplifications, which are certainly valid during the typical execution speeds, were made: For simplicity and to protect BDI proprietary information, we consider both torque controlled actuators as ideal torque sources. The limitations of this simplification will be discussed in Sec. III-A. Furthermore, we use a fixed base arm model, which is suitable for slow torso movements. This requires only the known orientation of the upper torso and not the full base acceleration vector.

TABLE I
MDH PARAMETERS OF THE ATLAS ARMS (VERSION 5)

i	1	2	3	4	5	6	7
α	0	$-\pi/2$	$\pi/2$	$-\pi/2$	$\pi/2$	$-\pi/2$	$\pi/2$
a	0	a_2	a_3	a_4	a_5	0	0
θ	q_1	$q_2-\pi/2$	q_3	q_4	q_5	q_6	q_7
d	d_1	0	d_3	0	d_5	0	0

Arm kinematics can be easily determined with modified DH-parameter (MDH) convention [17], see Tab. I. The arm dynamics is modeled using the rigid body model

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}+\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}+\boldsymbol{g}(\boldsymbol{q})=\boldsymbol{\tau}_{\mathrm{m}}-\boldsymbol{\tau}_{\mathrm{f}}+\boldsymbol{\tau}_{\mathrm{ext}}, \tag{1}$$

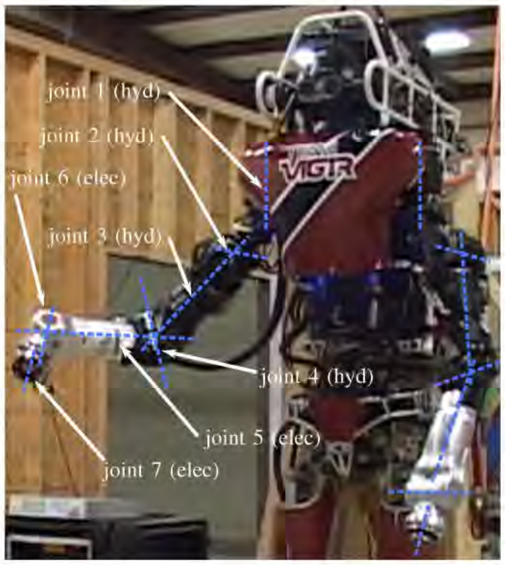


Fig. 2. Atlas version 5 with hybrid actuation concept

with generalized joint positions $\boldsymbol{q}\in\mathbb{R}^{n_{\mathrm{j}}}$, inertia matrix $\boldsymbol{M}(\boldsymbol{q})$, centrifugal and coriolis matrix $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$, gravity vector $\boldsymbol{g}(\boldsymbol{q})$, electric or hydraulic actuator torques $\boldsymbol{\tau}_{\mathrm{m}}$, friction torques $\boldsymbol{\tau}_{\mathrm{f}}$ and external torques $\boldsymbol{\tau}_{\mathrm{ext}}$. Joint friction is assumed to consist of Coulomb and viscous terms

$$\boldsymbol{\tau}_{\mathrm{f}}=\mathrm{diag}(\boldsymbol{d}_{\mathrm{v}})\dot{\boldsymbol{q}}+\mathrm{diag}(\boldsymbol{\mu}_{\mathrm{c}})\,\mathrm{sgn}(\dot{\boldsymbol{q}}), \tag{2}$$

where $\boldsymbol{d}_{\mathrm{v}}$ and $\boldsymbol{\mu}_{\mathrm{c}}\in\mathbb{R}^{n_{\mathrm{j}}}$ are the viscous and Coulomb friction coefficients for each joint.

For identification purpose the inverse dynamics model is transfered to its linear regressor form $\boldsymbol{\Phi}_{\mathrm{b}}\in\mathbb{R}^{n_{\mathrm{j}}\times n_{\mathrm{b}}}$, using the base parameters $\boldsymbol{\beta}_{\mathrm{b}}\in\mathbb{R}^{n_{\mathrm{b}}\times 1}$ of the rigid body system from [9]. This gives

$$\begin{aligned}\boldsymbol{\Phi}_{\mathrm{b}}\boldsymbol{\beta}_{\mathrm{b}}&=(\boldsymbol{\Phi}_{M}(\boldsymbol{q},\ddot{\boldsymbol{q}})+\boldsymbol{\Phi}_{c}(\boldsymbol{q},\dot{\boldsymbol{q}})+\boldsymbol{\Phi}_{g}(\boldsymbol{q}))\boldsymbol{\beta}_{\mathrm{b}}\\&=\boldsymbol{\tau}_{\mathrm{m}}-\boldsymbol{\tau}_{\mathrm{f}}+\boldsymbol{\tau}_{\mathrm{ext}}.\end{aligned} \tag{3}$$

The regressor formulation of the joint friction model is simply

$$\boldsymbol{\tau}_{\mathrm{f}}=\boldsymbol{\Phi}_{\mathrm{f}}(\dot{\boldsymbol{q}})\begin{pmatrix}\boldsymbol{d}_{\mathrm{v}}\\\boldsymbol{\mu}_{\mathrm{c}}\end{pmatrix}, \tag{4}$$

where

$$\boldsymbol{\Phi}_{\mathrm{f}}(\dot{\boldsymbol{q}})=\begin{pmatrix}\mathrm{diag}(\dot{\boldsymbol{q}}) & \mathrm{diag}(\mathrm{sgn}(\dot{\boldsymbol{q}}))\end{pmatrix}\in\mathbb{R}^{n_{\mathrm{j}}\times 2n_{\mathrm{j}}}. \tag{5}$$

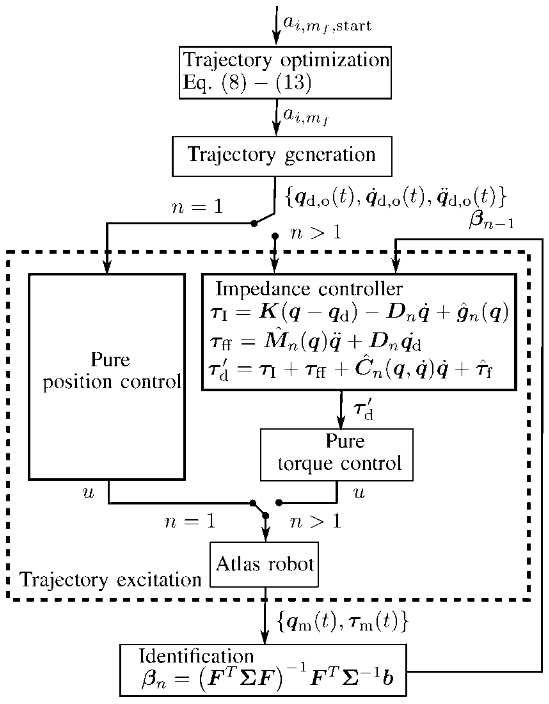
This allows the definition of an extended parameter vector $\boldsymbol{\beta}$ and a new regressor formulation $\boldsymbol{\Phi}$ for the combined model

$$\boldsymbol{\Phi}=\begin{pmatrix}\boldsymbol{\Phi}_{\mathrm{b}} & \boldsymbol{\Phi}_{\mathrm{f}}\end{pmatrix},\boldsymbol{\beta}=\begin{pmatrix}\boldsymbol{\beta}_{\mathrm{b}}\\\boldsymbol{d}_{\mathrm{v}}\\\boldsymbol{\mu}_{\mathrm{c}}\end{pmatrix}\in\mathbb{R}^{n_{\beta}}. \tag{6}$$

Now, the motor torques $\boldsymbol{\tau}_{\mathrm{m}}$ can be written as

$$\boldsymbol{\tau}_{\mathrm{m}}=\boldsymbol{\Phi}\boldsymbol{\beta}-\boldsymbol{\tau}_{\mathrm{ext}}. \tag{7}$$

Preprint submitted to 2015 IEEE-RAS International Conference on Humanoid Robots.
Received July 7, 2015.



$a_{i,m_f,\mathrm{start}}$
Trajectory optimization
Eq. (8) – (13)
a_{i,m_f}
Trajectory generation
$\{\boldsymbol{q}_{\mathrm{d,o}}(t), \dot{\boldsymbol{q}}_{\mathrm{d,o}}(t), \ddot{\boldsymbol{q}}_{\mathrm{d,o}}(t)\}$
$n = 1$
$n > 1$
$\boldsymbol{\beta}_{n-1}$
Pure position control
Impedance controller
$\boldsymbol{\tau}_{\mathrm{I}} = \boldsymbol{K}(\boldsymbol{q} - \boldsymbol{q}_{\mathrm{d}}) - \boldsymbol{D}_n\dot{\boldsymbol{q}} + \hat{\boldsymbol{g}}_n(\boldsymbol{q})$
$\boldsymbol{\tau}_{\mathrm{ff}} = \hat{\boldsymbol{M}}_n(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{D}_n\dot{\boldsymbol{q}}_{\mathrm{d}}$
$\boldsymbol{\tau}'_{\mathrm{d}} = \boldsymbol{\tau}_{\mathrm{I}} + \boldsymbol{\tau}_{\mathrm{ff}} + \hat{\boldsymbol{C}}_n(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \hat{\boldsymbol{\tau}}_{\mathrm{f}}$
$\boldsymbol{\tau}'_{\mathrm{d}}$
Pure torque control
u
u
$n = 1$
$n > 1$
Atlas robot
Trajectory excitation
$\{\boldsymbol{q}_{\mathrm{m}}(t), \boldsymbol{\tau}_{\mathrm{m}}(t)\}$
Identification
$\boldsymbol{\beta}_n = \left(\boldsymbol{F}^T\boldsymbol{\Sigma}\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{b}$

Fig. 3. Flow chart of steps for parameter identification

B. Dynamics and friction identification

In the following, the identification routine for obtaining the rigid body and friction parameters similar to [26] is shortly reviewed. The goal is to find the numerical values of the parameter vector $\boldsymbol{\beta}$, which is part of models (3) or (7). In turn, the model can then be used in the feed-forward controls and the model based impedance controller. Fig. 3 shows the overall automatic optimization and arm identification procedure. Prior to the experimental data recording, optimal excitation trajectories are generated offline as described in Eq. (8) - (13) to facilitate later convergence to a feasible solution $\boldsymbol{\beta}$. The identification routine is then performed repeatedly based on the previously identified parameter vector $\boldsymbol{\beta}_{n-1}$ in order to improve model quality and thus tracking performance iteratively until the trackig error converges. Next, the details on the generation of optimal excitation trajectories are discussed.

1) Optimal identification trajectories: According to [26] a sum of Fourier series and a polynominal

$$q_{\mathrm{o},i}(t) = \lambda_i(t) + \delta_i(t) \tag{8}$$

of a joint i can be used as an parameterizable model function, where

$$\delta_i(t) = \sum_{m_f=1}^{5} a_{i,m_f} \cos\left(\frac{m_f \pi}{t_f} t\right) \tag{9}$$

is the Fourier series with coefficents a_{im_f}. The duration of a single period is given by t_f. The function

$$\lambda_i(t) = \sum_{j=0}^{5} \lambda_{ij} t^j \tag{10}$$

denotes a time-depending polynomial of 5th order and coefficients λ_{ij}, which are obtained by solving a linear equation system with the constraints

$$\begin{aligned} q_i(0) &= \lambda_i(0) + \delta_i(0) & q_i(t_f) &= \lambda_i(t_f) + \delta_i(t_f) \\ \dot{q}_i(0) &= \dot{\lambda}_i(0) + \dot{\delta}_i(0) & \dot{q}_i(t_f) &= \dot{\lambda}_i(t_f) + \dot{\delta}_i(t_f) \\ \ddot{q}_i(0) &= \ddot{\lambda}_i(0) + \ddot{\delta}_i(0) & \ddot{q}_i(t_f) &= \ddot{\lambda}_i(t_f) + \ddot{\delta}_i(t_f). \end{aligned} \tag{11}$$

An information matrix $\boldsymbol{F} \in \mathbb{R}^{n_j \cdot N \times n_\beta}$ of N data samples is built by stacking the regressor matrices $\boldsymbol{\Phi}$ for every time step t_k in dependency of the trajectory data $\boldsymbol{q}_{\mathrm{o}}(t)$, $\dot{\boldsymbol{q}}_{\mathrm{o}}(t)$ and $\ddot{\boldsymbol{q}}_{\mathrm{o}}(t)$ with

$$\boldsymbol{F} = \begin{pmatrix} \boldsymbol{\Phi}\left(\boldsymbol{q}_{\mathrm{o}}(t_1), \dot{\boldsymbol{q}}_{\mathrm{o}}(t_1), \ddot{\boldsymbol{q}}_{\mathrm{o}}(t_1)\right) \\ \boldsymbol{\Phi}\left(\boldsymbol{q}_{\mathrm{o}}(t_2), \dot{\boldsymbol{q}}_{\mathrm{o}}(t_2), \ddot{\boldsymbol{q}}_{\mathrm{o}}(t_2)\right) \\ \vdots \\ \boldsymbol{\Phi}(\boldsymbol{q}_{\mathrm{o}}(t_f), \dot{\boldsymbol{q}}_{\mathrm{o}}(t_f), \ddot{\boldsymbol{q}}_{\mathrm{o}}(t_f)) \end{pmatrix}. \tag{12}$$

The optimization of the trajectory was performed by minimizing the condition number κ of the information matrix $\boldsymbol{F}$. Further conditions like minimum and maximum joint positions/velocities, as well as collisions between end-effector and torso were taken into account as well. This has been done by defining the valid translation endeffector set $\boldsymbol{P}_{\mathrm{EE,v}}$. Thus, the minimization problem can be written as

$$\begin{aligned} \{a^*_{i,m_f}\} &= \arg\min_i \kappa(\boldsymbol{F}(\boldsymbol{q}_{\mathrm{ident}}, \dot{\boldsymbol{q}}_{\mathrm{ident}}, \ddot{\boldsymbol{q}}_{\mathrm{ident}})), \\ \text{for } \boldsymbol{q}_{\mathrm{ident}} &= \boldsymbol{q}_{\mathrm{o}}(\{t_0, \ldots, t_f\}, a_{i,m_f}) \\ & s.t. \\ \boldsymbol{q}_{\mathrm{o,min}} &\leq \boldsymbol{q}_{\mathrm{o}} \leq \boldsymbol{q}_{\mathrm{o,max}} \\ \dot{\boldsymbol{q}}_{\mathrm{o,min}} &\leq \dot{\boldsymbol{q}}_{\mathrm{o}} \leq \dot{\boldsymbol{q}}_{\mathrm{o,max}} \\ \boldsymbol{p}_{\mathrm{EE}}(\boldsymbol{q}_{\mathrm{o}}) &\in \boldsymbol{P}_{\mathrm{EE,v}}. \end{aligned} \tag{13}$$

Within the optimization algorithm we chose $t_f = 5$ s, $N = 1500$ and an arbitrary start vector $\{a_{i,m_f,\mathrm{start}} = 1 | i = 1 \cdots n_j, m_f = 1 \cdots 5\}$. The achieved condition number was $\kappa = 44.4$.

2) Identification algorithm: The parameter vector $\boldsymbol{\beta}$, containing rigid body and friction parameters, can be identified by a weighted least squares estimate

$$\boldsymbol{\beta} = \left(\boldsymbol{F}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{b}. \tag{14}$$

The information matrix $\boldsymbol{F}$ is filled with measurements of $\boldsymbol{q}_{\mathrm{m}}$, $\dot{\boldsymbol{q}}_{\mathrm{m}}$ and $\ddot{\boldsymbol{q}}_{\mathrm{m}}$, where velocity $\dot{\boldsymbol{q}}_{\mathrm{m}}$ and acceleration $\ddot{\boldsymbol{q}}_{\mathrm{m}}$ were determined by numerical differentiation. The impact of noise was mitigated by a phaseless filter of first/second order. The measurement vector

$$\boldsymbol{b} = \begin{pmatrix} \boldsymbol{\tau}_{\mathrm{m}}(t_1) \\ \boldsymbol{\tau}_{\mathrm{m}}(t_2) \\ \vdots \\ \boldsymbol{\tau}_{\mathrm{m}}(t_f) \end{pmatrix} \in \mathbb{R}^{n_j N} \tag{15}$$

is the stacked vector of measured torques τ_{m} for every time step t_k. The variance $\boldsymbol{\Sigma}$ of hydraulic and electric actuators was determined in steady state from noise measurements.

C. Joint Impedance Control and Feedforward

Having now modeled and identified the Atlas arm, the basic joint impedance control law

$$\boldsymbol{\tau}_{\mathrm{I}} = \boldsymbol{K}(\boldsymbol{q}_{\mathrm{d}} - \boldsymbol{q}) - \boldsymbol{D}\dot{\boldsymbol{q}} + \hat{\boldsymbol{g}}(\boldsymbol{q}) \tag{16}$$

can be formulated, where $\boldsymbol{K}$ denotes the desired joint stiffness matrix. The desired joint damping matrix $\boldsymbol{D}$ is obtained with the factorization damping design approach

$$\boldsymbol{D} = \sqrt{\boldsymbol{M}(\boldsymbol{q})}\boldsymbol{D}_{\xi}\sqrt{\boldsymbol{K}} + \sqrt{\boldsymbol{K}}\boldsymbol{D}_{\xi}\sqrt{\boldsymbol{M}(\boldsymbol{q})}, \tag{17}$$

from [2] where the square root of a positive definite matrix $\boldsymbol{M}$ being the positive definite matrix $\sqrt{\boldsymbol{M}}$ that fulfills $\sqrt{\boldsymbol{M}}\sqrt{\boldsymbol{M}} = \boldsymbol{M}$. The diagonal matrix $\boldsymbol{D}_{\xi}$ contains the desired damping coefficients. Centrifugal and coriolis compensation terms $\boldsymbol{\tau}_{\mathrm{c}}$ and inertial and damping feedforward $\boldsymbol{\tau}_{\mathrm{ff}}$ are finally simply computed as

$$\boldsymbol{\tau}_{\mathrm{c}} = \hat{\boldsymbol{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}. \tag{18}$$

$$\boldsymbol{\tau}_{\mathrm{ff}} = \hat{\boldsymbol{M}}(\boldsymbol{q})\ddot{\boldsymbol{q}}_{\mathrm{d}} - \boldsymbol{D}\dot{\boldsymbol{q}}_{\mathrm{d}} \tag{19}$$

Adding friction compensation $\hat{\boldsymbol{\tau}}_{\mathrm{f}}$ from model (2) to the controllers (16), (18) and (19), the full desired joint torque becomes

$$\boldsymbol{\tau}'_{\mathrm{d}} = \boldsymbol{\tau}_{\mathrm{I}} + \boldsymbol{\tau}_{\mathrm{c}} + \boldsymbol{\tau}_{\mathrm{ff}} + \hat{\boldsymbol{\tau}}_{\mathrm{f}} \ . \tag{20}$$

The last missing part of our controller is a compensator for all unknown disturbances, which is detailed next.

D. Disturbance Estimation

The torque

$$\boldsymbol{\tau}_{\varepsilon} = \boldsymbol{\delta}_{\mathrm{dyn}} + \boldsymbol{\delta}_{\mathrm{f}} + \boldsymbol{\delta}_{\mathrm{ext}} + \boldsymbol{\delta}_{\mathrm{off}} + \boldsymbol{\delta}_{\mathrm{noise}} \tag{21}$$

lumping together all unknown disturbances contains all errors $\boldsymbol{\delta}_{\mathrm{dyn}}$ from the assumed dynamic model (1) such as parameter identification errors and forces from torso movement, errors $\boldsymbol{\delta}_{\mathrm{f}}$ of the friction model (2), external forces $\boldsymbol{\delta}_{\mathrm{ext}}$, sensor offsets $\boldsymbol{\delta}_{\mathrm{off}}$ and sensor noise $\boldsymbol{\delta}_{\mathrm{noise}}$. This disturbance torque is estimated with a generalized momentum based disturbance observer from [11] by

$$\hat{\boldsymbol{\tau}}_{\varepsilon} = \boldsymbol{K}_{\mathrm{O}}\left(\int_0^T [\boldsymbol{\tau}_{\mathrm{m}} - \boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \hat{\boldsymbol{\tau}}_{\mathrm{f}} - \hat{\boldsymbol{\tau}}_{\varepsilon}]\mathrm{d}t - \hat{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}}\right), \tag{22}$$

where

$$\boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \hat{\boldsymbol{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \hat{\boldsymbol{g}}(\boldsymbol{q}) - \dot{\hat{\boldsymbol{M}}}(\boldsymbol{q})\dot{\boldsymbol{q}} \ . \tag{23}$$

The matrix $\boldsymbol{K}_{\mathrm{O}}$ denotes the observer gain. It can be shown that $\hat{\boldsymbol{\tau}}_{\varepsilon}$ is a first order filtered version of the true disturbance torques and thus no nonlinear effects detoriate the performance of the scheme. The full commanded desired joint torque output of the controller is finally

$$\boldsymbol{\tau}_{\mathrm{d}} = \boldsymbol{\tau}'_{\mathrm{d}} + \hat{\boldsymbol{\tau}}_{\varepsilon} \ . \tag{24}$$

III. EXPERIMENTAL RESULTS

In this section an experimental evaluation of the identified model and the impedance controller is presented. The arm identification and impedance control validation experiments were performed on a wooden rack structure (Fig. 2) to suppress disturbance effects like swaying during stand mode.

Please note, that our results were gained under the following limitations:

- Since our impedance controller makes use of a proportional force controller, the overall tracking performance depends significantly on tuning and limitations of the proportional gain approach.
- The upright alignment of the robot during identification presumably leads to a weak excitation of shoulder-specific rigid body parameters (joint one). A tilted fixation of the whole robot might solve this problem but could not be realized in our experiments.
- The first right shoulder joint probably suffered from a broken pressure sensor, which led to false torque measurements.

A. Identification Results

1) General Workflow: The following identification procedure was done iteratively under the use of a generated optimal arm trajectory, see Fig. 3. The first approach was run with the BDI position controller. This basic model-free controller suffered from noticable vibrations, which led to difficulties in determining acceleration for the identification scheme. Three additional iterations were done applying each previously determined base parameter set β_{n-1} to our impedance controller. This significantly increased tracking and thus also identification performance.

2) Experiment: In the following experiment (Fig. 4) different highly dynamic optimal trajectories were applied to each robot arm in order to validate the tracking performance of the impedance controller and the validity of the identified model. The desired joint stiffness for this experiment was set to $K = 300$ Nm/rad, using the provided BDI force controller as the inner cascade. Furthermore, the mechanical model of (3) and friction model (2) with friction/stiction compensation were applied as feed forward term. The applied optimal trajectory is different from the one used for identification. Fig. 4 (a) and (c) show the angular position tracking between $\boldsymbol{q}$ and $\boldsymbol{q}_{\mathrm{d}}$ for the left and right arm for joints $1-7$. In columns (b) and (d) measured and modelled torques, $\boldsymbol{\tau}_{\mathrm{m}}$ and $\boldsymbol{\tau}_{\mathrm{ident}} = \boldsymbol{\Phi}\beta$ are depicted, which were existent while running trajectory (a) and (c), respectively. Additionally, Tab. II and III show maximum speed and accelerations of the hydraulic joints $1-4$ and statistical values of absolute mean errors and variances regarding to torque and position tracking.

3) Results: It can be seen that desired angles are tracked correctly for all joints. In joint 1 of the right arm an increased error for negative angles can be observed. This reflects the technical problems of the joint mentioned in the beginning of Sec. III. Investigating the joint torques, it can be seen that the shape of measured torques can be predicted correctly for the hydraulic joints $1-4$. In joint 1 a detoriated prediction can be observed, which is due to the high error and variance values in Tab. II and III. This is probably also effected by a weak

Preprint submitted to 2015 IEEE-RAS International Conference on Humanoid Robots.
Received July 7, 2015.

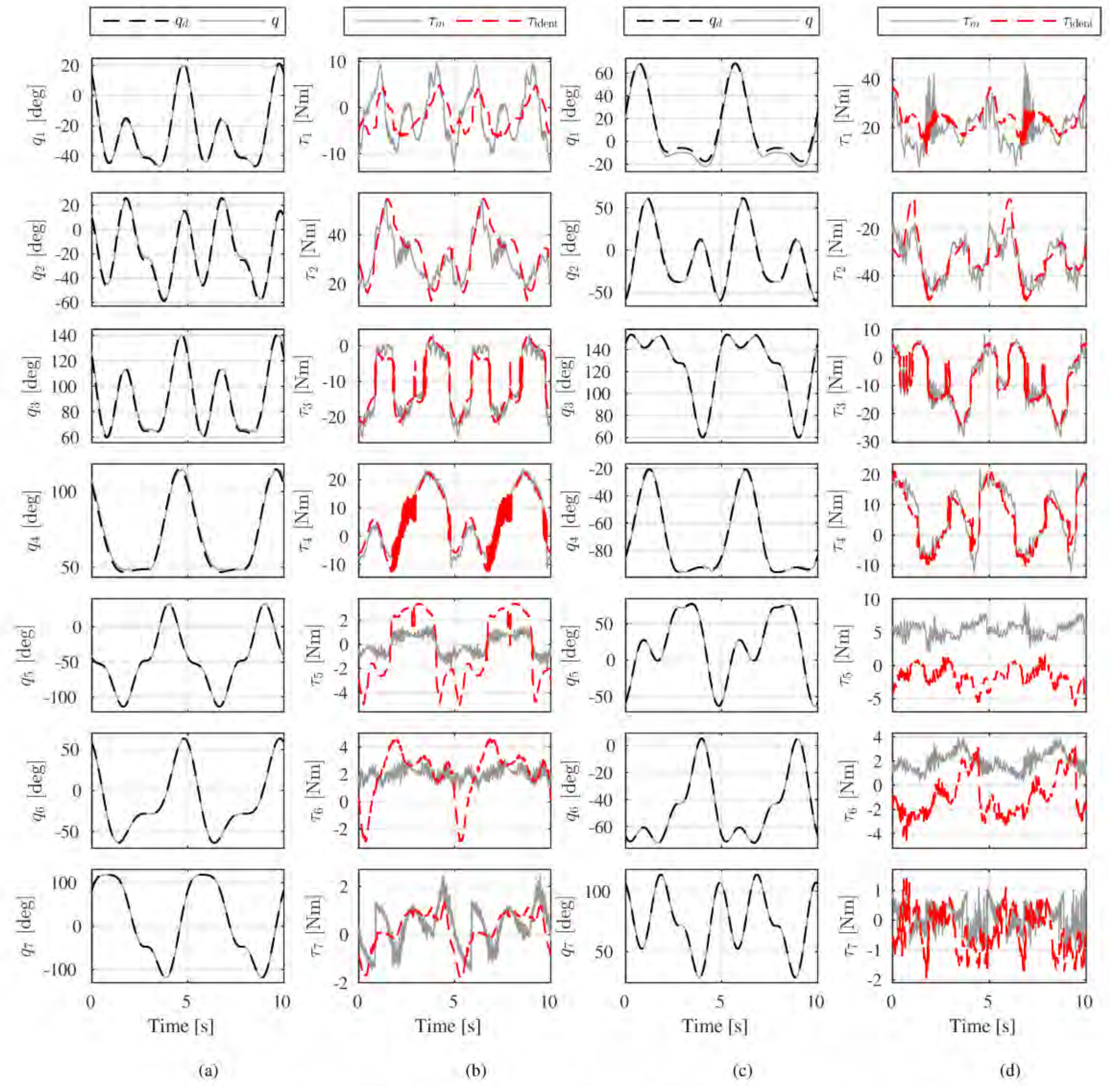


Fig. 4. Position tracking (a) (c), measured and modeled torques (b) (d) of left and right arm for joint 1-7

excitation of the shoulder dynamics within the identification against gravitation influences. The measured torque of the electric joints $5-7$ are not reproduced in a reasonable matter by the model, which however is obvious as the link side torque sensors could not yet be included and the large gear friction thus had large effects.

B. Impedance Control Results

In the following paragraphs the performance of the impedance control is investigated. First, the joint level tracking error is examined for the two cases of feed-forward modeling turned on and off. These experiments are applied under the use of a quasi-static (slow) and a dynamic trajectory. Second, two comparisons between our impedance controller and the PD-position-controller are provided, analyzing joint position and cartesian space errors. Third, the results of the disturbance observer, compensating modeling inaccuracies, are discussed. Finally, typical impedance control characteristics and contact-compliance are demonstrated.

1) Modeling influence on tracking performance: Fig. 5 (a) shows the joint tracking error of joint 2 of the right arm for the two cases of switching the model on and off. A seven joint quasi-static motion was chosen to allow an analyzation of static modeling influences only, allowing to neglect dynamical terms of the model. The controller was set to a low stiffness of 100 Nm/rad to emphasize the effect of the feed-forward model. Further, a desired damping

Preprint submitted to 2015 IEEE-RAS International Conference on Humanoid Robots.
Received July 7, 2015.

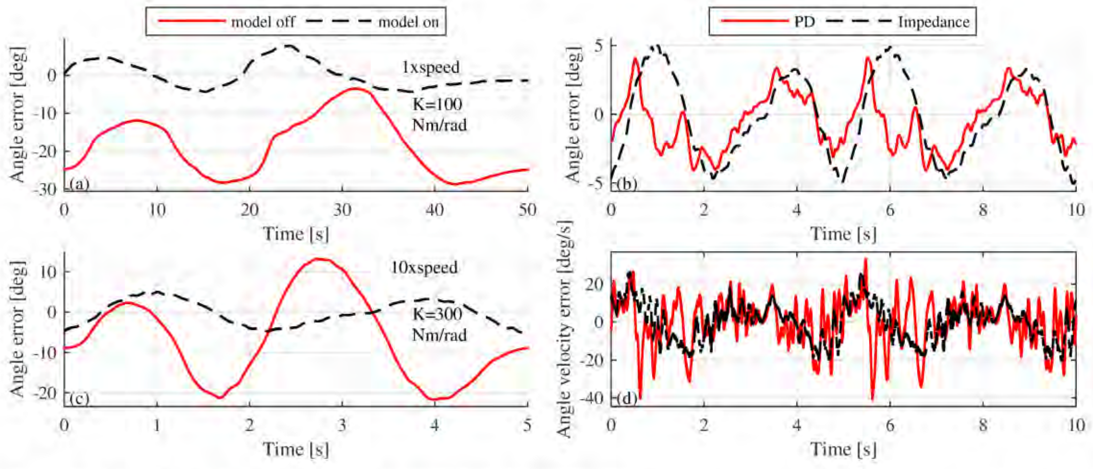


Fig. 5. (a,c) Comparison of tracking error with and without model for dynamic and quasi-static motion (b,d) Comparision between PD-position and impedance control in terms of angular and velocitiy tracking error

TABLE II
LEFT ARM EXPERIMENT: MAXIMUM VELOCITIES & ACCELERATIONS, MEAN & VARIANCE OF ANGLE TRACKING & MODEL

joint	1	2	3	4
$\lvert\dot{q}_{\mathrm{d,max}}\rvert$ [deg/s]	92.5	112.6	114.8	65.7
$\lvert\ddot{q}_{\mathrm{d,max}}\rvert$ [deg/s^2]	261.3	345.3	312.4	125.0
mean $\lvert q - q_d\rvert$[deg]	1.041	2.451	2.192	1.760
variance $(q - q_d)$ [deg]^2	1.382	6.572	5.375	3.836
mean $\lvert\tau - \tau_m\rvert$ [Nm]	4.187	5.152	2.065	2.594
variance $(\tau - \tau_m)$ [Nm]^2	21.760	33.846	6.202	10.647

TABLE III
RIGHT ARM EXPERIMENT: MAXIMUM VELOCITIES & ACCELERATIONS,MEAN & VARIANCE OF ANGLE TRACKING & MODEL

joint	1	2	3	4
$\lvert\dot{q}_{\mathrm{d,max}}\rvert$ [deg/s]	92.1	159.3	119.9	81.3
$\lvert\ddot{q}_{\mathrm{d,max}}\rvert$ [deg/s^2]	222.3	398.1	329.7	171.3
mean $\lvert q - q_d\rvert$[deg]	3.352	2.619	0.934	1.079
variance $(q - q_d)$ [deg]^2	5.353	8.749	1.260	1.660
mean $\lvert\tau - \tau_m\rvert$ [Nm]	4.187	5.152	2.065	2.593
variance $(\tau - \tau_m)$ [Nm]^2	21.760	33.846	6.202	10.647

coefficient of $D_\xi = 0.7$ was applied.

In case of gravity compensation switched off, position errors between -3 deg and -29 deg can be noticed. When switching on the gravitiy compensation under the same circumstances, the position error can be significantly reduced to ± 6 deg.

The influence on the position tracking error for a dynamic trajctory was investigated in Fig. 5 (c). In this experiment the speed of the same trajectory was increased by factor 10 to excite dynamic effects of the model. In contrast to Fig. 5 (a) a higher stiffness of 300 Nm/rad was used to prevent joint limit collisions. Without model-based compensations, the position tracking error lies between -22 deg and 12 deg. Even though the stiffness is three times larger in contrast to Fig. 5 (a), a similar tracking error magnitude can be observed. As a consequence, the influence of dynamics cannot be neglected. Using the model-based compensations, the position error can be reduced by factor four to ± 6 deg. This error lies in the same range in comparison to quasi-static motions (Fig. 5 (a)) despite the higher stiffness values.

2) Comparison between impedance and PD-position control: Fig. 5 (c) and (d) show the position and velocity tracking performance of the impedance controller compared to those of the PD-position controller for joint 2 of the right arm. The impedance controller was used with inverse dynamics and friction compensation, a damping of 0.7 and a stiffness of 300 Nm/rad, without using the disturbance observer. As it can be seen from Fig. 5 (b), the tracking performance regarding the position error of about ± 5 deg lies in the same range for both controllers. It can be exposed in Fig. 5 (d) that the absolute velocity error of the impedance controller (7.6 deg/s) is smaller and contains less vibrations in contrast to the PD-position controller (9.1 deg/s).

Fig. 6 shows the translational position error $e = \|\boldsymbol{p}_{\mathrm{EE}}(\boldsymbol{q}) - \boldsymbol{p}_{\mathrm{EE}}(\boldsymbol{q}_{\mathrm{d}})\|_2$ of the left arm endeffector under the use of a trajectory which is similar to the one applied in Fig. 4. This error is of special interest when thinking of manipulation tasks to obtain a reliable grasping performance. The plot shows the error of the PD-position controller and the impedance controller with a stiffness of 300 Nm/rad and all models from (20) enabled. It has to be considered that the cartesian error was evaluated by internal measurements only. Fig. 6 reveals an average error of ca. 20 mm for the PD-position controller and ca. 19 mm for the impedance controller. Improved performance can be observed when looking at the maximum errors of 35 mm and 58 mm.

3) Disturbance observer: All previously described experiments were applied without the disturbance observer (Eq. 22). Fig. 7 shows the impedance control position error, already presented in Fig. 5 (b), together with online calculated disturbance joint torques $\hat{\boldsymbol{\tau}}_\varepsilon$ from the observer,

Preprint submitted to 2015 IEEE-RAS International Conference on Humanoid Robots.
Received July 7, 2015.

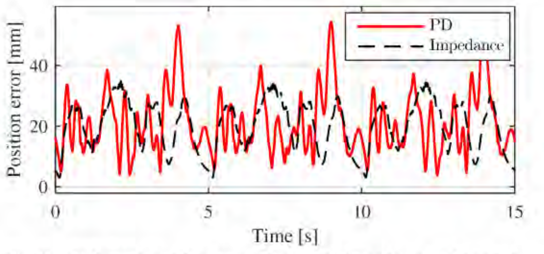


Fig. 6. Endeffector position error (2-norm) of PD-position and impedance controller

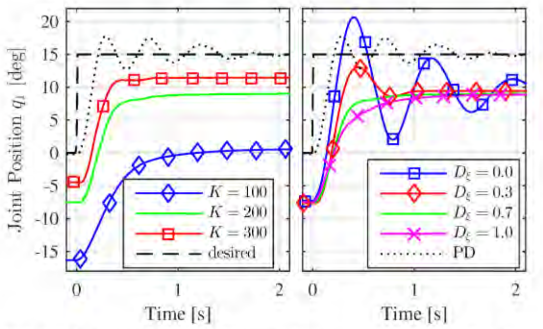


Fig. 8. Step responses with different stiffness settings at $D_\xi = 0.7$ (left); step responses with different damping settings at $K = 200$ Nm/rad (right)

which however were not implemented as feed-forward term. The strong correlation between position error and disturbance torque shows that the disturbance torque is calculated qualitatively correct, assuming low coupling between the arm joints. These promising results show the ability of the observer to compensate for model inaccuracies. Further investigations are planned to also incorporate the disturbance compensation into the operational controller.

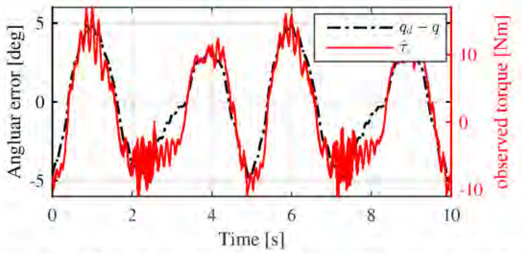


Fig. 7. Observed disturbance torque and position error of impedance controller ($K = 300$ Nm/rad, $D_\xi = 0.7$)

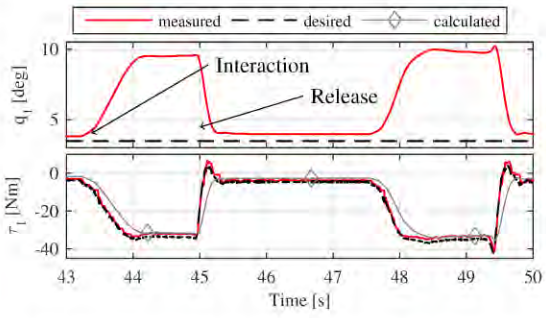


Fig. 9. Compliant manipulation with the impedance controller at $K =$ 300 Nm/rad by human interaction using a rod

4) Step response: Step response experiments were performed to verify typical impedance control characteristics. A desired position step of 15 deg was commanded to the first joint of the right arm with the controller scheme (20) configured with the identified dynamic parameters from Sec. III-A. Fig. 8 (left) shows the step responses for a desired modal damping of $D_\xi = 0.7$ for different desired stiffnesses. Furthermore, the step response of the PD-position controller with fixed gains, enabling a comparison, is shown. The qualitative results show an increasing settling time for increased desired joint stiffnesses. Still uncompensated joint torque measurement offsets of ca. 20 Nm lead to a remaining position offset with decreasing joint stiffness. Assessing this position offset, the achieved joint stiffness is up to 35 Nm/rad too low. Fig. 8 (right) depicts step responses for $K_1 = 200$ Nm/rad and different desired modal damping coefficients. Although the desired damping is achieved qualitatively, the measured damping of the step responses is 0.14 to 0.46 higher than desired. One reason for this aberration is the conservative setting of the friction compensation and the remaining effective joint friction.

5) Compliance: To test the compliance rendering of the impedance controlled arm, a constant desired position was set and the endeffector was pushed with an external force. Fig. 9 shows the measured and desired values of position and torque for the first left arm joint and the calculated joint torque $\boldsymbol{K}(\boldsymbol{q}-\boldsymbol{q}_\mathrm{d})$, resulting from the desired compliant behaviour of the controller. The human interaction can be observed by a deflection from the desired angular position. When the force is released, the arm returns to its inital position. The matching joint torque underlines that the desired compliant behaviour is well achieved.

IV. CONCLUSION

To sum up, even at rather moderate joint stiffness $\boldsymbol{K} =$ diag(300 Nm/rad) the overall position tracking performance of our model-based impedance controller is comparable to that of the tuned PD-position controller, while rendering the correct desired joint compliance at the same time. Certain effects, which we believe to be able to compensate for, limit our current performance:

- In the first right arm shoulder joint, a pressure sensor, which is used for the internal torque loop, reads signifcantly erroneous values. These errors do not affect the position controller, which therefore has a better performance on this joint.
- The proportional controller used for the inner joint torque loop has a steady-state error of up to 4 Nm. This explains the majority of the observed joint position error left in our scheme.
- Complex friction effects [18], [8] may not be reflected in our simplified model.

However, despite the given limitations, our algorithm shows better velocity tracking and thus much smoother less vibratory motions. It also enables compliant manipulation and weight compensation of hands and grasped objects. With according compensation schemes the aforementioned limitations can presumably be overcome and the controller is expected to significantly outperform the reference solution.

ACKNOWLEDGMENT

The authors would like to thank the other members of Team ViGIR for supporting the robot experiments. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Laboratory (AFRL) contract FA8750-12-C-0337.

REFERENCES

[1] A. Ajoudani, J. Lee, A. Rocchi, M. Ferrati, E. M. Hoffman, A. Settimi, N. G. Tsagarakis, D. G. Caldwell, and A. Bicchi. Dual arm impedance control with a compliant humanoid: Application to a valve turning task.

[2] A. Albu-Schäffer, C. Ott, U. Frese, and G. Hirzinger. Cartesian impedance control of redundant robots: recent results with the dlr-light-weight-arms. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 3704–3709 vol.3, Sept 2003.

[3] B. Armstrong, O. Khatib, and J. Burdick. The explicit dynamic model and inertial parameters of the puma 560 arm. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 510–518. IEEE, 1986.

[4] C. G. Atkeson, C. H. An, and J. M. Hollerbach. Estimation of inertial parameters of manipulator loads and links. *The International Journal of Robotics Research*, 5(3):101–119, 1986.

[5] K. Ayusawa, G. Venture, and Y. Nakamura. Identification of humanoid robots dynamics using floating-base motion dynamics. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2854–2859. IEEE, 2008.

[6] J. Craig and M. Raibert. A systematic method for hybrid position/force control of a manipulator. *IEEE Computer Software Applications Conf.*, pages 446–451, 1979.

[7] A. Dietrich, T. Wimböck, and A. Albu-Schäffer. Dynamic whole-body mobile manipulation with a torque controlled humanoid robot via impedance control laws. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3199–3206. IEEE, 2011.

[8] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. DArpino, R. Deits, M. DiCicco, D. Fourie, and et al. An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229254, Oct 2014.

[9] M. Gautier and W. Khalil. Direct calculation of minimum set of inertial parameters of serial robots. *IEEE Transactions on Robotics and Automation*, 6(3):368373, Jun 1990.

[10] M. Gautier and G. Venture. Identification of standard dynamic parameters of robots with positive definite inertia matrix. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 5815–5820. IEEE, 2013.

[11] S. Haddadin. *Towards safe robots : approaching Asimov's 1st law.* PhD thesis, RWTH Aachen, 2011.

[12] N. Hogan. Impedance control: An approach to manipulation, part I - theory. *ASME Journal of Dynamic Systems, Measurement, and Control*, 107:1–7, 1985.

[13] M. A. Hopkins, D. W. Hong, and A. Leonessa. Compliant locomotion using whole-body control and divergent component of motion tracking.

[14] S.-H. Hyon, J. Hale, and G. Cheng. Full-body compliant human-humanoid interaction: Balancing in the presence of unknown external forces. *IEEE Trans. Robot.*, 23(5):884898, Oct 2007.

[15] T. Iwasaki, G. Venture, and E. Yoshida. Identification of the inertial parameters of a humanoid robot using grounded sole link. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 449–454. IEEE, 2012.

[16] H. Kaminaga, J. Ono, Y. Nakashima, and Y. Nakamura. Development of backdrivable hydraulic joint mechanism for knee joint of humanoid robots. *2009 IEEE International Conference on Robotics and Automation*, May 2009.

[17] W. Khalil and E. Dombre. *Modeling, Identification and Control of Robots.* Hermes Penton Science, 2002.

[18] S. Kohlbrecher, A. Romay, A. Stumpf, A. Gupta, O. von Stryk, F. Bacim, D. Bowman, A. Goins, R. Balasubramanian, and D. Conner. Human-robot teaming for rescue missions: Team ViGIRs approach to the 2013 DARPA robotics challenge trials. *Journal of Field Robotics*, 32(3):352–377, 2015. First published online 4 Dec 2014.

[19] O. Kwon and J. H. Park. Asymmetric trajectory generation and impedance control for running of biped robots. *Autonomous Robots*, 26(1):47–78, 2009.

[20] K.-H. Lee and W. S. Newman. Natural admittance control of an electro-hydraulic humanoid robot. 2014.

[21] G. Liu, K. Iagnemma, S. Dubowsky, and G. Morel. A base force/torque sensor approach to robot manipulator inertial parameter estimation. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3316–3321. IEEE, 1998.

[22] M. Mistry, S. Schaal, and K. Yamane. Inertial parameter estimation of floating base humanoid systems using partial force sensing. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 492–497. IEEE, 2009.

[23] Y. Ogawa, G. Venture, and C. Ott. Dynamic parameters identification of a humanoid robot using joint torque sensors and/or contact forces. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 457–462. IEEE, 2014.

[24] C. Ott, A. Albu-Schäffer, A. Kugi, and G. Hirzinger. On the passivity based impedance control of flexible joint robots. *IEEE Transactions on Robotics and Automation*, 2008.

[25] J. H. Park. Impedance control for biped robot locomotion. *Robotics and Automation, IEEE Transactions on*, 17(6):870–882, 2001.

[26] K.-J. Park. Fourier-based optimal excitation trajectories for the dynamic identification of robots. *Robotica*, 24(05):625–633, 2006.

[27] R. Paul and B. Shimano. Compliance and control. In *Joint Automatic Control Conf. (JACC1976), San Francisco, USA*, pages 694–699, 1976.

[28] M. Prats, S. Wieland, T. Asfour, A. P. Del Pobil, and R. Dillmann. Compliant interaction in household environments by the armar-iii humanoid robot. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 475–480. IEEE, 2008.

[29] B. Raucent, G. Campion, G. Bastin, J.-C. Samin, and P. Y. Willems. On the identification of the barycentric parameters of robot manipulators from external measurements. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1169–1174. IEEE, 1991.

[30] J. Salisbury. Active stiffness control of a manipulator in cartesian coordinates. In *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on*, pages 95–100, Dec 1980.

[31] L. Sentis, J. Park, and O. Khatib. Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *Robotics, IEEE Transactions on*, 26(3):483–501, 2010.

[32] C. D. Sousa and R. Corteso. Physical feasibility of robot base inertial parameter identification: A linear matrix inequality approach. *Int. J. Rob. Res.*, 33(6):931–944, May 2014.

[33] S. Traversaro, A. Del Prete, R. Muradore, L. Natale, and F. Nori. Inertial parameter identification including friction and motor dynamics. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pages 68–73. IEEE, 2013.

[34] G. Venture, K. Ayusawa, and Y. Nakamura. Human and humanoid identification from base-link dynamics. In *Biomedical Robotics and Biomechatronics, 2008. BioRob 2008. 2nd IEEE RAS & EMBS International Conference on*, pages 445–450. IEEE, 2008.

[35] D. Whitney. Quasi-static assembly of compliantly supported rigid parts. *ASME J. of Dynamic Systems, Measurement, and Control*, 99:65–77, 1982.

Preprint submitted to 2015 IEEE-RAS International Conference on Humanoid Robots.
Received July 7, 2015.

E. MANIPULATION PLANNING SYSTEM

This appendix provides details on the implementation of the manipulation system. This appendix includes two sections that cover information in more detail, and describe experiments conducted after the DRC Finals. Appendix G-3 includes a technical paper [3] that covers our design through the DRC Trials, and G-4 includes another recent paper [5] submitted to the Humanoids 2015 conference that extends our concept of “usability”.

E.1. Object Template and Usability-based Manipulation

The initial version of the Object Template approach used during the VRC considered only the 3D mesh of the object and potential grasp pose information. With these capabilities we were able to score the “lift fire-hose from the table” point in all of the five runs. However, the lack of manipulation capability in an affordance level such as “turn” required the operator to perform the rotation motions to attach the fire-hose manually, which in a cartesian-space teleoperated approach have high complexity. The results of the VRC can be shown in [2].

After the development phase between the VRC and the DRC Trials, we incorporated additional capabilities to our OT approach. These capabilities included physical information of the object of interest, such as mass and center of mass, which were used for control while manipulating objects (e.g. the drill). Also, we implemented Cartesian and Circular Markers to generate constrained paths for the robot’s end-effectors. These markers are visualized as floating independent frame of references that were manipulated by the human operator and located in a desired pose. Cartesian plans are calculated using the initial end-effectors pose and the origin of the marker as target pose. Circular motions were calculated using individual Cartesian paths around the “X” vector of the marker as rotation axis (see [1] Appendix A).

After the DRC Trials, we evolved our OT to provide the functionality that the Cartesian and Circular markers were providing. With the new OT implementation it was now possible to assign multiple motion constraints into one single frame of reference plus all the previous functionality of the OTs. This brought the concept of affordances to the OT because now we are able to define motions that the object offered [3].

Additionally we developed the concept of “usability.” Usabilities allow the operator to select points of interest in a grasped object so that this point can be used while planning motions. Instead of having one “tool tip” per object, the OTL can describe multiple points in the reference frame of an Object Template. For example, the Drill Template will have at least three usabilities: the origin of the template, the ON/OFF switch, and the bit (see Figure 59). These usabilities allow objects that are grasped by the robot to be considered as online-augmented end-effectors. With this information, affordances can then be executed using these points as reference for motion planning. As shown in Figure 59, the Drill Template (left) has three usabilities: Origin, Trigger, and Bit. The Paint Roller Template (right) has three usabilities: Origin, Base, and Roller. The “bit” in the drill is located around 10 cm above the origin of the reference frame of the Drill Template, for this reason special planning has to be done to achieve the desired cut pattern in the wall (see Figure 58).

As shown in Figure 58, normal planning with respect to the robot hand creates a smaller (dark green) circle about the center axis of rotation of the wall template based on the relative position of the hand (left).

Using the drill bit usability as the reference point results in the correct hand motion pattern to cut the wall, since the drill bit is the one rotating around the axis of the Wall Template (right).

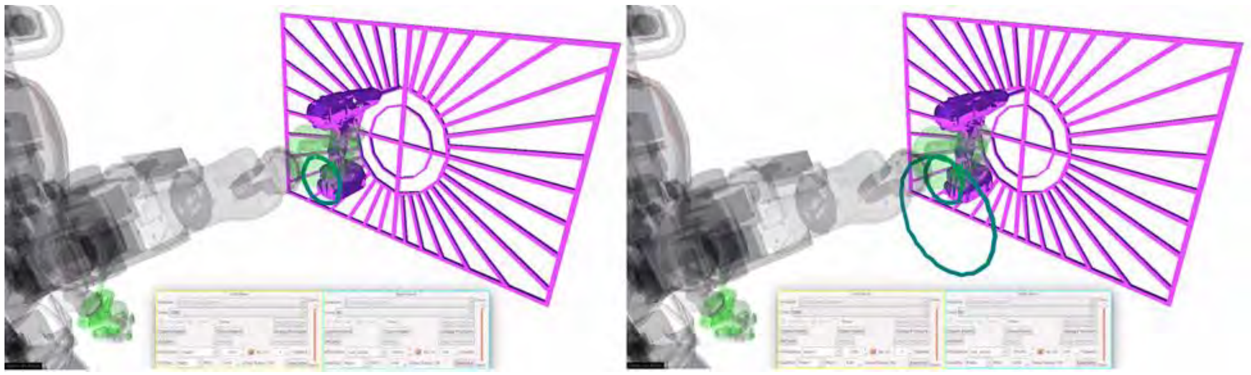


Figure 58. Cut circle in wall with the drill tool.

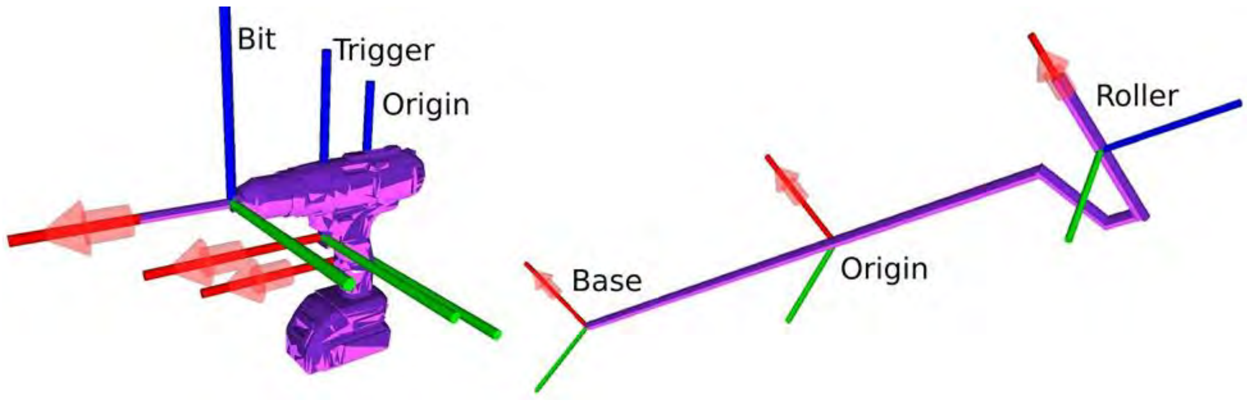
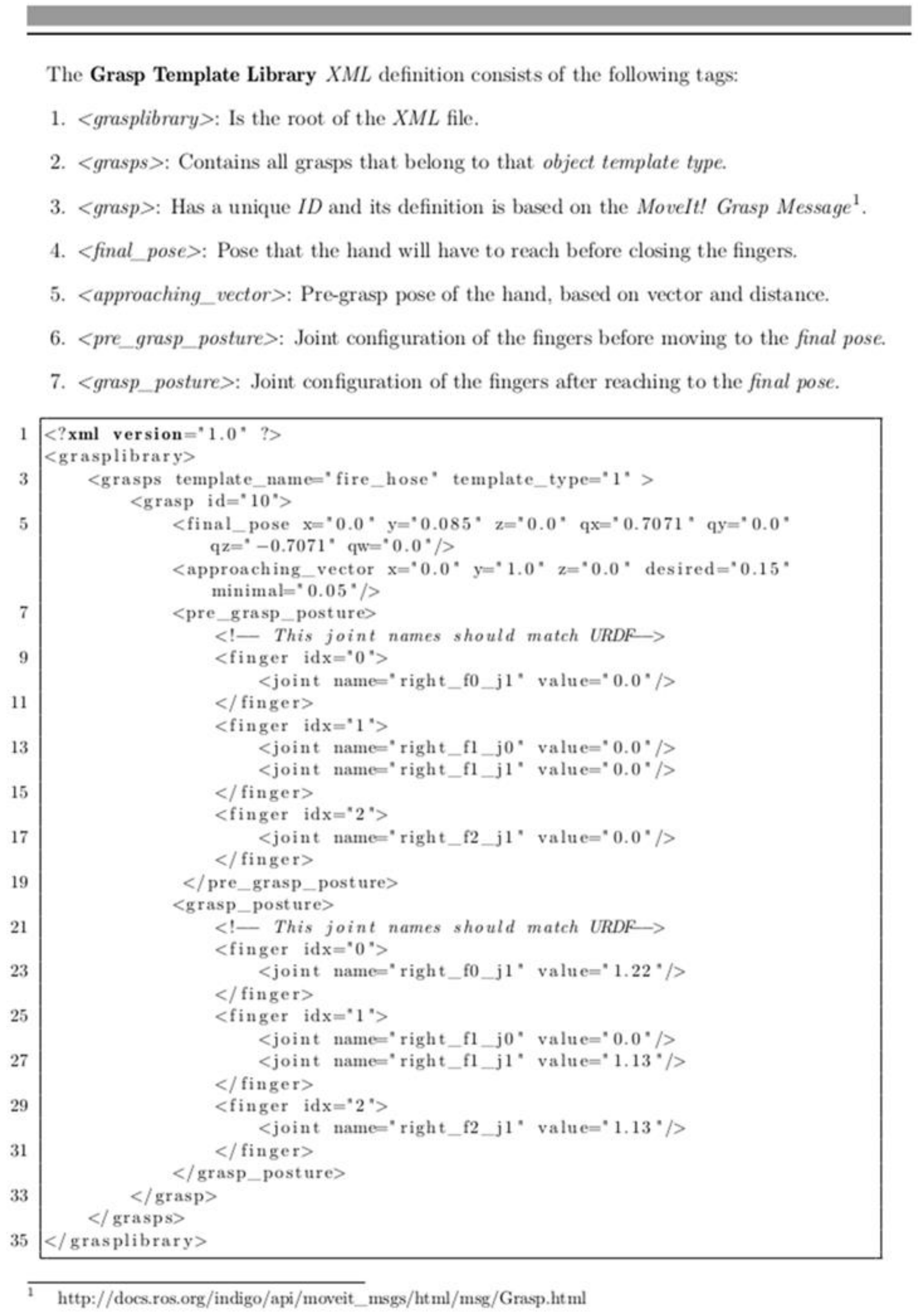


Figure 59. Object usabilities for the drill and paint roller

As described in Section 3.2.4, the Object Template Library is divided in three main groups of information. Here we present example XML flies of each group. The Grasp Template Library, shown in Figure 60, is used to store pre-calculated potential grasp poses for the robot's end-effectors. It also defines the finger postures required for a particular grasp, before closing the fingers and after closing the fingers. The final-grasp is the pose that the end-effector needs to reach before closing the fingers.



The **Grasp Template Library** *XML* definition consists of the following tags:

1. <*grasplibrary*>: Is the root of the *XML* file.
2. <*grasps*>: Contains all grasps that belong to that *object template type*.
3. <*grasp*>: Has a unique *ID* and its definition is based on the *MoveIt! Grasp Message*[1].
4. <*final_pose*>: Pose that the hand will have to reach before closing the fingers.
5. <*approaching_vector*>: Pre-grasp pose of the hand, based on vector and distance.
6. <*pre_grasp_posture*>: Joint configuration of the fingers before moving to the *final pose*.
7. <*grasp_posture*>: Joint configuration of the fingers after reaching to the *final pose*.

```
<?xml version="1.0" ?>
<grasplibrary>
    <grasps template_name="fire_hose" template_type="1" >
        <grasp id="10">
            <final_pose x="0.0" y="0.085" z="0.0" qx="0.7071" qy="0.0"
                qz="-0.7071" qw="0.0"/>
            <approaching_vector x="0.0" y="1.0" z="0.0" desired="0.15"
                minimal="0.05"/>
            <pre_grasp_posture>
                <!-- This joint names should match URDF-->
                <finger idx="0">
                    <joint name="right_f0_j1" value="0.0"/>
                </finger>
                <finger idx="1">
                    <joint name="right_f1_j0" value="0.0"/>
                    <joint name="right_f1_j1" value="0.0"/>
                </finger>
                <finger idx="2">
                    <joint name="right_f2_j1" value="0.0"/>
                </finger>
             </pre_grasp_posture>
            <grasp_posture>
                <!-- This joint names should match URDF-->
                <finger idx="0">
                    <joint name="right_f0_j1" value="1.22"/>
                </finger>
                <finger idx="1">
                    <joint name="right_f1_j0" value="0.0"/>
                    <joint name="right_f1_j1" value="1.13"/>
                </finger>
                <finger idx="2">
                    <joint name="right_f2_j1" value="1.13"/>
                </finger>
            </grasp_posture>
        </grasp>
    </grasps>
</grasplibrary>
```

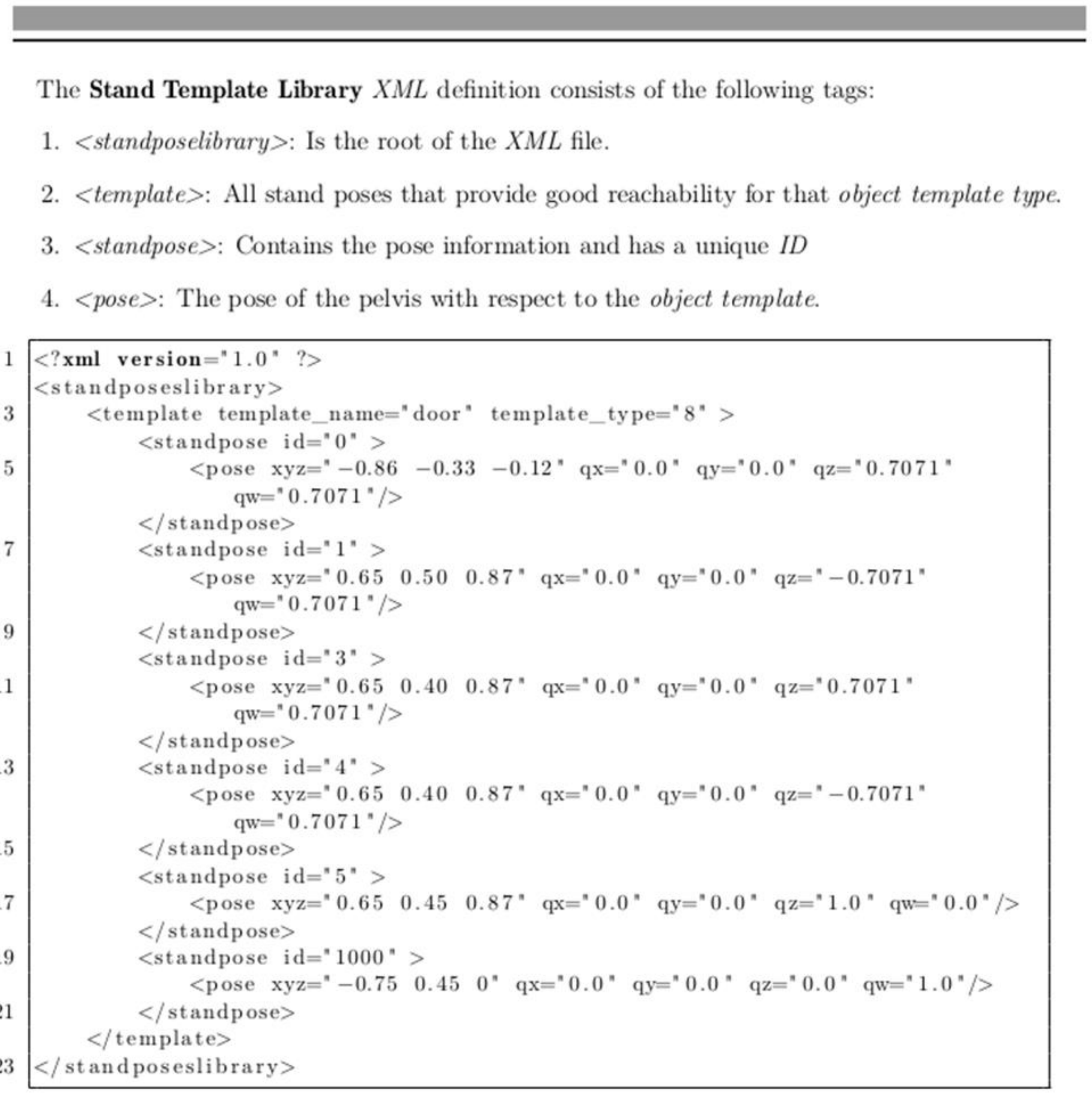
[1] http://docs.ros.org/indigo/api/moveit_msgs/html/msg/Grasp.html

Figure 60. Grasp Template Library XML file

An approaching_vector is defined in a way that the end-effector can safely reach a pose near the object.

After reaching this "pre-grasp" pose, the end-effector only needs to move in the direction of the approaching_vector to reach the final-pose. Each grasp has its own ID and they are linked to one single template_type.

Another issue to be tackled was the determination of suitable stand poses for manipulation relative to a given object. An inverse reachability approach, available as open source as part of the Simox library[48] was integrated with Team ViGIR's software for this purpose. Prior knowledge about DRC tasks made the use of this automated inverse reachability system and the added complexity introduced by it unnecessary. To simplify usage, Team ViGIR used the Stand Template Library, shown in Figure 61, to store pre-calculated stance poses for the robot pelvis that will allow the robot to properly reach the object. It is a six degree of freedom pose of the robot's pelvis with respect to the OT frame of reference. Each stand pose has its own ID and they are linked to one single template_type. For use within real disaster environments, a fully integrated inverse reachability approach that considers possible collisions with the environment, biped balance constraints, and of sensor visibility is desireable.



The **Stand Template Library** *XML* definition consists of the following tags:

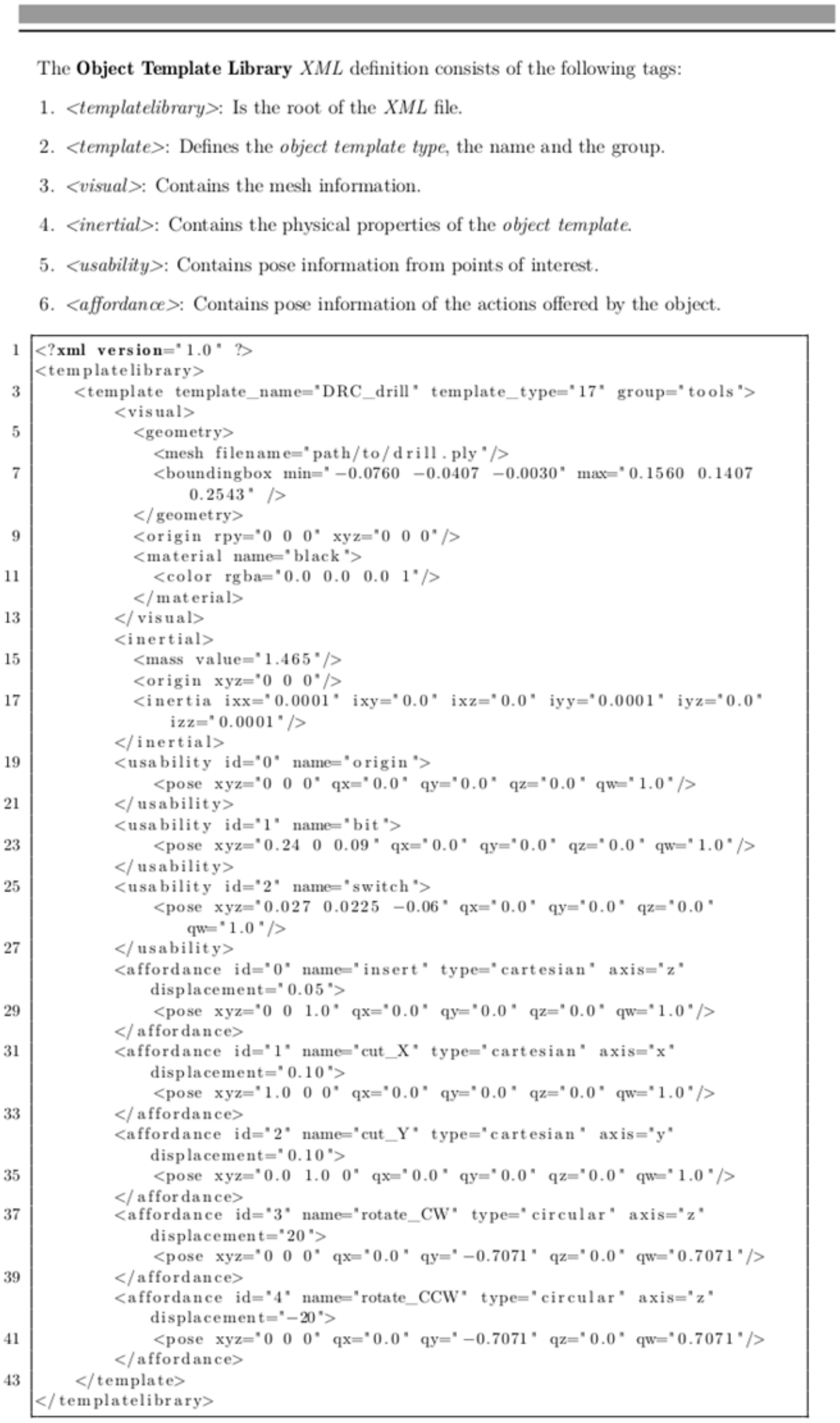
1. *<standposelibrary>*: Is the root of the *XML* file.
2. *<template>*: All stand poses that provide good reachability for that *object template type*.
3. *<standpose>*: Contains the pose information and has a unique *ID*
4. *<pose>*: The pose of the pelvis with respect to the *object template*.

```
<?xml version="1.0" ?>
<standposeslibrary>
    <template template_name="door" template_type="8" >
        <standpose id="0" >
            <pose xyz="-0.86 -0.33 -0.12" qx="0.0" qy="0.0" qz="0.7071"
               qw="0.7071"/>
        </standpose>
        <standpose id="1" >
            <pose xyz="0.65 0.50 0.87" qx="0.0" qy="0.0" qz="-0.7071"
               qw="0.7071"/>
        </standpose>
        <standpose id="3" >
            <pose xyz="0.65 0.40 0.87" qx="0.0" qy="0.0" qz="0.7071"
               qw="0.7071"/>
        </standpose>
        <standpose id="4" >
            <pose xyz="0.65 0.40 0.87" qx="0.0" qy="0.0" qz="-0.7071"
               qw="0.7071"/>
        </standpose>
        <standpose id="5" >
            <pose xyz="0.65 0.45 0.87" qx="0.0" qy="0.0" qz="1.0" qw="0.0"/>
        </standpose>
        <standpose id="1000" >
            <pose xyz="-0.75 0.45 0" qx="0.0" qy="0.0" qz="0.0" qw="1.0"/>
        </standpose>
    </template>
</standposeslibrary>
```

Figure 61. Stand Template Library XML file

[48] Vahrenkamp, Nikolaus, Tamim Asfour, and Rudiger Dillmann. "Robot placement based on reachability inversion." *Robotics and Automation (ICRA), 2013 IEEE International Conference on* 6 May. 2013: 1970-1975.

The Object Template Library, shown in Figure 62, contains the physical information of the real object it represents. It has also 3D mesh information of the shape that can be linked with a path to a PLY mesh file. The OTL also contains the semantic information of the object in the way of affordances and usabilities. The template_type is used to relate information of a template to the Grasp Library and the Stand Library.



The **Object Template Library** *XML* definition consists of the following tags:

1. *<templatelibrary>*: Is the root of the *XML* file.
2. *<template>*: Defines the *object template type*, the name and the group.
3. *<visual>*: Contains the mesh information.
4. *<inertial>*: Contains the physical properties of the *object template*.
5. *<usability>*: Contains pose information from points of interest.
6. *<affordance>*: Contains pose information of the actions offered by the object.

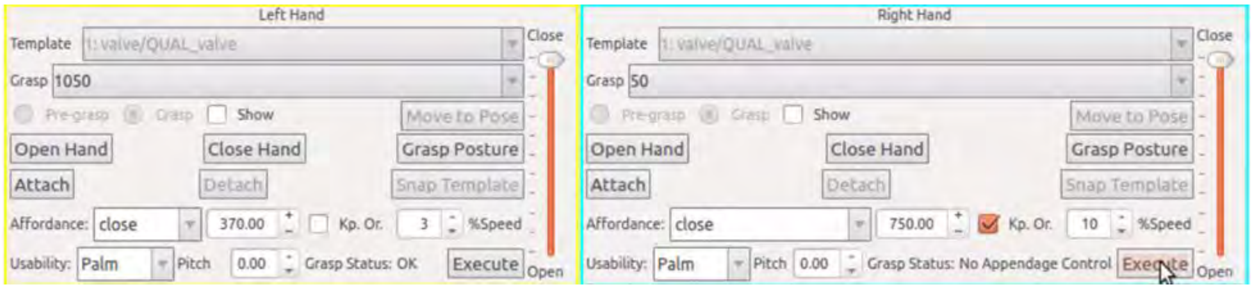
```xml
<?xml version="1.0" ?>
<templatelibrary>
    <template template_name="DRC_drill" template_type="17" group="tools">
        <visual>
          <geometry>
            <mesh filename="path/to/drill.ply"/>
            <boundingbox min="-0.0760 -0.0407 -0.0030" max="0.1560 0.1407
                0.2543" />
          </geometry>
          <origin rpy="0 0 0" xyz="0 0 0"/>
          <material name="black">
            <color rgba="0.0 0.0 0.0 1"/>
          </material>
        </visual>
        <inertial>
          <mass value="1.465"/>
          <origin xyz="0 0 0"/>
          <inertia ixx="0.0001" ixy="0.0" ixz="0.0" iyy="0.0001" iyz="0.0"
              izz="0.0001"/>
        </inertial>
        <usability id="0" name="origin">
            <pose xyz="0 0 0" qx="0.0" qy="0.0" qz="0.0" qw="1.0"/>
        </usability>
        <usability id="1" name="bit">
            <pose xyz="0.24 0 0.09" qx="0.0" qy="0.0" qz="0.0" qw="1.0"/>
        </usability>
        <usability id="2" name="switch">
            <pose xyz="0.027 0.0225 -0.06" qx="0.0" qy="0.0" qz="0.0"
                qw="1.0"/>
        </usability>
        <affordance id="0" name="insert" type="cartesian" axis="z"
            displacement="0.05">
            <pose xyz="0 0 1.0" qx="0.0" qy="0.0" qz="0.0" qw="1.0"/>
        </affordance>
        <affordance id="1" name="cut_X" type="cartesian" axis="x"
            displacement="0.10">
            <pose xyz="1.0 0 0" qx="0.0" qy="0.0" qz="0.0" qw="1.0"/>
        </affordance>
        <affordance id="2" name="cut_Y" type="cartesian" axis="y"
            displacement="0.10">
            <pose xyz="0.0 1.0 0" qx="0.0" qy="0.0" qz="0.0" qw="1.0"/>
        </affordance>
        <affordance id="3" name="rotate_CW" type="circular" axis="z"
            displacement="20">
            <pose xyz="0 0 0" qx="0.0" qy="-0.7071" qz="0.0" qw="0.7071"/>
        </affordance>
        <affordance id="4" name="rotate_CCW" type="circular" axis="z"
            displacement="-20">
            <pose xyz="0 0 0" qx="0.0" qy="-0.7071" qz="0.0" qw="0.7071"/>
        </affordance>
    </template>
</templatelibrary>
```

Figure 62. Object Template Library XML file

E.1.1. Manipulation Control Widget

The user interface used to interact with the remote robots consist of a manipulation widget for each hand (see Figure 63). This widget is access from the Main UI window presented in Section 3.1.3.1. This widget is responsible of providing to the human operator all the functionalities that the OT approach has.

Once an OT is inserted in the environment, the operator can double click that OT to let know the



Left Hand
Template 1: valve/QUAL_valve
Grasp 1050
Affordance: close 370.00 Kp. Or. 3 %Speed
Usability: Palm Pitch 0.00 Grasp Status: OK

Right Hand
Template 1: valve/QUAL_valve
Grasp 50
Affordance: close 750.00 ✓ Kp. Or. 10 %Speed
Usability: Palm Pitch 0.00 Grasp Status: No Appendage Control

Figure 63. Manipulation Control Widgets for each Hand.

Manipulation Widget that that is the OT of interest. The Manipulation Widget then displays all the information available for this OT (see. Figure 64). The pre-grasp and final grasp poses for a specific Grasp Template can be shown. The fingers can be Opened, Closed, set to the specific joint configuration defined for that Grasp, and there is the possibility to select the percentage of closure if the fingers are going to be manually controlled. If the object is going to be moved around the environment, the operator can "Attach" the OT to the robot, allowing the motion planner to consider the real object for collision avoidance, in the same way the OT can be detached from the robot. The Usability combo box allows the operator to select the frame of reference in the end-effector that the motion planning is going to be done with respect to (e.g. Palm, Poke Stick, the origin of the template, or any point of interest included as a usability in the OTL). Affordances can be executed with different parameters. Once the affordance is selected from the combo box, the default values for that affordance are automatically loaded, afterwards the operator can change this parameters. The displacement parameters use degrees for rotational motions and meters for translational motions. The operator can also select if the motion is going to be performed keeping the end-effector orientation or not. In case the affordance is rotational, the operator can give a pitch to that affordance to convert the circular motion into a "spiral" motion. Finally, the speed of the motion execution can also be set.

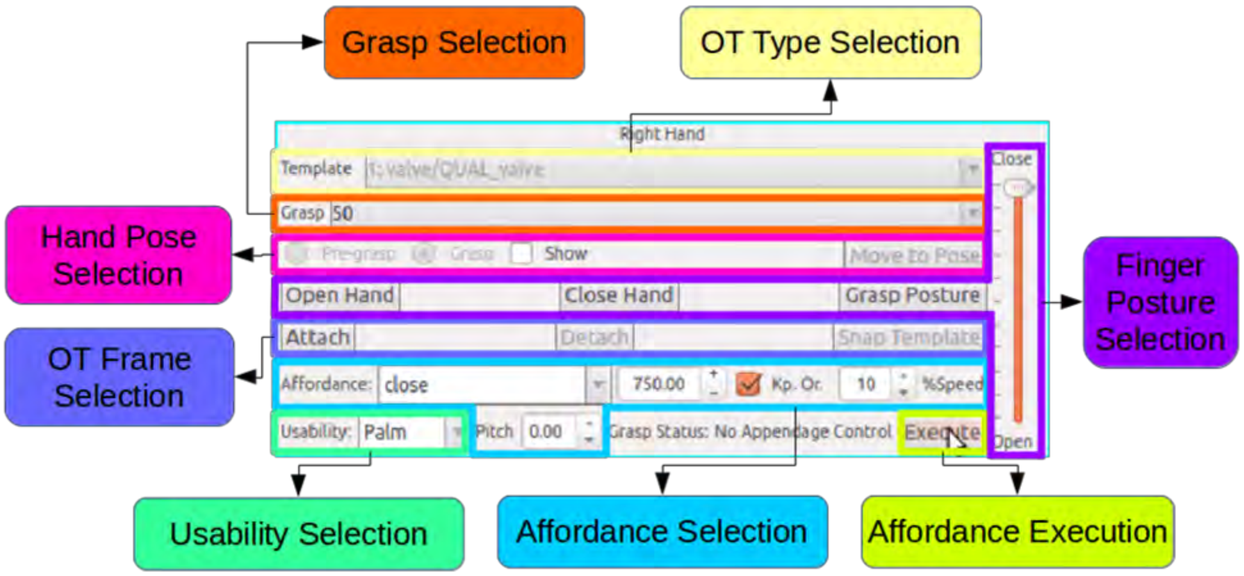


Figure 64. Description of Manipulation Widget functions that interact with Object Templates (OT).

E.1.2. Transfer of Manipulation Skills between Objects

During some practice tests, we found ourselves using a different OT than the one that was designed for that task. For example, while turning the steering wheel of the Polaris vehicle, we initially used the Valve Template before creating the Steering Wheel template. This is possible given that motions required to perform a manipulation task do not depend on how and where the robot has grasped the object. In a recently submitted to paper [5], we present an approach that shows how the operator can use an OT to perform versatile manipulation tasks. This is demonstrated during an experiment where the robot is not able to reach a valve because the stand position required is blocked by debris. A combination of two DRC tasks was created and the use of OT allows the operator, for example, to pick up a piece of debris and utilize it to reach and turn the valve [Appendix Experiments].

E.1.3. Object Template Alignment

It is a known disadvantage, as shown during experiments with behaviors in Blacksburg, that manual alignment of OT consumes most of the time during a manipulation task. Initially Team ViGIR during the VRC, and later on in collaboration with Team VALOR during the DRC Finals, attempted to develop automatic OT matching algorithms to match the 3D mesh to the perceived sensor data to determine the 6D pose of the object. Test results of automatic OT alignment to the sensor data corresponding to the real object were not robust enough, and had too many corner cases. During the competition and the experiments, auxiliary operator manually aided in performing object identification and alignment of the OT to the sensor data.

E.2. Manipulation Experiments

To validate the theoretical concepts described in Section 3.2.4, we performed some experiments that demonstrate how manipulation tasks can be efficiently performed by the human operator using the Object Template approach. Appendix H contains experiments that show how the same usabilities and affordances can be incorporated into autonomous behaviors. A playlist with all experiment videos can be found here: https://www.youtube.com/playlist?list=PL7v9EfkjLswJQn2yZE3qv5sECx-qZbeXO .

E.2.1. Wall Task

The wall task is considered the most challenging manipulation task in the DRC. It requires object manipulation, interacting with small object parts such as the ON/OFF switch, and planning with environmental constraints such as the wall plane and the region that needs to be cut.

This experiment shows how the human operator using the Manipulation Widget commands the robot to pick up the drill and draw a circle in the wall. In this case, the task requires to perform motion planning in two different frame of references at the same time: the wall and the drill. On one side, the Cut-Circle affordance of the wall needs to be used to generate a circular motion around the frame of reference of the wall. On the other side, the robot needs to calculate the path to follow, not with respect to the hand, but with respect to the drill bit. This is a perfect example where the operator can use the *affordances* of the wall while selecting and planning with respect to the drill bit *usability*. In Figure 65 and the associated video[49], we used a marker in place of the drill bit to demonstrate the path.

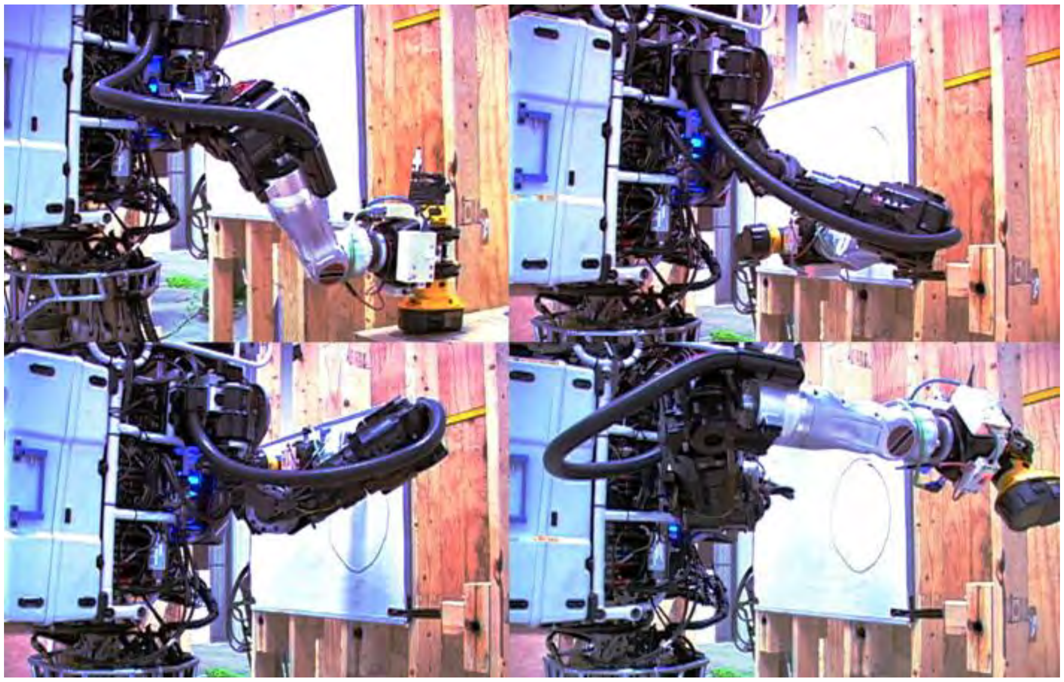


Figure 65. Drawing a circle using affordances defined in the Wall and Drill Object Templates.

Upper left: Picking up drill. Upper right: Using "Insert" affordance of the drill. Lower left: Using "Cut-Circle" affordance of the wall. Lower right: Circle completed.

[49] https://www.google.com/url?q=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3D4aIpvpqwvJY

E.2.2. Cord Plug Surprise Task

From the 3 surprise tasks, the Cord Plug task was the most challenging because of the accuracy required to introduce the cord plug into the socket. While we did not get to attempt the Cord Plug Task on Day 2 of the DRC Finals due to hardware issues, we demonstrated this task in experiments.

During this experiment, we performed the Cord Plug task in around 3 minutes. Using the Manipulation Widget, the operator can easily send the robot's hand to pre-grasp and final grasp positions for both sockets, the operator only needs requiring only to use afterwards the "extract" and "insert" affordances of the socket. Given inaccuracies while grasping the cord plug, the pre-calculated insert positions of the socket are not aligned. However, after minimal alignment from the operator, the "insert" affordance of the socket can be used. Since this affordance only describes the the motion of the hand needs to be parallel to the axis of insertion of the socket, the orientation of the hand is not relevant to perform the manipulation motion of insertion (see Figure 66 and video[50]).

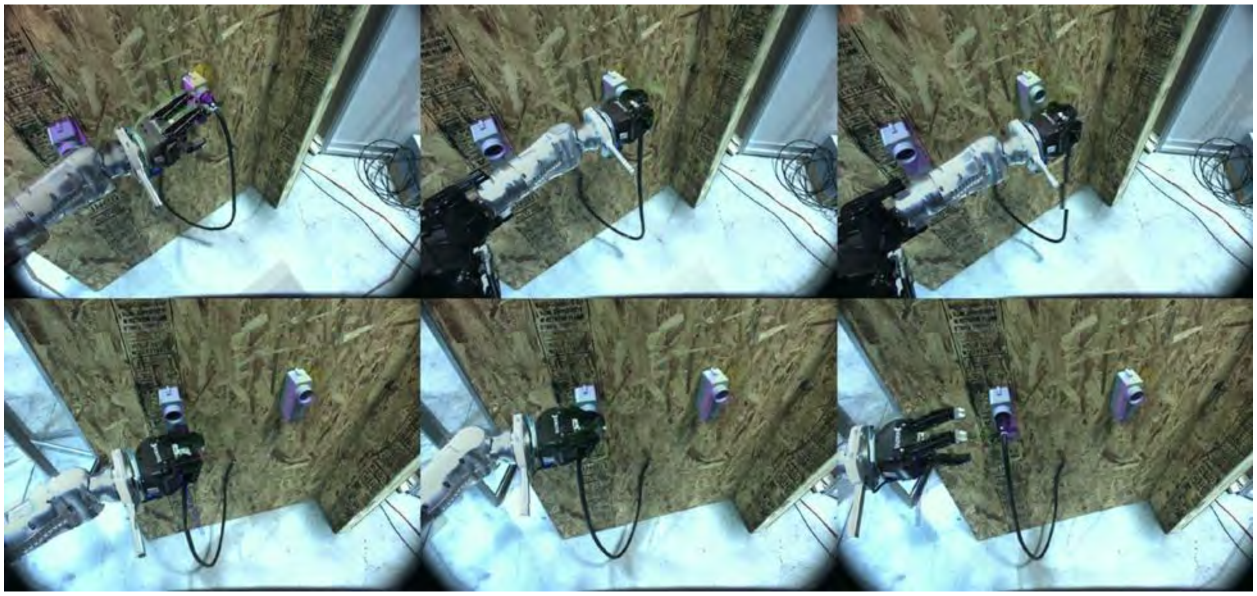


Figure 66. Cord Plug Surprise Task Demonstration

From top-left to bottom-right: Pre-grasp, grasp, extract, pre-insert, insert, release.

E.2.3. Robustness Experiments

After the DRC, Team ViGIR continued performing experiments with the Atlas robot. While some of the experiments were a repetition of the DRC tasks, we tested the robustness of our approach for cases where the robot is not able to reach the objects of interest (situation that can easily happen in a post-disaster scenario).

As described in Section 3.2.4, the manipulation skills that the affordances provide are grasp-agnostic. That said, we envisioned a disaster scenario similar to a combination of the Valve Task and the Debris Task in the DRC. In this scenario, access to the valve is blocked by debris. Normally, the robot would have first to remove all debris until it gains access to the valve, and then perform the turning motion.

[50] https://www.google.com/url?q=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DtduZBtkuDWM

However, if the case is that the debris cannot be removed completely (e.g., it is heavy or big), then the task would be impossible complete. To demonstrate how the OT approach can allow the operator to improvise, provide the following experiment.

The operator identifies a piece of debris that can be used to reach and turn the valve. The operator performs the required manipulation motions to pick up a stick from the debris (just like in the Debris Task) but it then uses this stick to turn the valve by inserting the edge within the crossbars of the valve. Once the stick is in place, without any modification to the approach, the operator can then execute the turn affordance of the valve, and the required circular motions to turn the valve will be done using the stick (see Figure 67 and the associated video[51]).

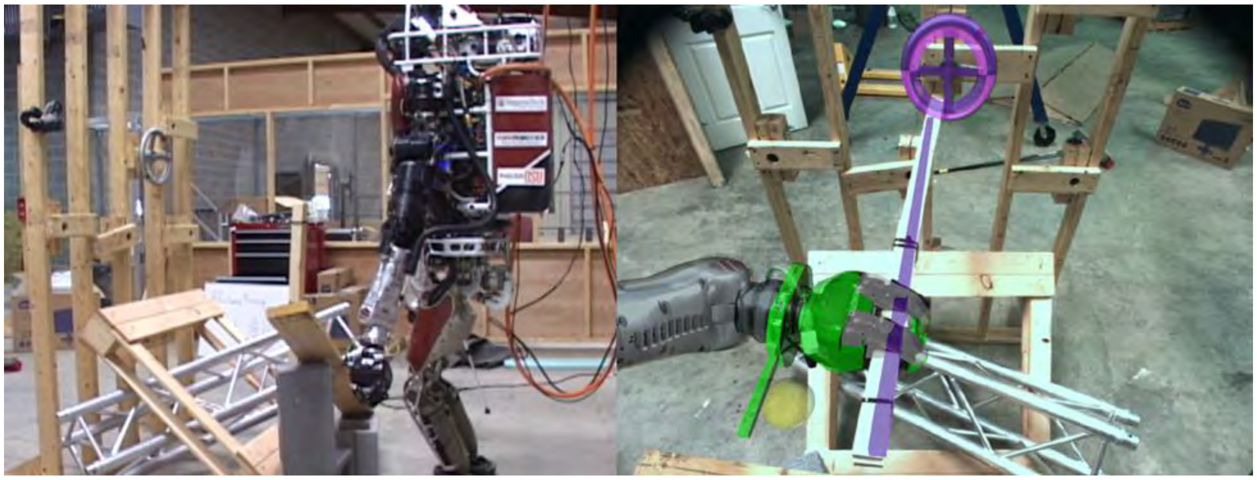


Figure 67. Atlas using a stick to turn the valve

Atlas is unable to reach a valve because of debris blocking the stand pose needed to grasp the valve with the hands (left). The human operator identifies a stick among the debris and uses it to reach the valve (right). The turning affordance of the valve is used in the same way when grasping the valve with the hands as when having a stick inserted within the cross bar of the valve.

In another experiment, Atlas is unable to turn a valve because it is in a higher place than it can reach. The human operator identifies a long L-shaped stick (e.g. paint roller) which can be grasped and used to reach the valve. This experiment is different from the previous one, because in this case, the point that needs to follow a circular path around the valve is not located in the end-effector, but in the "roller" part of the object. To plan with respect to a point of interests in the grasped object, the operator can select the "usability" that belongs to that point (in this case is the "roller" usability). With this online-augmented end-effector, the turning affordance of the valve can then be used in the same way as when turning the valve with the hands (see Figure 68 and associated video[52]).

[51] https://www.youtube.com/watch?v=HN8PEf4ftmU (accessed August 19, 2015)
[52] https://www.youtube.com/watch?v=4km_aaatA0M (accessed August 19, 2015)

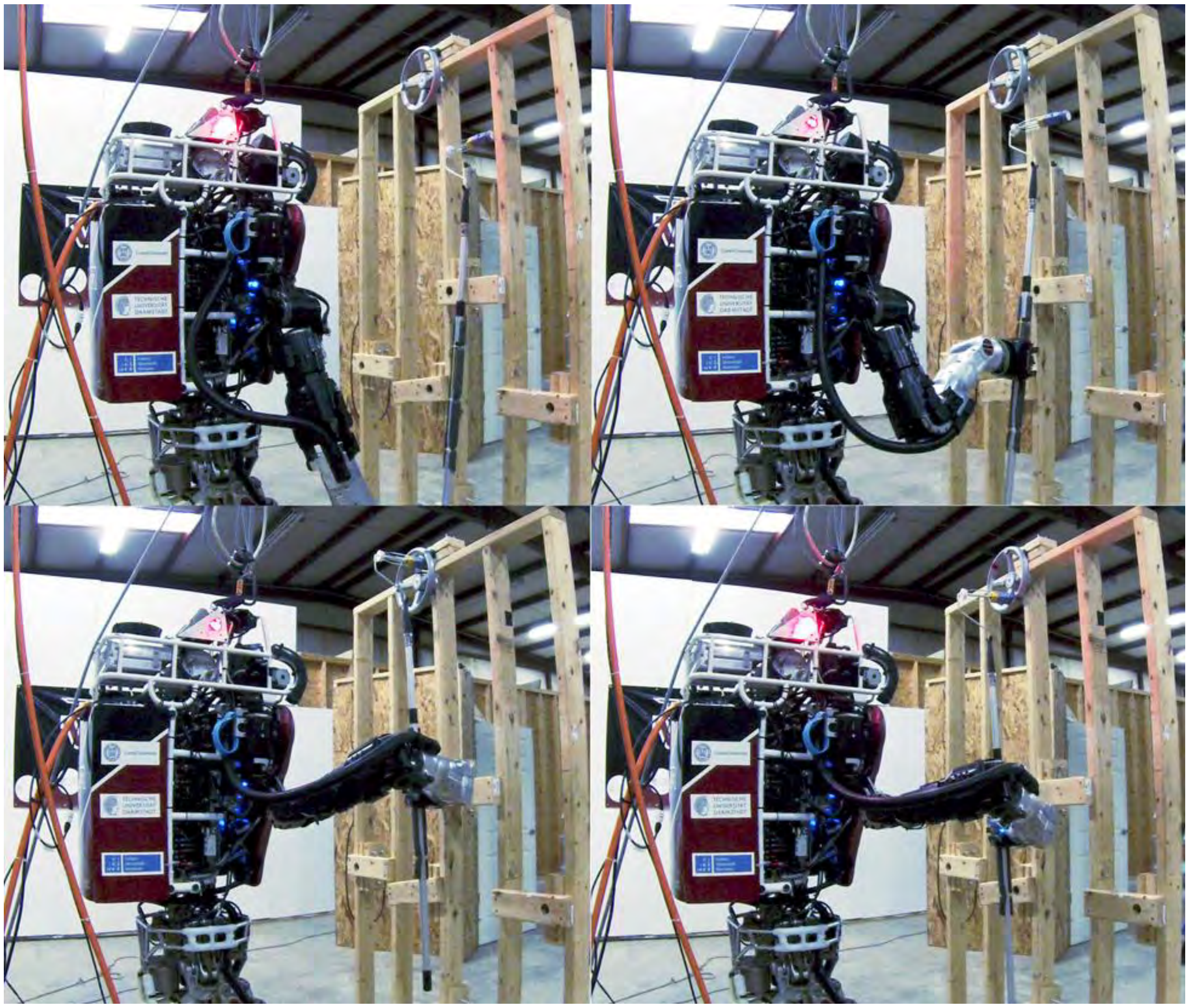


Figure 68. Atlas turning a high non-reachable valve using a paint roller

This page intentionally blank.

E.3. Humanoids 2014 Paper on Manipulation [3]

Template-Based Manipulation in Unstructured Environments for Supervised Semi-Autonomous Humanoid Robots

Alberto Romay*, Stefan Kohlbrecher*, David C. Conner†, Alexander Stumpf* and Oskar von Stryk*

***Abstract*— Humanoid robotic manipulation in unstructured environments is a challenging problem. Limited perception, communications and environmental constraints present challenges that prevent fully autonomous or purely teleoperated robots from reliably interacting with their environment. In order to achieve higher reliability in manipulation we present an approach involving remote human supervision. Strengths from both human operator and humanoid robot are leveraged through a user interface that allows the operator to perceive the remote environment through an aggregated worldmodel based on onboard sensing, while the robot can efficiently receive perceptual and semantic information from the operator. A template based manipulation approach has been successfully applied to the Atlas humanoid robot; we show real world footage of the results obtained in the DARPA Robotics Challenge Trials 2013.**

I. INTRODUCTION

Autonomous and teleoperated manipulation using humanoid robots are still complex problems. While challenges such as object recognition, self-localization and obstacle avoidance have to be tackled for autonomous systems, purely teleoperated robots require dealing with communication constraints such as loss of information and latency as well as providing the proper feedback and situational awareness to the operator.

If we consider a fully autonomous robot navigating and interacting in an unconstrained environment, the capabilities of this robot should include extensive databases of information about possible objects of interest to be found, highly efficient grasping algorithms and the ability to react to unforeseen situations, which are still unsolved problems. On the other hand, if we consider a purely teleoperated robot, the capabilities of the robot and the operator should include near real-time feedback without disruptions in the communications as well as transmission of large amounts of data to the operator. The performance of teleoperated robots also strongly depends on the experience and ability of the human operator to interact with the environment.

Now, as a third option, if we consider a human-robot cooperation that implements the strengths from both, the challenges of either fully autonomous and purely teleoperated robots could be tackled.

A. Related Work

Fully autonomous robots have the advantage of not needing communications and independence of robot control from

*Department of Computer Science, TU Darmstadt, Germany romay,kohlbrecher,stumpf,stryk@sim.tu-darmstadt.de
†TORC Robotics, VA, USA conner@torcrobotics.com

(a) Atlas

(b) iRobot Hand

(c) BDI Hook

Fig. 1: a) Atlas robot with 28 hydraulically actuated DOF, stands 1.88 m tall and weighs approximately 150 Kg. b) iRobot Hand [17], a 3 fingered robotic hand with 5 DOF and c) a metal hook developed by BDI.

an operator. While a lot of research has been performed for autonomous humanoid robots in structured environments with impressive results [16], [21], fewer results have been obtained for humanoid robots in unstructured environments. Full autonomy presents challenges such as object recognition and mission planning, which given the conditions that can be encountered in unstructured environments, make fully autonomous robots not yet feasible to perform these tasks efficiently.

To try to overcome the challenges of autonomously identifying objects and understanding how these objects can be grasped and used, some researchers have studied the concept of affordances. The term "affordances" was first introduced by the psychologist J.J. Gibson in [6]. An affordance, from the perspective of Psychology, is a term that describes the possible actions that an object offers to an organism in the environment. This concept has been adopted by research fields related to robotics as an approach to define how a robot should behave in order to accomplish a determined task using an object. It is the relationship between an object and how a robot should manipulate this object. The extensive work of Şahin et al. [20] to formalize the term of affordance in the robotics field has been key to the development of robot control approaches [1], [25]. Kruger et al. [11] proposed Object-Action Complexes (OACs), which aim at defining the relationships between objects and actions. OACs is a concept created to formalize this relationship and it has been used to develop control approaches that aim at allowing robots with the possibility to understand how an object will behave after an action is performed over it. OACs allow autonomous robots to learn and predict the behaviours derived from performing an action over an object as well as to build symbolic representations of continuous sensorimotor experience.

The work of Nagatani et al. [15], [14] showed that the use of teleoperated robots can help obtaining information of hazardous environments. In their project, they use a wired network to overcome communication issues, but this also leads to limited exploration range and risk of entanglement. During the mission of the robot Quince to the Fukushima nuclear plant in 2011, the access to the third floor was blocked by rubble; semi-autonomous manipulation of such unstructured objects could have led to continued exploration.

The approach presented in [5] is similar to ours. The differences are the way in which we define the possible actions to be executed over an object and how we interact with them. In their approach, for example, they developed automatic template fitting algorithms which are faster than human assisted alignment. While in our approach, motion planning happens onboard the robot, they create the motion plans in their control station and send them afterwards. Another difference is the way we approach the term affordance. For example, to use the open-ability and close-ability of a valve, in their approach, the operator manually rotates the template of the valve to create a motion plan, while in our approach, the operator specifies a semantic action such as "open" or "close" using a number like "± 360 degrees".

B. Overview

Humans can work in a highly abstract, discrete space, having the knowledge and the perception capacity to easily identify and classify objects of interest (by their semantic properties), as well as decision making to accomplish a task. Robots have the capacity to perform calculations on a continuous space estimating objects' physical properties (e.g. mass and inertia). Mixing the strengths of both, humans can in principle aid a remote semi-autonomous humanoid robot to perform rescue tasks in environments that are dangerous for humans. In order to command a robot through a remote environment, we make use of a graphic user interface or Operator Control Station (OCS), where a human operator can use a 3D visualization of the remote environment through the information provided by the robot's sensors.

To be able to provide the robot with information about the environment it perceives, we created an approach that incorporates physical and semantic information about the objects of interest into a template that is sent to the robot through the OCS. This object template concept is inspired by the theory of affordances [6] and OACs [11] (see Section I-A). The theory of affordances and OACs define the concept of "object" as an entity of the environment. However, for the purposes of our approach, we created the concept of "object template" as an augmentation of the concept "object" which includes not only physical information such as a 3D mesh, mass and center of mass, but also includes semantic information such as grasp types, potential end effector poses, robot stand poses and information about how the robot should manipulate the objects (explained in detail in Section II). This additional information is of high relevance because it simplifies remote semi-autonomous robot control by providing knowledge about how the state of the environment is and how to interact with it.

Human supervision of semi-autonomous robots implies the robot needs the capability to autonomously perform parts of manipulation tasks. Humanoid robots are complex due to the high number of degrees of freedom (DOF), making direct joint-based teleoperation infeasible. In order to execute manipulation tasks, robots must provide motion planning capabilities [2] considering collision avoidance [12], as well as geometrical world model information such as 3D point clouds [19] and/or grid maps for locomotion planning. This information also needs to be communicated to the human operator to provide situational awareness, supplemented by additional information such as still images and video streams as needed. However, this remote communication is subject to constraints such as interrupts, bandwidth and latency. To overcome some of these communication constraints, the robot must compress and abstract the sensor data transmitted to the human operator. More details about semi-autonomous manipulation, world modelling and providing situational awareness to a remote human operator will be discussed in Section III.

This paper focuses on the improvement of human-robot cooperation to perform manipulation tasks in complex environments based on object templates, where humanoid robots are remotely commanded by a human operator, taking advantage of the capabilities and strengths of humans and robots. We can summarize our contributions as follow:

- Human supervision of semi-autonomous robots by the proposed approach can significantly increase the efficiency to perform manipulation tasks in unconstrained environments compared to fully autonomous or purely teleoperated robots.
- A concept of object templates is proposed which, inspired by the theory of affordances and object-action complexes, allows a fast communication of information from the operator to the robot, which would be error prone and difficult to obtain if the robot would have to extract it autonomously.
- This template based manipulation approach has been implemented in an operator control station and its effectiveness has been demonstrated to accomplish tasks representative for rescue or recovery missions in a disaster scenario (see Section V).

The announcement of the DARPA Robotics Challenge (DRC) gave us the opportunity to test our developments. We participated in the DRC as Team ViGIR[1], a cooperation between our research group at the Technische Universität Darmstadt in Germany and other research institutions in the USA. As members of a track B team, the DRC had imposed challenging time constraints on the development;

[1]http://www.teamvigir.org

we had 8 months to implement our software in a simulation environment to participate in the Virtual Robotics Challenge (VRC) [9] and practically less than three months to implement and test our software in the real Atlas humanoid robot (Fig. 1) developed by Boston Dynamics Inc. (BDI) before participating in the DRC Trials in December 2013 (Section IV). To speed up our developments and in spirit of open source code, we make use of highly advanced open source libraries such as Robot Operation System (ROS) [18], MoveIt! [2], GraspIt! [13] and the Gazebo simulator as complementation for our own software developments. A general description of the complete approach to the DRC including details on OCS, footstep planning, bandwidth-constrained communication and kinematics calibration is presented in [10].

II. OBJECT TEMPLATES

To be able to provide fast and efficient semantic information of the objects of interest to the robot, we created templates of known objects. Graphically, these templates are visualized as a simplified 3D mesh of the object it represents. They are designed to be a general shape of the real object so they can be used for similar objects (e.g. drills from different brands). These object templates contain additional information about each object such as mass and center of mass (CoM), we also created grasp templates to provide pre-computed potential pre-grasp and final-grasp poses, basic grasp types (e.g. cylindrical, prismatic, spherical), as well as information about possibilities of action over each object. This concept of possibilities of action over an object has been previously studied and researched as Affordances [20] or Object-Action complexes [11] (see Section I-A).

Using object templates, the human operator can aid the robot to identify objects of interest in cluttered sensor data as well as their respective properties. In the OCS, the operator can overlap the 3D mesh of an object template with the visualization of the sensor data corresponding to the object of interest. This way, the robot can use the relative pose of the 3D object template to estimate the real object's pose. Once the robot has an estimate of the real pose of the object, the operator can then iterate through the pre-computed list of grasp templates visualizing the arm configurations prior to performing motions on the real robot (Fig. 2).

A. Grasp Template Definition

A grasp template contains the necessary information about where and how to grasp an object template. It allows the operator to visualize the arm configuration needed to grasp the object before actually performing a motion with the real robot. Grasp templates are defined by the tuple:

$$g = (H, E, N, S, P_p, P_f),$$

where:

- $H = \{1, 0\} : LeftHand = 1, RightHand = 0,$
- $E = \{cylindrical, prismatic, spherical\}$ defines the type of grasp to be used,
- N is the vector of fingers joint values where the fingers make contact with the object,
- $S \in \mathbb{R}^2$ defines the desired 2D position of the robot pelvis relative to the template,
- pose $P_p \in \mathbb{R}^3 \times \mathbb{SO}(3)$ defines the position and quaternion orientation of the hand for the pre-grasp, and
- pose $P_f \in \mathbb{R}^3 \times \mathbb{SO}(3)$ defines the position and quaternion orientation of the hand for the the final-grasp.

Several grasps templates are created offline for each object template using the GraspIt! simulator [13]. Different grasps E with their corresponding final finger joints N are created for each object and tested a priori in simulation. Pre-grasps poses P_p are potential hand poses to place the hand in a position near the object (we designed these poses to be around 10 cm away from the object) to reduce the risk of collision (Fig. 2a) while reaching the final-grasps poses P_f, which are the final poses that the hand needs to reach before closing the fingers around the object (Fig. 2b).

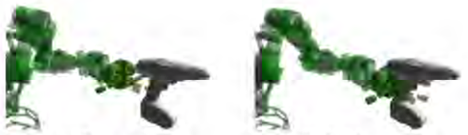

(a) Pre-grasp. (b) Final-grasp.

Fig. 2: Using a "ghost robot" the pre-grasp and final-grasp poses (here shown for the drill template) can be visualized prior to perform an arm motion on the real robot.

To visualize the pre-grasps and final-grasps for each object template, a 3D transparent hand or "ghost hand" is projected in the pose relative to the template. That way, the human operator can choose the location of the hand for a particular task (Fig. 3). The pose S of the robot pelvis relative to the object is selected in a way that allows the end effector to reach the object with high manipulability [24] (Fig. 5b).

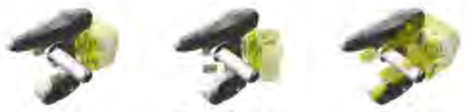

(a) Front. (b) 45 degrees. (c) Handle.

Fig. 3: Final grasps for the drill template.

B. Object Template Definition

Once we have a list of grasp templates for each object we can now create object templates defined by the tuple:

$$x = (I, T, M, C, G, U),$$

where:

- $I \in \mathbb{N}$ is the ID number of the object of interest,
- $T \in \mathbb{N}$ is the type of template (e.g. tools, debris, hose),

- $M \in \mathbb{R}$ is the estimated mass of the object,
- $C \in \mathbb{R}^3$ is the estimated CoM of the object,
- G is a list of potential grasp templates g,
- $U = \{T_x, T_y, T_z, R_x, R_y, R_z\} \in \{0,1\}^6$ is a six-dimensional vector (3D translation and rotation) that defines if an action is possible over a dimension.

Inspired by the theory of affordances from J.J. Gibson [6] we created a vector U to define which actions are possible to be performed over an object. We constrained these actions to translations and rotations in a defined (but not unique) frame of reference in a template. That way, we can command the robot through our OCS to perform actions to a template, for example, the valve can be turned, the drill can be pushed and the door can be opened by turning the handle and pushing or pulling. These constrained path motions of the robot's hand are described in Section III-E. Table I shows examples of the possible actions for the drill, door and hose templates.

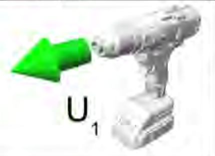
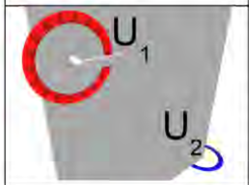
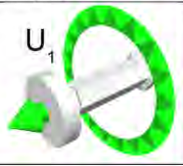
Drill Template	$U_1 = \{1,0,0,0,0,0\}$
	The drill action possibility is a translation along the X axis (green arrow).
Door Template	$U_1 = \{0,0,0,0,1,0\}$ $U_2 = \{0,0,0,0,0,1\}$
	The door action possibilities are to rotate around the Y axis (red ring) in U_1 and rotate around the Z axis (blue ring) in U_2.
Hose Template	$U_1 = \{1,0,0,1,0,0\}$
	The hose action possibility is a translation and a rotation around the X axis (green arrow and ring).

TABLE I: Possible actions over object templates.

III. OBJECT TEMPLATE MANIPULATION

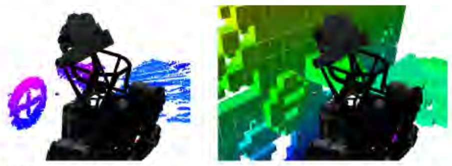
To be able to perform manipulation tasks using object templates, information of the environments as well as from the robot state needs to be gathered.

A. World Model

A 3D model of the environment is generated by aggregating sensor data from LIDAR and cameras. A pose estimate of the robot is obtained by the IMU on its pelvis, and we keep track of different coordinate frames in order to fully reconstruct the pointclouds requested relative to different fixed frames. Using robot pose estimate, internal joint sensing and external sensors, we generate a world model that can also be visualized in the OCS (Fig. 4). This model is used for multiple applications such as visualizing all joint states, self filtering from sensor data and collision avoidance.

B. Planning

Once the world model is obtained, the sensor data is processed and simplified to generate efficient motion plans. To avoid collisions with the environment [12], the robot creates a 3D Octomap [7] representation of the world model (Fig. 4b). For locomotion planning, 2D grid map slices from regions of interest are created and used to generate a collision free footstep plan [8], [23] (Fig. 10c).



(a) World and Robot model. (b) 3D Octomap.

Fig. 4: World model of a valve scenario with robot model and the 3D Octomap for planning.

C. Providing Situational Awareness to the Operator

The robot has access to full resolution sensor data, however, given all the possible constraints that the communication link might be subject to, this information needs to be compressed in order to provide the human operator with situational awareness. Although only joint states and IMU information is sent periodically to the operator, additional sensor data like 3D pointclouds, 2D images and video are provided on request. To reduce the amount of bandwidth used, sensor data is down-sampled and cropped according to the operator's request (Fig. 8d).

D. Pipeline with templates

In this section we will describe a use case of our template manipulation approach (Fig. 5).

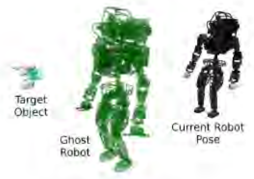
1) Sense: The human operator requests sensor data from the robot's environment to gather situational awareness.

2) Plan: After identifying the sensor data that corresponds to the real object, the human operator overlaps an object template (Fig. 5a). The ghost robot then moves to the pose S of the template, and places the end effector in the final-pose P_f (Fig. 5b).

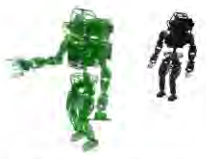
3) Walk: The human operator commands the robot to walk to the stand pose S for which a a footstep plan is generated (Fig. 5c). The robot executes this plan.

4) Grasp: Once the robot reaches the stand pose S, the human operator commands the robot to grasp the object and the robot generates a motion plan. The pre-grasp P_p pose, the final-grasp P_f pose and the grasp N are then executed.

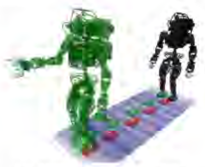
5) Use: The human operator can then command the robot to generate a manipulation plan for the object based on the possible actions U.



(a) Ghost robot



(b) Ghost robot grasping

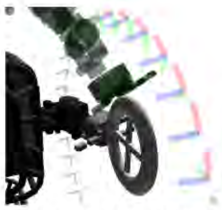


(c) Footstep to template

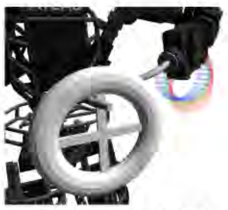
Fig. 5: Ghost robot grasp preview with a footstep plan to reach the object [23].

E. Cartesian and Circular Path planning

The vector U defined in each template is used to create motions that are constrained to follow a Cartesian path between the initial and final end effector's pose. Waypoints are generated based on linear interpolation between initial and final poses. By using spherical linear interpolation (Slerp) [22] orientations for the end effector's goal pose can be different from the start end effector's orientation. More complex constrained motions such as circular motion are generated by concatenating multiple short linearly interpolated Cartesian paths. These constrained motions can also be designed to maintain the end effector's orientation (Fig. 6). This video[2] shows the operator commanding the robot to close a valve in a simulation environment.



(a) Rotating valve with hand.



(b) Rotating valve with hook.

Fig. 6: Circular path plan to turn a valve 360 degrees. In (a) the hand rotates around changing its orientation, while in (b) the hook rotates around keeping its orientation. Interpolated poses are shown for the last joint of the arm.

IV. EXPERIMENTS AND RESULTS

The DRC Trials 2013 were held in Homestead, FL, USA. Research groups from different countries participated in a series of tasks to demonstrate robot capabilities for rescue missions. These tasks considered robot capabilities such as mobility and manipulation in disaster environments like:

- Walk through rough terrain,
- Climb up a ladder,
- Remove debris blocking a doorway,
- Open three different types of doors,
- Break through a wall using a cutting tool,
- Attach a fire-hose to a wye,
- Close three different types of valves and
- Drive a car.

Seven of the eight tasks required grasping and manipulation. Each of the tasks consisted of three defined checkpoints. A point was given for each accomplished checkpoint and in case all checkpoints were accomplish without an intervention (robot failure which required a restart) an extra bonus point was given. The tasks required different amounts of mobility and manipulation and for the purposes and scope of this paper we will describe the Hose task and the Valve task because they contains good examples where we applied our template based manipulation approach. A comprehensive description of all eight task results is presented in [10].

A. Hose Task

In the Hose task, the robot needed to walk to a reel and pick up a fire hose, then walk with the fire hose towards a wye and attach it by turning the nozzle (Fig. 7a). The first point in this task was obtained when the robot crossed the yellow line on the floor while carrying the hose. The second point was given when the hose came in physical contact with the wye and the third point was obtain for attaching the hose. Our approach to accomplish this task was first to divide it in three subtasks: pick up the hose, touch the wye with the hose and attach it. The figures shown in this section contain screenshots from the OCS, either a top-down view of the environment or a 3D view. The figures also contain images of the real scenario, obtained from different cameras located in the walls of the task (Fig. 7a).

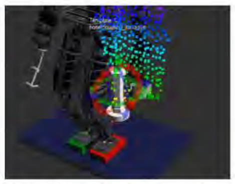
1) Pick up the hose: Following the pipeline described in Section III-D we began the task by acquiring sensor data information from the environment. The secondary operator requests a pointcloud of the reel and inserts a hose template aligning it to the 3D data belonging to the real hose (Fig. 7b). Then the primary operator requests a footstep plan to a position relative to the hose template where the robot can easily grasp the hose as shown in Fig. 7c and Fig. 7d. Once the robot is standing in front of the hose, the primary operator requests a grasp to pre-grasp pose of the ghost hand (Fig. 8a and Fig. 8b), then the robot moves the hand to the final-pose and executes the grasp (Fig. 8c and Fig. 8d).

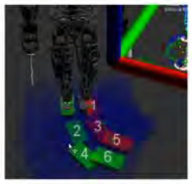
After grasping the hose the primary operator commands the robot to move one meter to the right based on an environment map previously requested by the secondary operator (Fig. 9).

2) Touch the wye with the hose: For this subtask we applied the same pipeline again. The secondary operator requests a pointcloud of the wye (Fig. 10a) and inserts the

[2]https://www.youtube.com/watch?v=wKFJO-Zkjck

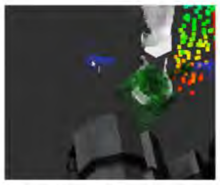
(a) Hose setup.

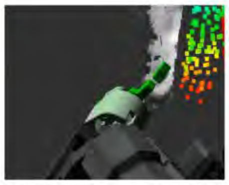

(b) Hose poincloud and template.


(c) Footstep plan to Hose.

(d) Robot executing plan.

Fig. 7: Request to walk to the hose. (a) Robot start position, hose reel and wye. (b) Reel and hose pointcloud with the template of the hose. (c) Footstep plan visualization to the hose in the OCS before walking. (d) Robot following the footstep plan to the hose.

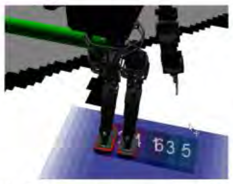

(a) Hose ghost hand.


(b) Robot executing plan (OCS).

(c) Robot executing plan.

(d) Robot grasping hose.

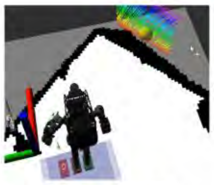
Fig. 8: Request to grasp hose. (a) Hose ghost hand visualization for the operator to verify. (b) and (c) Automatic motion of the robot to place the hand in final-pose. (d) Operator verifies the grasp through robot's camera.

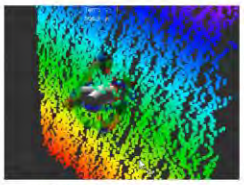

(a) Footstep plan away from reel.

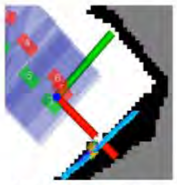
(b) Robot executing plan.

Fig. 9: Request to walk away from reel. (a) Footstep plan with lateral right steps. (b) Robot walking with the hose.

wye template, aligning it to the 3D data belonging to the real wye (Fig. 10b). Then the primary operator requests a footstep plan to a position relative to the wye template where the robot can easily touch the wye with the hose (Fig. 10c and Fig. 10d). Once the robot is standing in front of the wye, the primary operator request the robot to move the hand to the pre-grasp pose of the ghost hand (Fig. 11a), which the robot executes (Fig. 11b and Fig. 11c). Then the robot moves the arm to the final-grasp pose which makes the hose to come in physical contact with the wye (Fig. 11d).

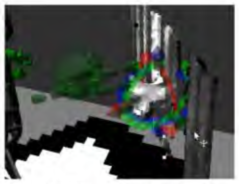

(a) Wye pointclod.

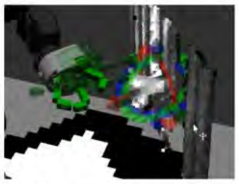

(b) Wye template aligned.


(c) Footstep plan to wye.

(d) Robot executing plan.

Fig. 10: Request to walk to the wye.


(a) Wye ghost hand.


(b) Robot moving arm to ghost hand.

(c) Robot moving arm to ghost hand.

(d) Hose and wye make physical contact.

Fig. 11: Robot commanded to bring the hand near the wye and then touch it.

3) Attach the hose: At this point we have successfully applied our template based approach to align the fire hose to the wye and the only missing thing is turning the nozzle to engage the hose. Given the extremely small size of the nozzle bumps used to turn it (around 0.25 cm^3), this subtask was not feasible to solve using our approach. Teleoperation was used instead, however, this was a fine manipulation task which required high precision and even the hose was correctly located (Fig. 12a) and the nozzle was turned 180 degrees (Fig. 12b), the threads of the wye and the hose did not engage, and the hose fell after releasing it.

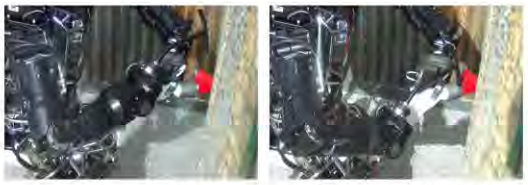

(a) Hose aligned. (b) Robot turning the nozzle.

Fig. 12: Attach hose and turn nozzle (teleoperated).

B. Valve Task

For accomplishing the Valve task, the robot needed to close three different valves (Fig. 13a). We followed the same pipeline as in the Hose Task in order to command the robot to place the hook inside the valves as shown in Fig. 13b).

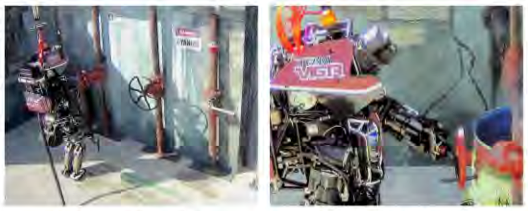

(a) Three valves setup. (b) Hook inside the valve.

Fig. 13: Valve task.

Once the robot placed the hook inside the valve and the valve template was aligned, the primary operator selects the option to close the valve from a menu in the OCS simply by indicating the amount of degrees. As shown in Fig. 14, the robot starts moving the arm in a circular path as described in Section III-E. This circular motion plan is always relative to the axis of rotation of the valve template.

C. Results

To compare our results, we analysed the performance of other teams using the Atlas robot during the DRC Trials. Table II shows the timetables of the activities performed during the Hose Task day one [3] and day two [4].

These results show that we were the fastest to perform manipulation tasks. We were able to walk with the hose through the yellow line in the floor within 8 minutes, touch the wye at time 10:10, align it with the wye at 18:00 and start turning it at time 22:20. From these results we can see

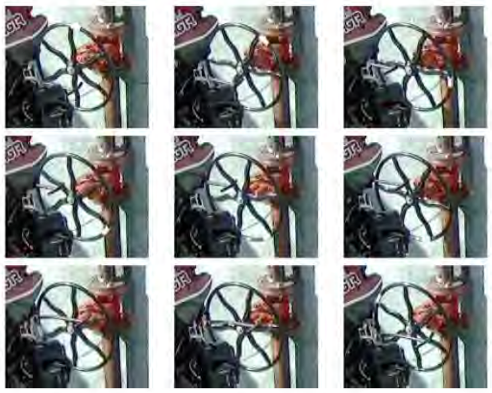


Fig. 14: Robot autonomously closing a real valve.

that there were only two teams able to turn the nozzle of the hose, and in most of the other manipulation activities we were faster than other teams.

V. CONCLUSIONS AND FUTURE WORK

We have presented a humanoid robot manipulation approach based on templates of objects of interest. The proposed approach enables human-robot cooperation to perform manipulation tasks in a more efficient way compared to pure teleoperated and full autonomous robot approaches. The use of object templates allows a human operator to efficiently send semantic commands to a remote humanoid robot, such as “Walk to this waypoint”, “Grasp this object” or “Turn this object” so a mission can be planned on the fly. As shown in Section IV our template based manipulation approach is very useful for grasping and manipulating various objects. Some limitations of our approach are that small objects that require fine manipulation are still a challenging problem and also that the operator has to invest time to manually align templates to sensor data. As shown in Table II we were the fastest Atlas team to obtain two points in the Hose Task and the fastest to turn the nozzle from the only two teams that tried. We can see how, even though we placed 9^{th} overall in the Trials, our template manipulation approach gave better results than the 2^{nd} place results in the Hose Task.

Future work will focus on developing automatic template fitting algorithms to increase the speed of our approach. We are also researching automatic grasp planners to be able to create new grasps on the fly. Our current definition of templates only allows to define the possibilities of action as constrained motion paths, but we would like to explore options for adding information about expected forces needed for these motions. An analysis of some (anonymous) DRC teams results has been done in [26] which we are analysing to find opportunities for improvements in our approach.

ACKNOWLEDGMENT

The authors would like to thank all members of Team ViGIR, their contribution and support enabled the realization

Activity	**Team ViGIR**	IHMC	MIT	TRACLabs	WRECS	Trooper	HKU
Task Start Time (min)	00:00	00:00	00:00	00:00	00:00	00:00	00:00
Stand in front of Hose	00:40	01:15	02:15	01:00	01:00	03:25	03:40
Picked Hose	04:00	08:40	03:53	06:20	03:10	14:40	-
First Point (Crossed Line)	**08:00**	11:50	09:30	08:45	07:55	29:30	-
Stand in front of Wye	08:30	13:25	11:00	09:15	08:18	-	-
Second Point (Touched Wye)	**10:10**	15:20	15:25	16:25	11:28	-	-
Hose alignment	18:00	16:20	22:35	29:00	-	-	-
Turned Nozzle	**22:20**	23:00	-	-	-	-	-
Points	2	2	2	2	2	1	0
Place overall DRC Trials	9^{th}	2^{nd}	4^{th}	6^{th}	6^{th}	8^{th}	10^{th}

TABLE II: Hose Task timetable for all Atlas teams [3], [4].

of this work; we would especially like to thank Alex Goins and Dr. Ravi Balasubramanian from Oregon State University for assistance getting the grasping software working and testing, and Felipe Bacim and Dr. Doug A. Bowman of Virginia Tech for their work in the operator interface design and development. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract FA8750-12-C-0337, by the Deutscher Akademischer Austauschdienst (DAAD Germany), and by the Secretaría de Educación Pública (SEP México).

REFERENCES

[1] A. Bierbaum and M. Rambow, "Grasp affordances from multi-fingered tactile exploration using dynamic potential fields," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 168–174.

[2] S. Chitta, I. Sucan, and S. Cousins, "Moveit!" *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[3] DARPA Robotics Challenge Trials, "DRC Trials Blue Hose Task Day 1," https://www.youtube.com/watch?v=n3aGCe0vDLU, 2013, accessed: 2014-10-11.

[4] ——, "DRC Trials Blue Hose Task Day 2," https://www.youtube.com/watch?v=B35A7-JDldI, 2013, accessed: 2014-10-11.

[5] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. Perez D'Arpino, R. Deits, M. DiCicco, D. Fourie, *et al.*, "An architecture for online affordance-based perception and whole-body planning," 2014. [Online]. Available: http://hdl.handle.net/1721.1/85690

[6] J. J. Gibson, "The theory of affordances," in *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, R. Shaw and E. J. Bransford, Eds., Hilldale, USA, 1977, pp. 67–82.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com. [Online]. Available: http://octomap.github.com

[8] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, 2012, pp. 674–679.

[9] S. Kohlbrecher, D. C. Conner, A. Romay, F. Bacim, D. A. Bowman, and O. von Stryk, "Overview of Team ViGIR's approach to the Virtual Robotics Challenge," in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1–2.

[10] S. Kohlbrecher, A. Romay, A. Stumpf, A. Gupta, O. von Stryk, F. Bacim, D. A. Bowman, A. Goins, R. Balasubramanian, and D. C. Conner, "Human-robot teaming for rescue missions: Team ViGIR's approach to the 2013 DARPA Robotics Challenge Trials," 2014, accepted for Journal of Field Robotics.

[11] N. Krüger, C. W. Geib, J. H. Piater, R. P. A. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrcen, A. Agostini, and R. Dillmann, "Object-action complexes: Grounded abstractions of sensory-motor processes," *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 740–757, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/ras/ras59.html#KrugerGPPSWUAKOAD11

[12] A. Leeper, K. Hsiao, M. Ciocarlie, I. Sucan, and K. Salisbury, "Methods for collision-free arm teleoperation in clutter using constraints from 3d sensor data," in *IEEE Intl. Conf. on Humanoid Robots*, Atlanta, GA, 10/2013 2013.

[13] A. Miller and P. K. Allen, "Graspit!: A Versatile Simulator for Robotic Grasping," *IEEE Robotics Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[14] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, *et al.*, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[15] K. Nagatani, S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada, "Redesign of rescue mobile robot quince," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 13–18.

[16] M. Nieuwenhuisen, J. Stückler, A. Berner, R. Klein, and S. Behnke, "Shape-primitive based object recognition and grasping," in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE, 2012, pp. 1–5.

[17] L. U. Odhner, L. P. Jentoft, M. R. Claffee, N. Corson, Y. Tenzer, R. R. Ma, M. Buehler, R. Kohout, R. D. Howe, and A. M. Dollar, "A compliant, underactuated hand for robust manipulation," *International Journal of Robotics Research*, 2013.

[18] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[19] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[20] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.

[21] D. Schiebener, J. Schill, and T. Asfour, "Discovery, segmentation and reactive grasping of unknown objects," in *Humanoids*, 2012, pp. 71–77.

[22] K. Shoemake, "Animating rotation with quaternion curves," *ACM SIGGRAPH computer graphics*, vol. 19, no. 3, pp. 245–254, 1985.

[23] A. Stumpf, S. Kohlbrecher, D. C. Conner, and O. von Stryk, "Supervised footstep planing for humanoid robots in rough terrain tasks using a black box walking controller," in *Humanoid Robots (Humanoids), 2014. 14th IEEE-RAS International Conference on*. IEEE, 2014.

[24] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Robot placement based on reachability inversion," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1970–1975.

[25] K. Welke, J. Issac, D. Schiebener, T. Asfour, and R. Dillmann, "Autonomous acquisition of visual multi-view object representations for object recognition on a humanoid robot," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2012–2019.

[26] H. Yanco, A. Norton, W. Ober, D. Shane, A. Skinner, and J. Vice, "Analysis of human-robot interaction at the DARPA Robotics Challenge Trials," 2014, submitted to Journal of Field Robotics.

Humanoids 2015 Paper on Manipulation [5]

Achieving Versatile Manipulation Tasks with Unknown Objects by Supervised Humanoid Robots based on Object Templates

Alberto Romay[1], Stefan Kohlbrecher[1], David C. Conner[2] and Oskar von Stryk[1]

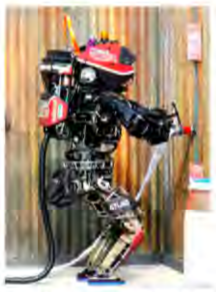
***Abstract*— The investigations of this paper are motivated by the scenario of a supervised semi-autonomous humanoid robot entering a mainly unknown, potentially degraded human environment to perform highly diverse disaster recovery tasks. For this purpose, the robot must be enabled to use any object it can find in the environment as tool for achieving its current manipulation task. This requires the use of potential unknown objects as well as known objects for new purposes (e.g. using a drill as a hammer). A recently proposed object template manipulation approach is extended to provide a semi-autonomous humanoid robot assisted by a remote human supervisor with the versatility needed to utilize objects in the described manner applying affordances [5] from other previously known objects. For an Atlas humanoid robot it is demonstrated how using a small set of such object templates with well defined affordances can be used to solve manipulation tasks using new unknown objects.**

I. Introduction

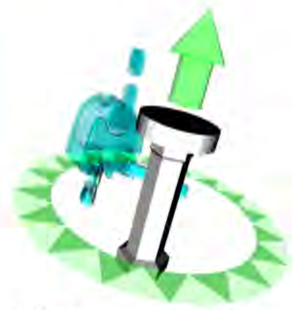
Robotic manipulation is a well known problem in the research community. It considers different aspects such as the task objective, the object used and the manipulation requirements needed to fulfil the task objective. Fully autonomous robots operating in remote unstructured environments are challenged by constraints such as lighting conditions, reachability uncertainties, degraded communications, and unknown availability of objects. On the other hand, purely teleoperated robots require high bandwidth with low latency communication channels and demand a high mental workload from the operator, which limits their utility.

Human supervised semi-autonomous robots leverage strengths from both fully autonomous and purely teleoperated approaches. This combination reduces the cognitive workload of the operator by allowing communication through semantic commands. These semantic commands are then leveraged with powerful computing abilities of robots to perform autonomous tasks such as collision-free manipulation motion planning and Cartesian trajectory planning. The Object Template manipulation approach introduced in [15] demonstrated the ability to provide this kind of interaction. In the context of this paper, the manipulation concept goes beyond the idea of how an object is grasped, and focuses on how an object should be moved around in the environment.

As motivated by the DARPA Robotics Challenge (DRC), a remote semi-autonomous robot should be capable of entering an unknown, partially degraded human environment to perform disaster recovery tasks. Utilizing a human supervisor in the loop to aid the robot with perception ability based on sensor data, the robot needs to be able to use available objects or tools found in the environment. During a robotic manipulation task, a human supervisor can remotely provide the robot with information of task objective and object locations in the environment. Still, accomplishing a remote manipulation task can be challenged by the lack of known objects in the environment. We present experiments that show how, using the approach in [15], a human supervisor can provide the robot with the versatility to perform manipulation tasks relying on the manipulation skills designed for some previously known objects.



(a) Atlas attaching a fire hose



(b) Fire hose template with affordances

Fig. 1: a) Developed by Boston Dynamics Inc. (BDI) the Atlas robot has 28 hydraulically actuated DOF, weighs approximately 150 Kg and stands 1.88 m tall is shown attaching a fire hose during the DRC Trials 2013. b) Shows the fire hose template, the potential grasp pose of the hand in translucent blue and in translucent green the available affordances of the fire hose, like the circular motion needed to tangle the threads while turning it and the Cartesian translation needed to push the fire hose against the pipe.

A. Related Work

Improvisation is a powerful human ability that has not yet been deeply explored in robotic manipulation research. An interesting research approach by Stilman *et al.* presented the "MacGyver" paradigm as a research problem [19]. They propose that robots should be able to use arbitrary objects in the environment to solve unforeseen manipulation tasks. Performing these actions autonomously for unknown tasks in real world scenarios with degraded conditions is not feasible within the next years. Some autonomous approaches like [21], [22], and [23] have demonstrated autonomous capability for obtaining semantic information from objects. However, these approaches still require development to be able to perform autonomously in less controlled environ-

[1]Department of Computer Science, TU Darmstadt, Germany romay,kohlbrecher,stryk@sim.tu-darmstadt.de
[2]TORC Robotics, VA, USA conner@torcrobotics.com

ments and unforeseen tasks. The idea of a robot applying manipulation skills to different objects has also been suggested by Leidner *et al.* [12]. In a semi-autonomous remote supervised approach, a human operator can effectively aid the robot to apply the necessary manipulation skills needed to achieve the task.

Şahin *et al.* [17] formalized the term of *affordance* [5] in the robotics field. This has been key to the development of robot control approaches [1], [24]. Object-Action Complexes (OACs) is a concept created by Kruger *et al.* [11] to formalize the relationships between objects and actions and defines affordances as state-transition functions that can be used for predictions. OACs give robots the possibility to understand how an object will behave after an action is performed and use continuous sensorimotor experience to build symbolic representations.

The approach in [7] presents the Affordance Template ROS package. Their approach provides a high level of adjustment and interactivity of the geometry information provided by the templates. They also introduce an Affordance Template Description Format which describes in an XML file the relationships between the used robot and the end effector waypoints needed to manipulate the corresponding template. Their user interface is based on the publicly available RViz interactive markers [6] which is analogously used in this approach. The approach previously presented in [15] is different given that grasp information is separately defined from the object information. The grasp library contains potential information of the robots end effector pose described in the template coordinate frame of reference, and the object template library contains, additional to the physical properties of the object (like mass, center of mass and inertia tensors among others), the affordances of the object. These affordances are defined as motion constraints like circular paths or Cartesian translations (either maintaining the end effector orientation or rotating along with the motion) commonly needed during manipulation of the object to achieve a particular manipulation task.

The approaches presented in [25], in [3] by MIT, and in [10] IHMC, also converge to an approach that analogously uses 3D geometric meshes with information about grasping capabilities. The differences are the way in which we define the possible actions to be executed over an object and how we interact with them. For example, to use the open and close affordances of a valve, the operator either manually rotates the template of the valve to create a motion plan or the user needs to previously define end effector waypoints like in [7]. In the approach presented here, the operator specifies a command such as "open" or "close" setting a number like "$\pm$360 degrees" and on-line waypoints are then generated on the fly.

B. Overview

We participated in the DRC as Team ViGIR[1] with the highly advanced humanoid robot Atlas shown in Fig. 1. Team ViGIR is a cooperation between research groups in Germany and other research institutions in the USA. After our participation in the three main events, the Virtual Robotics Challenge (VRC) [8] in June 2013, the DRC Trials in December 2013 [9] and the very recent DRC Finals in June 2015, we are confident that the previously proposed object template manipulation approach can be used to perform manipulation tasks in unstructured environments using human supervision of a remote semi-autonomous robot. This paper adds to our previous contribution an approach to perform versatile robotic manipulation tasks in remote unstructured environments based on object templates. This approach increases the accomplishment potential of manipulation tasks by providing the ability to transfer manipulations skills from known objects to similar new unknown objects on the fly.

[1]http://www.teamvigir.org

II. OBJECT TEMPLATES

This section briefly describes the concept of Object Templates. Detailed information is provided in [15]. Visually, an object template is a simplified 3D mesh of the object it represents, but implicitly contains physical and semantic information to aid the robot using it. Object Templates can contain physical object information such as mass, center of mass (CoM) and inertia tensor. Semantic information in an Object Template provides pre-computed potential pre-grasp and final-grasp poses, as well as information about their affordances as shown in Fig. 1b. We use an Operator Control Station (OCS) to manipulate object templates. The OCS helps in the visualisation of the sensor data acquired by the robot in an immersive 3D environment. Using the OCS a human operator can remotely identify the objects in the robot environment and overlap the 3D geometry mesh of the object template to the corresponding sensor data. The 3D pose of the Object Template can then be used by the robot for locomotion planing to walk towards the real object [20] and create motion plans to manipulate it. A demonstration of the use of Object Templates during the DRC Trials 2013 can be seen in [16].

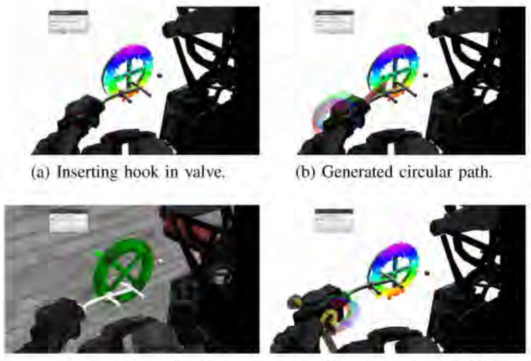

(a) Inserting hook in valve. (b) Generated circular path.
(c) Rotating valve with hook. (d) Hook keeps orientation.

Fig. 2: Simulation experiment to turn a valve 360 degrees.

The affordances in each template are defined as a pose $\in \mathbb{R}^3 \times \mathbb{SO}(3)$ used to create constrained motions paths between the initial and final end effector pose. We use linear interpolation to generate on-the-fly waypoints between the initial and final pose. The end effector goal pose orientation can be different from the start pose thus the end effector will rotate along with the translation motion [18]. Motions that will constrain the end effector to keep its original orientation during the translation motion can also be generated. Circular paths around the axis of rotation of a template can be generated by concatenating multiple short linearly interpolated Cartesian paths (Fig. 2), which can also generate spiral motions, by combining both linear and circular motions simultaneously. This experiment can also be seen in this video[2].

Since the affordance based design is grasp agnostic, the operator can command the robot to generate a circular path to rotate around the axis of rotation of the template independently from the pose of the end effector. Waypoints are located over a plane normal to the rotation axis where this plane intersects the origin of the frame of reference of the hand. This way, the robot has the freedom to grasp the object in any way while still being able to calculate waypoints around the axis of rotation.

III. VERSATILE OBJECT MANIPULATION

This section describes the concept of versatility in the context of robotic manipulation using object affordances. Inspired by the work of [19] we propose a "MacGyver" paradigm for operator assisted humanoid robots. In this paradigm, versatile manipulation is the key ability to succeed in a particular task where expected known objects are not present. Versatile manipulation using the proposed approach allows for skills designed for previously known objects to be transferred to new unknown objects on the fly, utilizing the object template of a similar known object and provided that objects found in the environment have similar properties to previously known objects. In the context of this paper we talk about versatile manipulation in how tasks can be achieve by moving an object through the environment. We focus in defining affordances for some objects to achieve a task, and how we either apply these affordances as designed, or we use them in a new way which was initially not planned.

As concluded by Liu et al. [13] in humans, grasps are distinguished by features related to the grasping action such as the intended motion, force, and stiffness. The ability of humans to transfer these properties between objects increases the rate of success in manipulation tasks. In a similar way, these properties are needed for robot control and the ability to transfer them between similar objects can help to achieve manipulation tasks. Doing this autonomously in unstructured and degraded environments is not feasible within the few next years. For this reason, the assistance of a human operator can help in identifying the tasks requirements and the objects that could be used to achieve them.

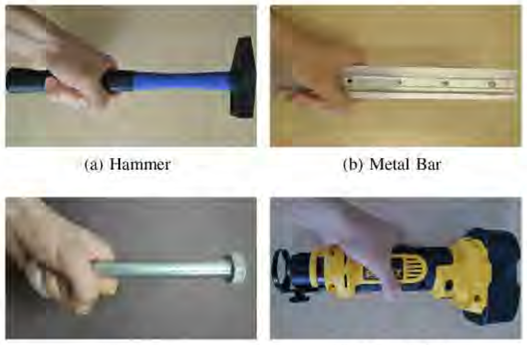
[2]`https://www.youtube.com/watch?v=wKFJO-Zkjck`

For example, consider the case where the robot enters a degraded environment and the human supervisor identifies the next task objective is breaking a glass panel to gain access to a fire hose or to allow smoke to dissipate. A commonly used object for this task would be a hammer. However, if a hammer is not be available in the environment, similar objects like debris, pipes or other tools that have similar properties to a hammer could be utilized with the same manipulation motions designed for using a hammer.

A. *Object Classification*

During a disaster scenario multiple objects from a wide variety of types can be found in the environment. In our approach, we will constraint the object space that our robot can use to objects that can be grasp and manipulated with a hand. To make clear the object space that we are considering we classify them in two groups:

1) Floating Objects: Refers to the objects mobile in the environment that can be grasped and lifted. For this reason similar physical properties between objects are needed to transfer affordances. Size, mass, center of mass, hardness among others need to be similar in such a way that the task might still be possible to achieve. In our hammering example, objects that would fit in the hand and that have the hardness and mass necessary to break a window could be used instead of the hammer as shown in Fig. 3.


(a) Hammer (b) Metal Bar

(c) Bolt (d) Vertical Drill

Fig. 3: Floating objects. Mass, center of mass, hardness and size properties are similar enough to use them to create a force impact into another object.

2) Constrained Objects: Refers to the objects that have limited degrees of freedom in the environment. In this case, physical properties of the objects are not so relevant compared to the motion constraints that define the use of the object. For example, the drawer of a desk, a door handle and the door itself have motion constraints that can be defined as linear or circular motions with respect to an axis in a frame of reference as shown in Fig. 4.

B. *Manipulation Classification*

In this section we describe the classification of manipulation tasks that we are considering in our approach. To

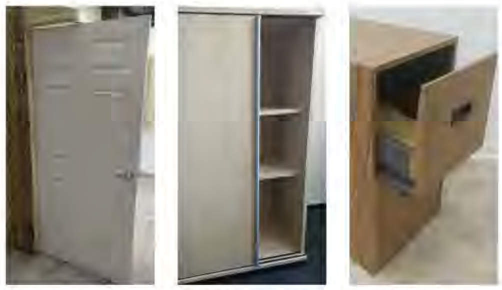

(a) Door and Handle (b) Sliding Door (c) Drawer

Fig. 4: Constrained objects. Degrees of freedom are well defined, the door and handle have one axis of rotation, the drawer and the sliding door have one axis of translation.

be able to transfer manipulation skills between two similar objects it is important to define in detail which properties of each manipulation motion are needed to accomplish a task. We constraint our approach to a representative set of manipulation tasks based on the classification of human manipulation behaviour made by Bullock et al. in [2] as seen in Fig. 5.

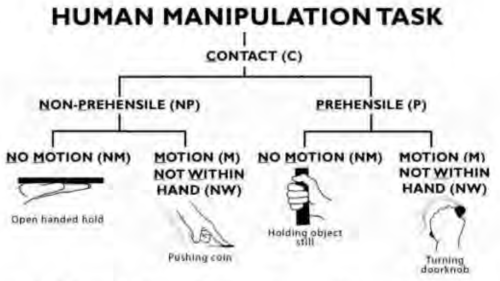


Fig. 5: Manipulation classification to be used in our approach. Image based on [2].

Based on the manipulation properties described by [14] and [4] we selected the six classes which are identified to be most commonly used in manipulation tasks. The nomenclature used to identify each class uses four letters, each of them represents a type of motion in a dimension and the letters used are "u" for unconstrained motion, "t" for a motion that is only translational, "r" for a motion that is only rotational, and "x" for a motions that is constrained. The selected classes can be seen in Table I.

C. Manipulation Transferring Types

We have defined three different types of examples to differentiate the ways in which a manipulation task can be achieved using our approach. Each one of these example types represent a way of transferring manipulation skills; we describe them as follow:

Class	Description	Example
uuu	Free Motion	Moving a floating object
uur	Point on a plane	Drawing on a white board
ttr	Surface against surface	Cutting with a vertical drill
uxx	Cylinder in slot	Attaching a hose to a pipe
rxx	One rotational DoF	Turning a doorknob
txx	One translational DoF	Pulling a drawer

TABLE I: Manipulation classes to be used in our approach. Table based on [4].

1) Type 1: Transferring manipulation skills between objects in which physical properties can differ, but that they can still be considered to be the same object. For example, turning valves of different radius or with different number of cross bars, pulling drawers of different shapes and lengths. This is a simple way of transferring skills between objects, and the cases where objects are exactly the same is considered as a special case of this type of manipulation transferring.

2) Type 2: Transferring manipulation skills between different objects, but with same manipulation classes. This means that the affordances of the objects can be utilized in the same way. For example, a common doorknob manipulation class is "rxx" which allows rotation in only one axis, this means we can use the turn affordance of the valve template which belongs to the same manipulation class. Pushing a box under a table requires a manipulation class "txx" which can be achieved by using the push affordance from a drawer. This type of manipulation transferring shows how a robot is capable of using an object based on manipulation skills designed for another object in a way that will allow the robot to achieve the task goal.

3) Type 3: Transferring manipulation skills through the use of intermediate objects. This type of manipulation skill transferring can be seen as the "MacGyver" paradigm described in this approach. This refers to how the robot is capable of manipulating an object for a different purpose than the one it was designed for and potentially utilizing manipulation skills from another object. The improvisation ability is still provided by the human supervisor, but it is now possible to transfer manipulation skills to use new objects. This increases the potential of the human-robot team to continue the manipulation task.

IV. EXPERIMENTS

In this section we show experiments that demonstrate how being able to transfer affordances from one object to another or manipulating objects in a way they have not been used before can increase the potential to achieve a manipulation task. Experiments of type 1 will be used to demonstrate that the basic manipulation skills can be performed by the humanoid robot, also shown in [15]. Experiments of manipulation skill transferring types 2 and 3 directly represent the proposed approach in this paper.

A. Drawing a Circle: Type 1

In this experiment we show a test designed to evaluate the necessary manipulation skills that will be required to cut a circle from a dry wall using a vertical drill. To test these motions, we installed a black marker instead of the drill bit, and used a white board to show the path that the drill is following. The manipulation class required to cut a circle in a drywall using a vertical drill is "ttr" which means that the drill should always be perpendicular to the drywall plane and that can be translated in upwards and downwards directions. The affordances of the vertical drill can then be used to draw a circle in a whiteboard since both manipulation tasks belong to the same class as shown in Fig. 6.

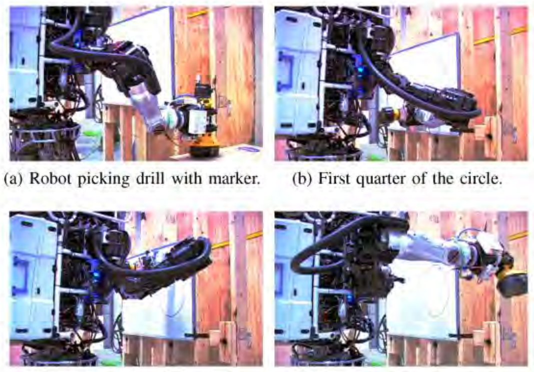

(a) Robot picking drill with marker. (b) First quarter of the circle.
(c) Three quarters of the circle. (d) Completed drawing circle.

Fig. 6: The robot generating a clock-wise circular path to draw a circle in a whiteboard, but using the "cut circle" affordance of a vertical drill. Video available as multimedia attachment.

B. Plugging a Cable: Type 1

In this experiment we show a laboratory experiment from one of the surprise tasks that was used during the DRC. The plug task consisted of unplugging a cable from a magnetic socket and plug it into another magnetic socket located within a reachable distance of the robot as shown in Fig. 7. The manipulation class required to plug a cable in a socket is "uxx" which means that the cable should always be aligned to the socket axis of insertion. This manipulation class is the same as the one required to attach a hose to a pipe [16] thus it was straight forward for the operator to utilize the affordances of the fire hose template.

C. Steering Wheel: Type 2

In this experiment a human-robot team has been prepared for tasks involving valve manipulation (opening and closing). But in this case, the robot runs into a situation where using a car is needed and the robot has no previous knowledge of how to manipulate a steering wheel. The steering wheel is a constrained object of class "rxx", it can only rotate around one axis. Assuming the robot can take a sit in the vehicle, the operator can then utilize the valve template to operate the steering wheel. Since the affordances of the valve template can produce the same necessary movements to turn a steering

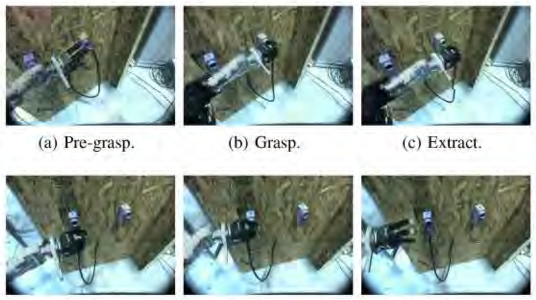

(a) Pre-grasp. (b) Grasp. (c) Extract.
(d) Pre-insert. (e) Insert. (f) Release.

Fig. 7: Robot first person view of cord-plug surprise task. The task requires the cable to be unplugged from the right socket and be plugged to the left socket.

wheel, the operator can overlap the valve template with the sensor data that belongs to the steering wheel and use the turn affordance of the valve template as shown in Figure 8.

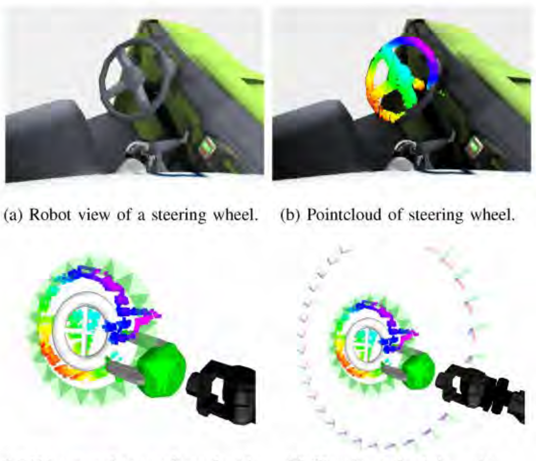

(a) Robot view of a steering wheel. (b) Pointcloud of steering wheel.
(c) Valve template overlapped with steering wheel pointcloud. (d) Circular path of the wrist.

Fig. 8: The robot generating a circular path to turn the steering wheel with the right hand, but utilizing the "turning" affordance of a valve template.

D. Blocked Door: Type 2

This experiment shows a task where the robot needs to open a door and walk through it. In this case the door has been blocked by a truss that fell in front of it as shown in Fig. 9. The truss is an overweighted tube structure which is difficult for the robot to lift. As humans would do, pushing or pulling a heavy object is a common manipulation motions for these cases which belong to class "txx" translation in one axis. A simple circular motion of the shoulder would not be sufficient due to the curved trajectory that the end effector would have, for this reason, a Cartesian path between the initial and final position of the end effector needs to be performed.

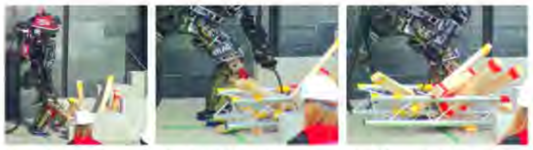
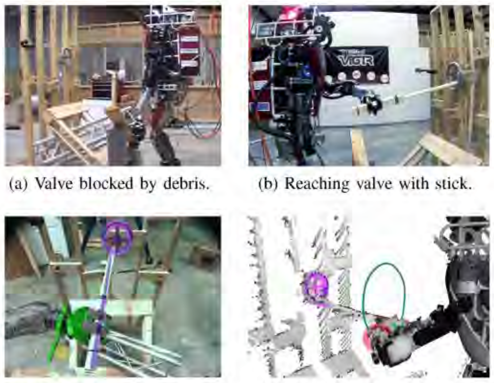

(a) Robot and truss. (b) Grabbing the truss. (c) Truss being pulled.

Fig. 9: A door blocked by a truss needs to be pulled away. Utilizing a translation affordance originally designed for an object with translation constraints (e.g. a drawer) can achieve the necessary manipulation motions to pull the truss in a direction parallel to the floor.

E. Unreachable Valve: Type 3

This experiment shows a situation where the robot needs to turn a valve, but in this case the valve is unreachable by the robot as shown in Fig. 10. The human supervisor then finds a wooden stick which can be used to reach the valve as shown in Fig. 10b. Since the affordances we designed are grasp agnostic, the operator can easily command the robot to generate a circular path to rotate the a stick around the axis of rotation of the valve. This experiment shows kinematic waypoints can be created on the fly regardless of the grasp pose with respect to the object and generate the same type of manipulation class required to accomplish the task.


(a) Valve blocked by debris. (b) Reaching valve with stick.

(c) Robot view from scene. (d) Circular path of the hand.

Fig. 10: The robot using the "turning" affordance of a valve template but utilizing a stick to reach the valve. The circular path shown in (d) is calculated for the hand with respect to the valve axis of rotation and keeping the end-effector orientation. Video available as multimedia attachment.

V. CONCLUSIONS AND FUTURE WORK

We have presented a versatile manipulation approach for a supervised semi-autonomous humanoid robot to increase the potential of achieving manipulation tasks. This approach allows manipulations skills to be transferred from known objects into new unknown objects on the fly. The ability to transfer manipulation knowledge between objects increases the potential to achieve a task by allowing the use of objects in new different ways as shown in Fig. 10 or the use of new unknown objects as shown in Fig. 8. We presented motivating examples which are by no means exhaustive, given the three example types for transferring manipulation skills one can think of different possible combinations.

Team ViGIR participated in the DRC Finals held in Pomona, California in the U.S. on June 2015. Systematic evaluation of the proposed approach has not yet been possible to achieve due to preparations for this event. We expect to add real experiments with the Atlas robot during July 2015. Ongoing work to improve our performance is focusing on automatic template fitting and tracking algorithms to increase the speed of our approach. Currently the definition of templates allows only to define affordances as constrained kinematic motion paths, but we would like to explore options for adding dynamic information, for example, expected forces needed for these motions.

ACKNOWLEDGMENT

The authors would like to thank all members of Team ViGIR, their contribution and support enabled the realization of this work; Felipe Bacim and Dr. Doug A. Bowman of Virginia Tech for their work in the operator interface design and development and Dr. Ravi Balasubramanian from Oregon State University for assistance getting the grasping software working and testing. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) through the Air Force Research Lab (AFRL) under contract FA8750-12-C-0337 and by the Deutscher Akademischer Austauschdienst (DAAD).

REFERENCES

[1] A. Bierbaum and M. Rambow, "Grasp affordances from multi-fingered tactile exploration using dynamic potential fields," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 168–174.

[2] I. Bullock and A. Dollar, "Classifying human manipulation behavior," in *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, June 2011, pp. 1–6.

[3] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. Perez D'Arpino, R. Deits, M. DiCicco, D. Fourie, *et al.*, "An architecture for online affordance-based perception and whole-body planning," 2014. [Online]. Available: http://hdl.handle.net/1721.1/85690

[4] T. Feix, I. Bullock, and A. Dollar, "Analysis of human grasping behavior: Correlating tasks, objects and grasps," *Haptics, IEEE Transactions on*, vol. 7, no. 4, pp. 430–441, Oct 2014.

[5] J. J. Gibson, "The theory of affordances," in *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, R. Shaw and E. J. Bransford, Eds., Hilldale, USA, 1977, pp. 67–82.

[6] D. Gossow, A. Leeper, D. Hershberger, and M. Ciocarlie, "Interactive Markers: 3-D User Interfaces for ROS Applications [ROS Topics]," *Robotics Automation Magazine, IEEE*, vol. 18, no. 4, pp. 14–15, Dec 2011.

[7] S. Hart, P. Dinh, and K. Hambuchen, "The Affordance Template ROS Package for Robot Task Programming," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015.

[8] S. Kohlbrecher, D. C. Conner, A. Romay, F. Bacim, D. A. Bowman, and O. von Stryk, "Overview of Team ViGIR's approach to the Virtual Robotics Challenge," in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1–2.

[9] S. Kohlbrecher, A. Romay, A. Stumpf, A. Gupta, O. von Stryk, F. Bacim, D. A. Bowman, A. Goins, R. Balasubramanian, and D. C. Conner, "Human-robot teaming for rescue missions: Team vigir's approach to the 2013 darpa robotics challenge trials," *Journal of Field Robotics*, vol. 32, no. 3, pp. 352–377, 2015. [Online]. Available: http://dx.doi.org/10.1002/rob.21558

[10] T. Koolen, J. Smith, G. Thomas, S. Bertrand, J. Carff, N. Mertins, D. Stephen, P. Abeles, J. Englsberger, S. Mccrory, *et al.*, "Summary of team IHMC's Virtual Robotics Challenge entry," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2013.

[11] N. Krüger, C. W. Geib, J. H. Piater, R. P. A. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrcen, A. Agostini, and R. Dillmann, "Object-action complexes: Grounded abstractions of sensory-motor processes." *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 740–757, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/ras/ras59.html#KrugerGPPSWUAKOAD11

[12] D. Leidner, C. Borst, and G. Hirzinger, "Things are made for what they are: Solving manipulation tasks by using functional object classes," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, Nov 2012, pp. 429–435.

[13] J. Liu, F. Feng, Y. C. Nakamura, and N. S. Pollard, "A taxonomy of everyday grasps in action," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, Nov 2014, pp. 573–580.

[14] J. Morrow and P. Khosla, "Manipulation task primitives for composing robot skills," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 4, Apr 1997, pp. 3354–3359 vol.4.

[15] A. Romay, S. Kohlbrecher, D. C. Conner, A. Stumpf, and O. von Stryk, "Template-based Manipulation in Unstructured Environments for Supervised Semi-autonomous Humanoid Robots," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, Nov 2014, pp. 979–986.

[16] A. Romay, S. Kohlbrecher, A. Stumpf, O. von Stryk, F. Bacim, D. A. Bowman, A. Goins, R. Balasubramanian, and D. C. Conner, "Hose task at the 2013 DARPA Robotics Challenge trials: Team ViGIR's results video," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, Nov 2014, pp. 1095–1095.

[17] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.

[18] K. Shoemake, "Animating rotation with quaternion curves," *ACM SIGGRAPH computer graphics*, vol. 19, no. 3, pp. 245–254, 1985.

[19] M. Stilman, M. Zafar, C. Erdogan, P. Hou, S. Reynolds-Haertle, and G. Tracy, "Robots using environment objects as tools the "MacGyver" paradigm for mobile manipulation," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014, pp. 2568–2568.

[20] A. Stumpf, S. Kohlbrecher, D. C. Conner, and O. von Stryk, "Supervised footstep planing for humanoid robots in rough terrain tasks using a black box walking controller," in *Humanoid Robots (Humanoids), 2014. 14th IEEE-RAS International Conference on*. IEEE, 2014.

[21] J. Sturm, "Learning kinematic models of articulated objects," in *Approaches to Probabilistic Model Learning for Mobile Manipulation Robots*, ser. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2013, vol. 89, pp. 65–111. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37160-8_4

[22] M. Tenorth, S. Profanter, F. Balint-Benczedi, and M. Beetz, "Decomposing cad models of objects of daily use and reasoning about their functional parts," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 5943–5949.

[23] V. Tikhanoff, U. Pattacini, L. Natale, and G. Metta, "Exploring affordances and tool use on the icub," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, Oct 2013, pp. 130–137.

[24] K. Welke, J. Issac, D. Schiebener, T. Asfour, and R. Dillmann, "Autonomous acquisition of visual multi-view object representations for object recognition on a humanoid robot," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2012–2019.

[25] H. A. Yanco, A. Norton, W. Ober, D. Shane, A. Skinner, and J. Vice, "Analysis of human-robot interaction at the darpa robotics challenge trials," *Journal of Field Robotics*, vol. 32, no. 3, pp. 420–444, 2015. [Online]. Available: http://dx.doi.org/10.1002/rob.21568

This page intentionally blank.

F. FOOTSTEP PLANNING SYSTEM

In this section we present more details about the developed footstep planning system and framework.

F.1. Footstep Planning System

The basic footstep planning approach is already described in Section 3.2.5 and [4]. This approach tackles multiple challenges to enable full-size humanoid robots to cross difficult terrain in real world application. Even with no details of the underlying walking controller available, the planner is able to utilize the versatile locomotion capabilities of a full-size humanoid robot. It is capable of generating full 6 DoF footstep sequences that allows safe execution by walking controllers. A terrain model generator allows generating a quickly accessible 3D world model from all perceived 3D laser scans. Hence, we have presented an integrated footstep planner as it comes with full perception and planning pipeline. For further details we would like to refer to the mentioned sources.

This approach has been evaluated successfully with the Atlas robot in real world experiments. During the DRC Trials the integrated footstep planner allowed traversing the pitch ramp and chevron hurdles within eight minutes. The operator only had to command the desired goal position behind the obstacle. During the competition the footstep planner has already worked well, but there were still issues which had to be addressed until DRC Finals.

During the DRC Trials a major issue encountered was limited operator ability to correct planning. If the planning system failed to deliver a feasible solution for some reason e.g. a bad world model due to obstructed obstacles, the operator could not assist the planner effectively. The operator could only define simple step pattern commands using a dedicated widget. But back then the pattern mode was not able to assist the operator in terms of 3D planning or step validation.

For this reason the footstep planner was extended to provide better services for interaction via graphical user interfaces (in our particular case Team ViGIR's OCS). These services provide the following features:

- Stitch multiple plans
- Revalidation of the entire step plan
- Modify single steps of a plan
- Operator assistance (e.g. automatic 3D foot placement adjustment)
- Planning preemption
- Goal pose to feet poses transformation
- Waypoint mode (in preparation)

While depending on the implementation of the graphical user interface, all these features enable interactive footstep planning with the human in the loop. The operator can request a footstep plan and in case of bad steps just modify them instead of triggering replanning or manual pattern generation. An example how the interactive planning mode looks like is illustrated by Figure 28 in Section 3.2.5. In addition to the usage by graphical user interfaces these services provide a wide range of helper tools for any high-level software (e.g. behavior control), granting easier access to the comprehensive footstep planning interface.

Since the DRC Trials we have been able to improve the overall planning performance of the planner. Especially the planning runtime has been improved which allows to planner to optimize plans faster and deliver better results. The 3D terrain generator has been improved as well, providing the ability to generate terrain models for the footstep planning system online. This new terrain model generator has already been applied and validated for real world scenarios as shown in the results section.

F.2. Footstep Planning Framework

After the DRC Trials many opportunities arose to show that our approach supports a wide range of walking controllers for biped humanoid robots. First, IHMC announced to make their controller software available for all Atlas teams. Afterwards, Team VALOR adapted the Team ViGIR software infrastructure for use with their robot ESCHER. Lastly, Team Hector qualified their THOR-Mang robot for the DRC Finals. The planning system has been integrated with these three different biped humanoid robots, with each of these coming with their own walking controller. This provided the opportunity to show that the footstep planning approach can successfully be deployed on other full-size humanoids besides ATLAS. But variations of different available robot systems raised the question how to do this correctly.

This motivated the development of a footstep planning framework based on our prior work. The main objective is to provide an integrated footstep planning framework which may be deployed easily into an existing ROS setup. As a framework the planner has to be expandable for new features but closed for modifications. Any user of the framework should only have to implement and extend robot specific elements to get the advanced planning system running instead of developing a modified version of an existing planner or even starting from scratch each time. All already implemented and thus proven algorithms are kept untouched which decreases the likelihood of errors and saves a lot of implementation effort. Although, the framework must generalize well, it is able to solve difficult terrain task problems and utilize the versatile locomotion capabilities of the given walking controller.

In order to meet this objective the plugin management system *vigir_pluginlib*[53] has been implemented. It provides the capability to manage versatile plugins which can be also used outside of the footstep planning domain. Our package is based on *pluginlib*[54] which already allows for dynamically loading plugins using the ROS build infrastructure. We have extended the package into a semantic plugin management system. The practical implementation consists of two parts: The *plugin base class* and the *plugin manager*.

F.2.1. Plugins

Plugins are used to efficiently inject user specific code into the planning pipeline. The user is able to execute robot specific code during the footstep planning process without any modifications to the framework.

The plugin base class contains the basic maintenance variables and methods which are needed by the plugin manager. Each plugin can be identified by its unique name and contains semantic hints about the

[53] https://github.com/team-vigir/vigir_pluginlib
[54] http://wiki.ros.org/pluginlib

plugin's semantic base class in order to efficiently identify the plugin type and its capabilities. The semantic base class is not to be confused with the plugin base class, but rather is a specialized plugin base class which defines the functionality and content of all derived plugins. Figure 69 illustrates an example inheritance hierarchy for plugins which also shows that it is possible that semantic plugins are derived from other semantic plugins. In this case all derived plugins will give only semantic hints to the latest semantic base class in the hierarchy.

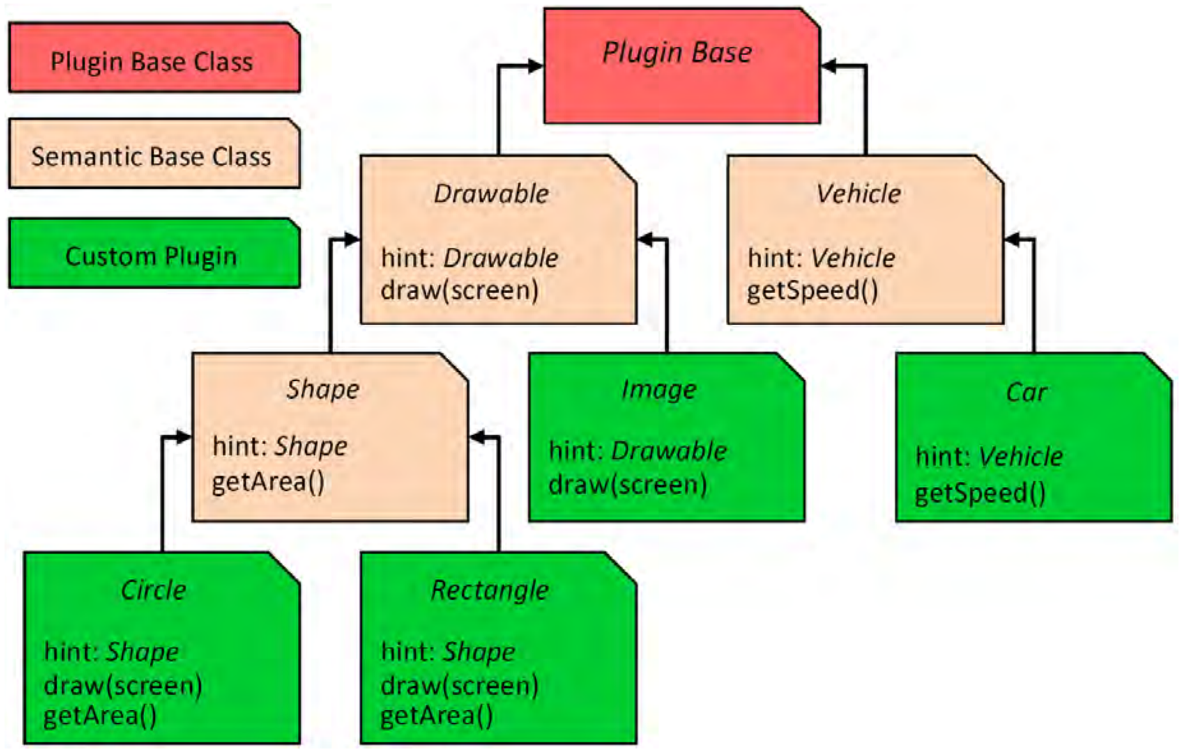


Figure 69. Example for a plugin inheritance hierarchy.
Hereby, it is illustrated to which base class the semantic hint will point to.

In some cases a plugin type may cause concurrency issues due to their intended purpose, when multiple instances of the same semantic base class exist. For this reason each semantic base class is able to declare itself to be a unique type. This declaration will disallow the plugin manager to maintain more than one instance of this plugin type at the same time. Once this uniqueness has been defined by any inherited semantic base class, each derived class must not remove this classification. Despite of a clear sign of a class hierarchy design flaw this could cause unexpected side effects.

Each (custom) package is able to export their own semantic base classes as well as concrete plugins using the ROS toolchain. Therefore, all generic tools like a user interface and even the plugin manager are getting automatically aware of every new plugin.

F.2.2. Plugin Manager

The plugin manager is responsible for maintaining and providing simple access to all plugins. Currently, the plugin loading sequence has to be hardcoded in the initialization of the footstep planner node. The option to load dynamically plugins is in preparation. In the meantime the plugin manager already supports adding, replacing and removal of plugins during runtime. It is possible to retrieve specific plugins in multiple ways: By name, by semantic hints, and by inheritance hierarchy.

Every plugin has to be named uniquely in the entire system and thus can be uniquely identified by its name. Therefore, the first and most straight-forward way to obtain a plugin from the plugin manager is by name (see Figure 70). Retrieving plugins by semantic hints will only deliver the ones which exactly match the given semantic hints. The inheritance hierarchy will be ignored as illustrated in Figure 71. This mode is less important and should only be used if an efficient lookup of a specific plugin type is needed, but the name is not known. In general, the most flexible and dynamic mode is lookup by inheritance hierarchy which should be preferred. In this mode the manager will check if a plugin inherits from the requested semantic base class. The manager is able to return all plugins that fulfill the requirements defined by the semantic base class independent of any semantic hints or plugin names (see Figure 72). This concept assumes that all plugins as well as the inheritance hierarchy are designed cleanly, thus all defined functionality of the inherited semantic base classes must be implemented properly by the plugin.

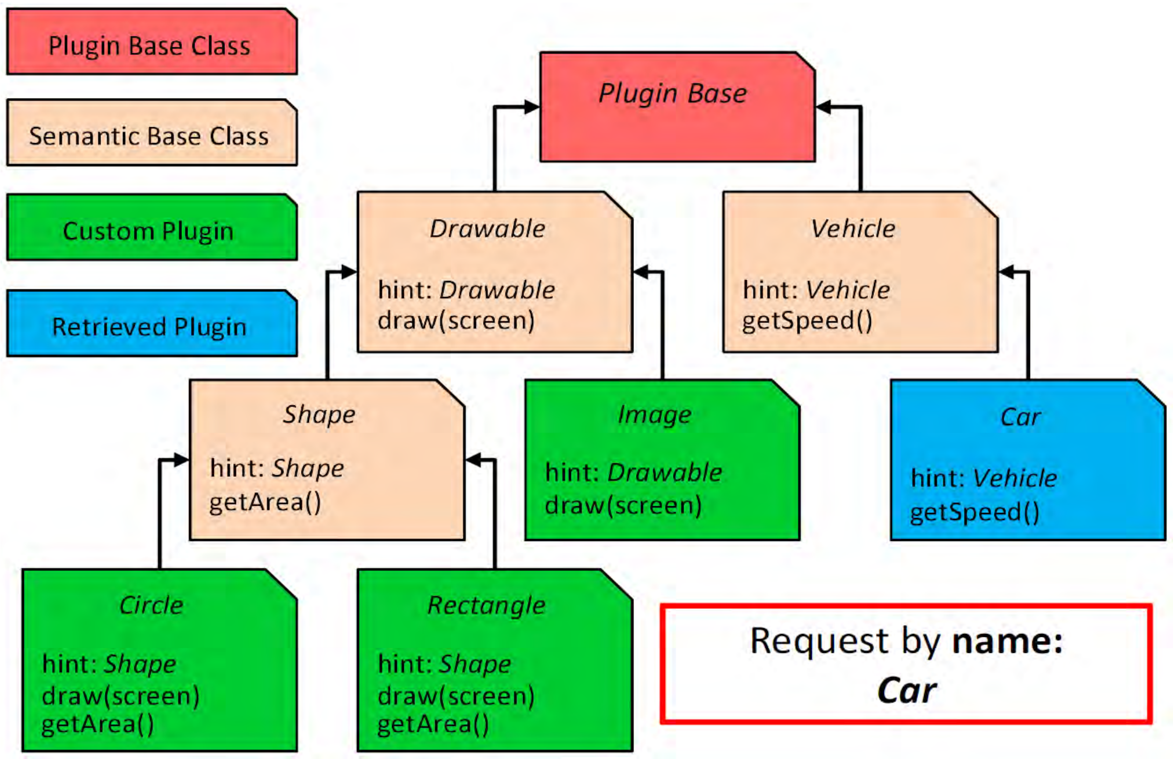


Figure 70. Example for obtaining plugins by their name. Here, the plugin named *Car* have been requested.

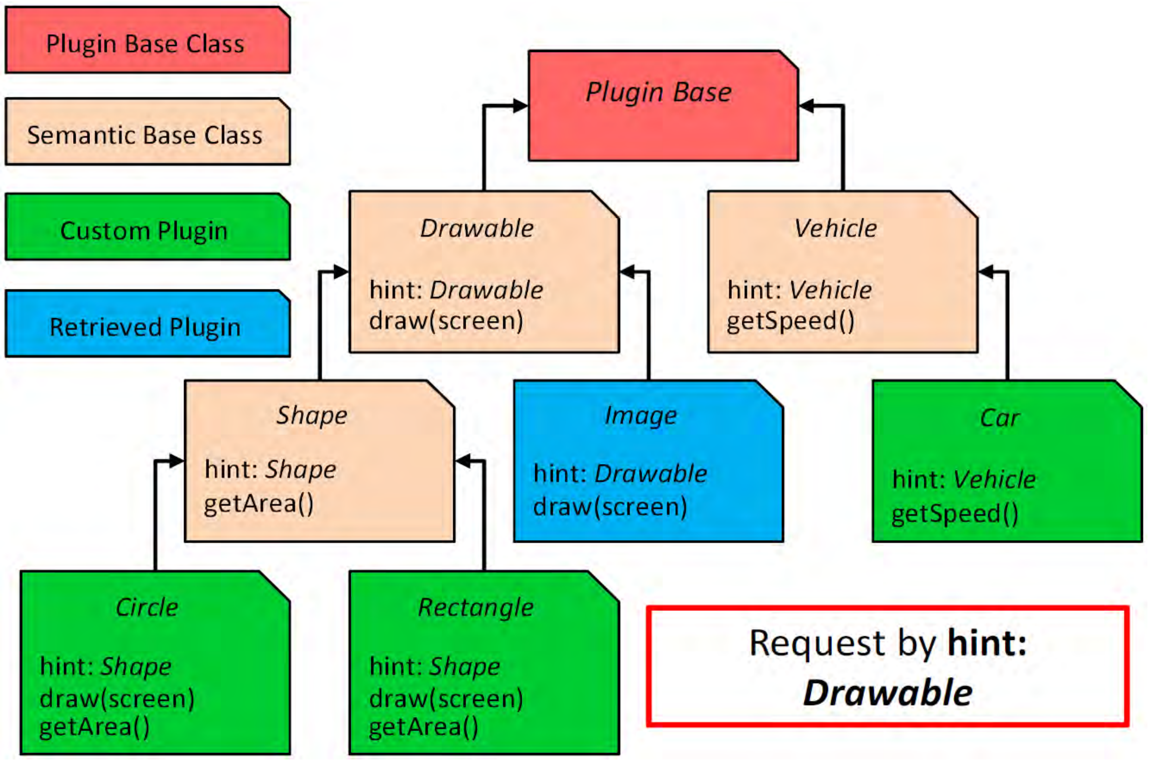

Plugin Base Class
Semantic Base Class
Custom Plugin
Retrieved Plugin
Plugin Base
Drawable hint: *Drawable* draw(screen)
Vehicle hint: *Vehicle* getSpeed()
Shape hint: *Shape* getArea()
Image hint: *Drawable* draw(screen)
Car hint: *Vehicle* getSpeed()
Circle hint: *Shape* draw(screen) getArea()
Rectangle hint: *Shape* draw(screen) getArea()
Request by **hint:** ***Drawable***

Figure 71. Example for obtaining plugins by their semantic hint. Here, all plugins having the semantic hint of *Drawable* have been requested.

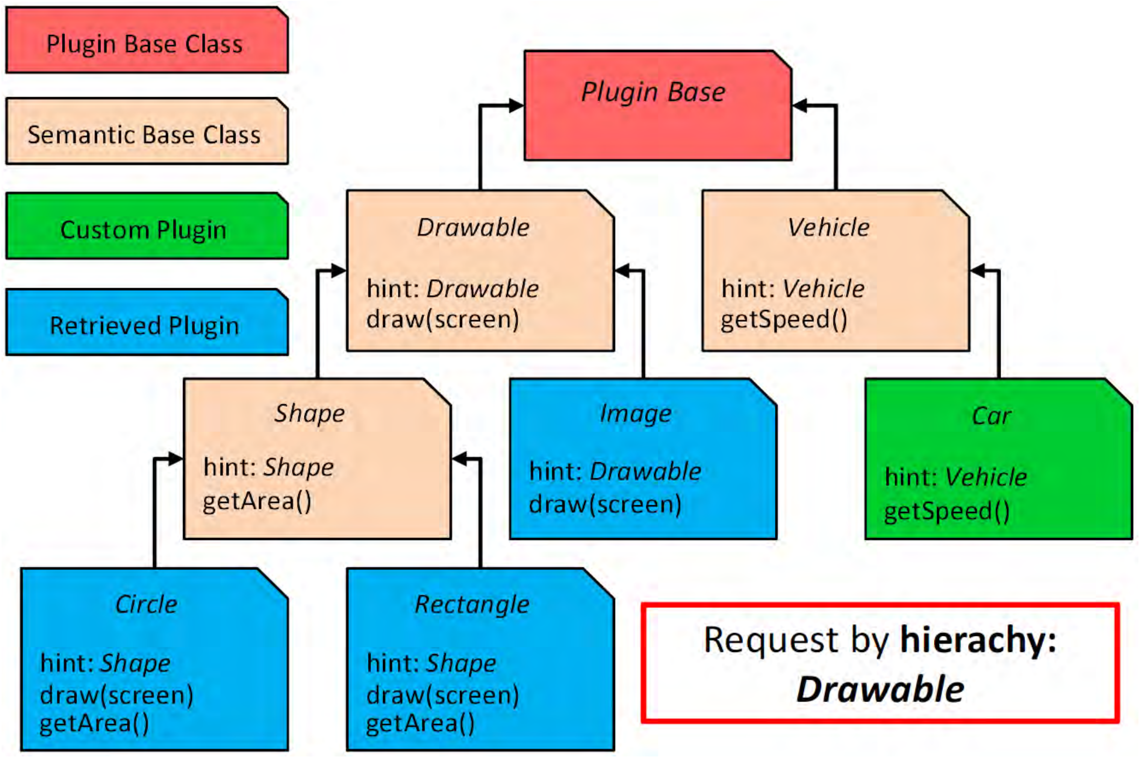

Plugin Base Class
Semantic Base Class
Custom Plugin
Retrieved Plugin
Plugin Base
Drawable hint: *Drawable* draw(screen)
Vehicle hint: *Vehicle* getSpeed()
Shape hint: *Shape* getArea()
Image hint: *Drawable* draw(screen)
Car hint: *Vehicle* getSpeed()
Circle hint: *Shape* draw(screen) getArea()
Rectangle hint: *Shape* draw(screen) getArea()
Request by **hierachy:** ***Drawable***

Figure 72. Example for obtaining plugins by their inheritance hierarchy. Here, all plugins derived from Drawable have been requested.

The plugin manager itself is automatically instantiated for the entire system as a singleton. This design decision was made to prevent issues due to multiple plugin manager instances and allows providing global and simplified access. It automatically sets up all ROS services and action servers which provides generic access to the plugin management capabilities (e.g. dynamically loading plugins).

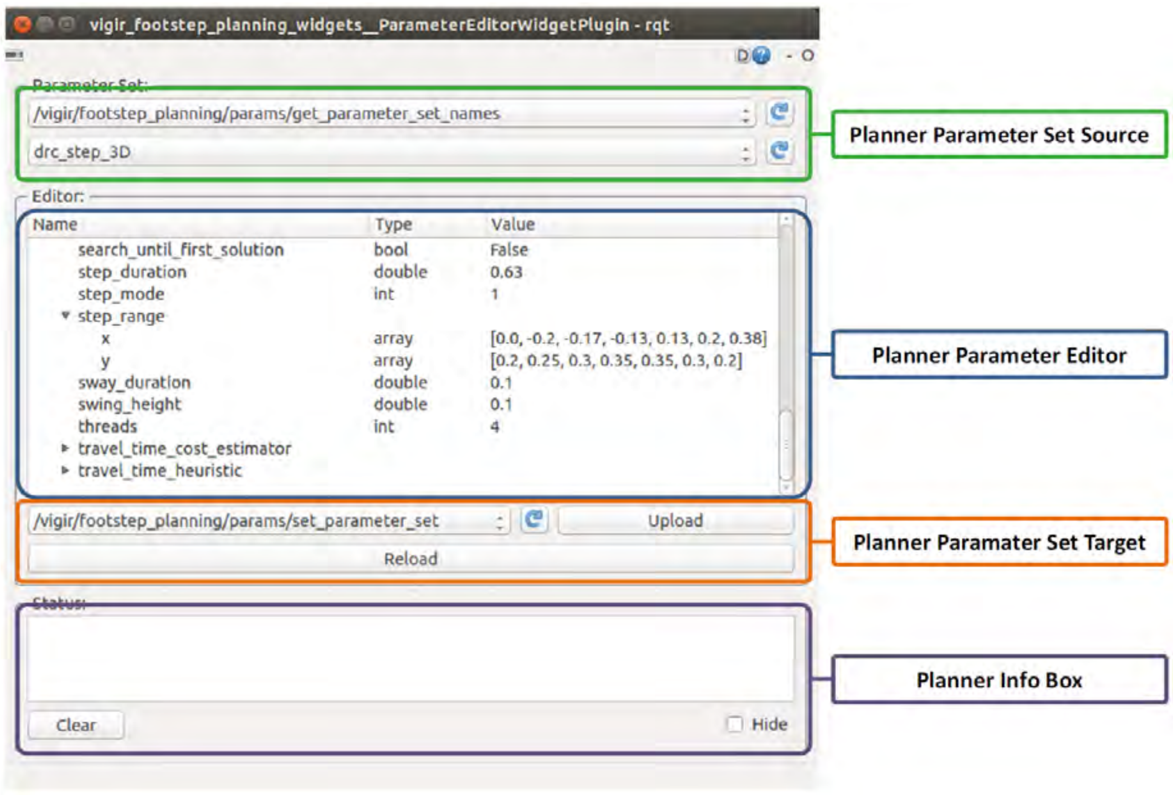
F.2.3. Parameter Management System

In real world application different terrain scenarios need to be tackled (e.g. flat surface, stairs or sloped terrain). The footstep planner can perform best if a dedicated set of parameters has been defined for each kind of terrain scenario. This also allows the operator to switch easily between different planning behaviors. Furthermore, it is desirable to be able to modify a parameter set if the situation requires it. In general these requirements can be solved using the available ROS message infrastructure. Plugins however, are supposed to extend the footstep planner with new features. The structure of parameter sets may vary which is in conflict to ROS messages as they require a static structure. A simple solution would be separate configuration files and well as user interfaces for each plugin which is undesirable due to high maintenance effort.

This motivated the development of a new parameter management system. The XML-RPC library already used by ROS system is used, as it already provides a suitable data structure for our purpose. Each parameter set can thus be modeled as nested XML-RPC values. This data representation allows easily applying a marshalling algorithm converting the data into a byte stream. The resulting byte stream can be packed into a regular ROS message as a vector of characters. This overcomes the basic conflict of static ROS message structures for interprocess communication and the need of flexible content due to user defined parameter sets. Although the approach is introduced here in the context of footstep planning, it can be used for any software system.

With the new parameter management system it is now very easy to manage multiple parameter set configuration files. If a new parameter set is needed, the new configuration file only has to be placed in a preconfigured folder. The parameter manager is able to locally load and store all parameter sets found in this folder. The OCS makes use of this feature and automatically updates the user interface to show all given parameter sets which can be selected by the operator afterwards (see Figure 40 in Section 4.2.3).

The parameter manager has been designed in a similar way like the plugin manager. It is automatically instantiated as a singleton, able to maintain multiple parameter sets and provides services for adding, removing and editing parameter sets which can be accessed via ROS service and action servers. In Figure 73 generic graphical user interface using these services is shown. It allows modifying parameter sets of any parameter set structure on-line.


vigir_footstep_planning_widgets__ParameterEditorWidgetPlugin - rqt

Parameter Set:

/vigir/footstep_planning/params/get_parameter_set_names

drc_step_3D

Editor:

```
Name                          Type     Value
    search_until_first_solution   bool     False
    step_duration                 double   0.63
    step_mode                     int      1
  ▾ step_range
        x                         array    [0.0, -0.2, -0.17, -0.13, 0.13, 0.2, 0.38]
        y                         array    [0.2, 0.25, 0.3, 0.35, 0.35, 0.3, 0.2]
    sway_duration                 double   0.1
    swing_height                  double   0.1
    threads                       int      4
  ▸ travel_time_cost_estimator
  ▸ travel_time_heuristic
```

/vigir/footstep_planning/params/set_parameter_set

Figure 73. Parameter Editor Widget

F.2.4. The Footstep Planning Framework

The new plugin and parameter management systems form the infrastructure base of the footstep planning framework. The footstep planner pipeline has been checked for places where a user might want to affect the behavior of the planner. For each found place a semantic base class has been introduced:

- *CollisionCheckPlugin*: Basic collision check of a given state or transition
- *CollisionCheckGridMapPlugin*: Specialized CollisionCheckPlugin for occupancy grid maps
- *HeuristicPlugin*: Computes heuristic value from current state to goal state
- *PostProcessPlugin*: Allows performing additional computation after each step or step plan has been computed.
- *ReachabilityPlugin*: Check if transition between two states is valid
- *StepCostEstimatorPlugin*: Estimates cost and risk for given transition
- *StepPlanMsgPlugin* (unique): Marshalling interface for robot specific data
- *TerrainModelPlugin* (unique): Provides 3D model of environment

The last two semantic base classes are defined to be unique which means there can be only one running instance at once. Figure 27 in Section 3.2.5 shows when which plugin takes effect on the planner pipeline. For a quick deployment of the framework concrete plugin implementations for common cases do already exist for all these semantic base classes.

One of our main goals is keeping the footstep planner efficiency high as possible. Therefore, the computational overhead of the plugin system must be kept to a minimum. It obviously is inefficient to retrieve needed plugins for each single call during the planning process. For this reason the planner retrieves all plugins only once and pushes the given parameters into them before starting planning. Additionally, a mutex locks all critical callback functions of the planning system. The footstep planner is thus protected against any changes of the plugin as well as parameter manager during the planning process.

The deployment into an existing ROS setup requires multiple steps, but many of them are optional. The first step is to create a ROS node which initializes custom plugins and adds them to the plugin manager. This step is going to become obsolete in the next version as the plugin manager will be able to instantiate default as well as customized plugins using configuration files. The most important integration part is the mandatory hardware interface. There currently is no explicit hardware interface provided by the footstep planning framework. In general each new robot or walking controller requires implementation effort for an appropriate hardware adapter which is can translate the generated footstep plan so it can be used by the walking controller.

Advanced walking controllers usually need very specific data to perform complex locomotion. For instance, this data could be intermediate trajectory points of the foot or the convex hull of expected ground contact. The framework has been designed to be able to provide this capability. The presented plugin system allows perform any kind of additional computing needed by the walking controller. Analogously to the parameter management system, all custom data can be carried as byte stream within the regular step plan messages. Marshalling algorithms already available for basic data types can be applied here as well. Marshalling for complex data types has to be implemented as customized StepPlanMsgPlugin. The framework is thus able to pack all custom data into the generic step plan message and send it to the hardware adapter, where it gets unpacked and forwarded to the walking controller. This illustrates how our framework supports any kind of walking controller without any modifications.

F.3. Results & Conclusions

For detailed results of our integrated footstep planner we refer to one of our publications. Thus, the following section we will focus on the new framework.

Although the novel footstep planning framework is still under development, it has already been evaluated. Thanks to the framework we could provide our footstep planning system to the three completely different humanoid robots: Atlas, ESCHER and THOR-Mang. Team VALOR (ESCHER) and Team Hector (THOR-Mang) have utilized the footstep planner for their own robots during the DRC Finals. They could perform locomotion tasks using exactly the same high level software as Team ViGIR. Thus far, in total five walking controllers have been interfaced successfully with the framework. The use with Atlas showed the benefit of the expandability, as BDI’s step mode needs additional data for each step to perform 3D walking. This data is be provided by dedicated plugins.

As already mentioned above, the 3D terrain generator has been enhanced to generate terrain models for the footstep planning system online. Figure 74 shows an example of a real world experiment. The terrain generator is able to accumulate all data while walking. The data stays consistent; the robot is thus able to step on the cinder block.

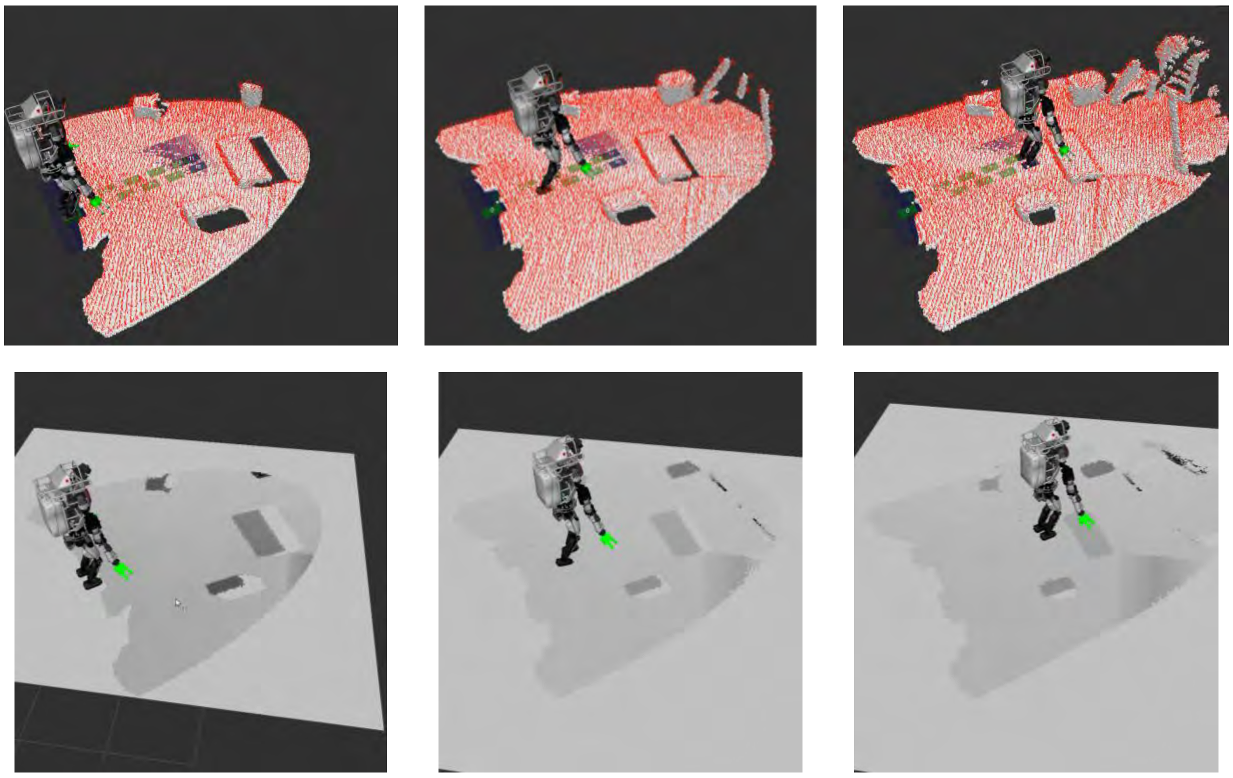


Figure 74. Example how the terrain model is extended while walking during a real robot experiment. The upper row shows the 3D data and estimated normals (red lines). The lower row shows a visualization of the generated height map.

The DRC Finals showed that our objective of a versatile footstep planning framework was achieved. The three mentioned robots are using different walking controllers, but the footstep planner core can be maintained easily across all robot platforms. Although the framework does already work well, there are still some issues and missing features which will be delivered in future versions.

The entire footstep planning has been already open-sourced at GitHub:

- https://github.com/team-vigir/vigir_footstep_planning_msgs
- https://github.com/team-vigir/vigir_footstep_planning_basics
- https://github.com/team-vigir/vigir_footstep_planning_core
- https://github.com/team-vigir/vigir_terrain_classifier
- https://github.com/team-vigir/vigir_pluginlib
- https://github.com/team-vigir/vigir_generic_params

By open sourcing our software we want to reduce re-invention of the wheel in the community and enable others to quickly get a footstep planning system working on their robots.

F.4. Future Work

Based on remaining issues and ideas there still are many options for improvement of the footstep planning framework. Many of them are already in preparation and will be available freely at GitHub. We are generally focused currently on improving performance and efficiency of the planner.

The basic footstep planner provides further opportunity for improvement. In future work we would like to see the ability of adaptive level-of-detail planning similar to what is described by Hornung et. al. in their paper[55]. This approach enables the planner to automatically switch the level of planning detail depending on the perceived environment. In our case the planner may use pattern generation on flat surfaces in the absence of any obstacles and then switch over to 3D planning when difficult terrain has to be traversed. This promises more efficient planning and should take away switching parameter sets from the operator.

It is desirable to improve the world modeling continuously as the performance of the footstep planner highly depends on world model quality. In general, methods should be investigated in order to increases robustness against noisy sensor data and obstructed perception. In certain cases it is also desirable to detect new features like grip of the surface. This ability can prevent the planner to plan over slippery terrain or at least consider it for feasible foot placements and therefore reduces errors in execution and possibility of falls. This challenge has been already encouraged by the VRC but not in any following competition.

Independent of any slippery terrain, placement errors can occur anytime during footstep execution. In this case the planner should be able to quickly deliver an adjusted sequence of footsteps in order to compensate for drift with respect to the underlying surface. This also leads to the question if it is possible to use the placement error as feedback for the footstep planning system in order to adapt the planning policy. We already investigated the option of using Gaussian Process Regression learning but it was shown to be unsuitable for our purposes [4]. Therefore, it is still an open topic how to adapt planning policies efficiently and how to automatically identify the constraints of the walking controller.

It took a lot of time to tune all parameters for good planning performance. Many experiments were required to determine the limits of the walking controller and even more experiments to discover all special cases. This motivates the investigation of intelligent approaches for identification and adaption of parameters for a given walking controller.

The development of the footstep planning framework is ongoing. As mentioned in one of the previous paragraphs, plugins must be instantiated hard-coded by a customized footstep planning node. This flaw will be removed in the upcoming version of the plugin manager. After this update, plugins can be instantiated just by using configuration files and additionally can be managed using a graphical user interface. Afterwards the next development milestone will be the support of collections of plugins. This allows the operator to replace multiple plugins at once and ensures that a predefined set of plugins is active. The behavior of the planner can thus be changed dynamically, allowing higher flexibility than a parameter system.

Currently there is no hardware interface provided by the framework. In future work the interfaces of walking controllers may be compared and a common interface extracted. Based on this evaluation it might be possible to provide at least a hardware interface skeleton which should support the migration of the footstep planning framework.

---

[55] Hornung, A. "Adaptive Level-of-Detail Planning for Efficient Humanoid ..." 2012. <http://ieeexplore.ieee.org/iel5/6215071/6224548/06224898.pdf?arnumber=6224898>

F.5. Humanoids 2014 Paper on Locomotion Planning [4]

This is a preprint of a paper that appeared in the Proceedings of IEEE-RAS International Conference on Humanoid Robots, pp. 287-294, Nov 18-10, 2014

Supervised Footstep Planning for Humanoid Robots in Rough Terrain Tasks using a Black Box Walking Controller

Alexander Stumpf*, Stefan Kohlbrecher*, David C. Conner† and Oskar von Stryk*

***Abstract*— In recent years, the numbers of life-size humanoids as well as their mobility capabilities have steadily grown. Stable walking motion and control for humanoid robots are already well investigated research topics. This raises the question how navigation problems in complex and unstructured environments can be solved utilizing a given black box walking controller with proper perception and modeling of the environment provided. In this paper we present a complete system for supervised footstep planning including perception, world modeling, 3D planner and operator interface to enable a humanoid robot to perform sequences of steps to traverse uneven terrain. A proper height map and surface normal estimation are directly obtained from point cloud data. A search-based planning approach (ARA*) is extended to sequences of footsteps in full 3D space (6 DoF). The planner utilizes a black box walking controller without knowledge of its implementation details. Results are presented for an Atlas humanoid robot during participation of Team ViGIR in the 2013 DARPA Robotics Challenge Trials.**

I. Introduction

There has been significant progress in humanoid robotics research in recent years. Advances both in hard- and software raise the question how humanoids can be used to assist or replace humans in hazardous environments. Motivated in part by the lack of suitable robotic response technologies at the Fukushima Daiichi nuclear disaster, the DARPA Robotics Challenge (DRC)[1] for disaster response scenarios aims at answering this question. A crucial capability for the use of humanoids in disaster environments is the ability to traverse different types of terrain in harsh environments. Examples of such terrain are uneven ground, narrow doorways and ladders, all of which are also part of the capabilities that are tested within the DRC. The first evaluation of the participating teams was done at the DRC Trials in December 2013 with real robots where the presented approach was already used.

The unpredictability of disaster scenarios implies that they are among the robotic applications that benefit the most from the cognitive abilities of a human operator. For this reason, an approach that leverages the complementary abilities of a human operator and a robotic system is likely to perform best in such situations.

Towards this end, we present a footstep planning system that allows supervised locomotion and navigation in uneven environments including perception, world modeling, 3D planning and operator interaction. The system uses the perception system of the robot to generate a 3D world model and 3D step sequences. Operators can thus select target poses for the robot system without having to care about terrain geometry or even single footstep placement. The implementation of the presented planning system is part of the Team ViGIR's approach [1] to the DRC Trials 2013.

*Department of Computer Science, TU Darmstadt stumpf,kohlbrecher,stryk@sim.tu-darmstadt.de
†TORC Robotics conner@torcrobotics.com

[1] http://therohoticschallenge.org

II. Related Work

Humanoid locomotion is still a challenging task even though walking capabilities of humanoid robots have increased significantly in the recent years. While walking over flat surfaces is relatively easy, the requirements for unstructured terrain are much higher. The walking controller has to face many external disturbance and the robot is more likely to slip on non-flat surfaces. In [2][3] this issue is addressed by implementing dynamic pattern generators. However, both implementations do not cover navigation. This problem might be mitigated by using a 3D planner that provides suitable foot positions for the walking controller to follow. In this way the walking controller can be informed about the underlying terrain in advance.

The *Covariant Hamiltonian Optimization for Motion Planning* (CHOMP) [4] is a trajectory planner which optimizes a given initial trajectory with respect to a cost function and avoiding collisions. Another approach by Schulman et al. [5] uses sequential convex optimization for optimal trajectory generation. For this purpose each criterion, such as ZMP-stability, can be modeled as as a constraint for optimization. All approaches are capable to solve rough terrain tasks with respect to robot kinematics, but they need an accurate robot model including kinematics and masses and solving the optimization problem may be computationally expensive. Furthermore they can not use existing walking controllers well because they overfit the solution towards a given robot model, which is likely not the same as the used model by the existing black box walking controllers.

Another category are contact-before-motion approaches which decouple the planning and motion layer. Bouyarmane et. al. introduce a multi-contact-planning approach in [6][7] which is designed to be used with whole-body-control (WBC). This approach generates a sequence of contact configurations which have to be executed by the WBC. It requires significant computational resources and so far is not feasible for use in tasks such as encountered in the DRC, as they do not allow for extended computation time.

As human resources and development time during the DRC competition are very limited, we require an approach which can be integrated in our systems easily and uses existing software. Thus, it should have a Robot Operation System (ROS)[2] interface and be able utilize existing walking controllers. Finally we decided to use a graph-based footstep planner similar to one that was already used successfully for Asimo in the past [8]. This approach generates a sequence of footsteps which can be used by walking controllers. Hornung et. al. [9][10] implemented an open-source version which is available for ROS, enabling quick integration in our system.

In [11][12] a first extension of previous implementation of Hornung et. al. footstep planner [9][10] is shown. In [12] they introduce how to generate a height map online using an onboard stereo camera system and a collision checking method using pre-generated inverse height maps of each action. Although they generate a 3D world model, their approach is only extended by special actions for step up and step down motions. This enables 3D planning on flat ground using predefined actions but is not sufficient for real world application due to missing capability for planning on sloped or random terrain.

Another open question is how to use the robot walking controller capabilities in rough terrain tasks. This question was addressed by the *Learning Locomotion Project* hosted by DARPA. All participating teams implemented similar approaches using a graph-based planner with learned or optimized cost functions [13][14][15][16] which shows that learning cost functions is a valid option.

In contrast to search-based planning approaches, MIT recently presented their footstep planning system [17] which generates a footstep plan by continuous optimization. For this purpose they implemented a mixed-integer quadratically-constrained quadratic program (MIQCQP). They demonstrate that their planning system is able to solve complex terrain scenarios but world modeling itself is still a weakness in their approach. In the presented work the operator has to define obstacle free regions manually by placing polygons.

Summarizing, only a few authors address terrain modeling in their work e.g. generation of height maps from stereo camera data [18][12]. Most approaches are based on external motion tracking systems or assume a known world model. This aspect may not be neglected if the robot should walk in real-world environments, which motivates our work.

III. Experimental Platform

The proposed approach is designed for humanoid robots providing a walking controller. Thus, the presented work is evaluated with the Atlas robot without limitation to generality. Atlas is a 1.88m tall with a weight of 150kg near anthropomorphic robot developed by Boston Dynamics Inc. (BDI) as the direct successor to PETMAN [19]. The main external sensor is a Carnegie Robotics Multisense SL sensor mounted as the head. This sensor uses both a Hokuyo UTM-30LX-EW LIDAR mounted on a slip ring for continuous rotation and a stereo camera system. Furthermore the robot provides a pose estimate based on a internal IMU and internal joint sensing.

[2]`http://www.ros.org/`

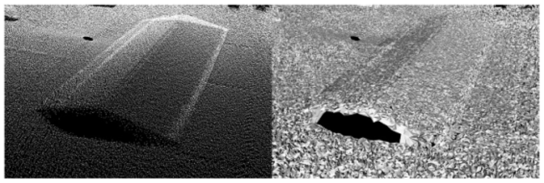
The robot is delivered with a proprietary walking controller developed by BDI. The closed source "BDI Walking/Stepping Behaviors" provide several controller modes including a quasi-statically stable stepping mode as well as dynamically stable walk mode. From our perspective, the BDI Walking Behavior is a pure black box; this emphasizes the generality of our footstep planning approach.

IV. World Modeling

To our knowledge graph-based planners have nearly only be used for planning in flat environments. In contrast 3D planning needs the ability to estimate the full 3D state of each placed foot. This concerns in particular the roll, pitch and height of the foot pose which are not needed for 2D planning approaches.

Thus, a suitable world model is essential for reliable footstep planning in rough terrain scenarios and has to provide all required information in an efficient way. Although Atlas is equipped with multiple sensor systems, only the LIDAR scanner is a suitable for large range scans providing accurate terrain data for long distance planning. For this purpose a processing pipeline was implemented to generate a suitable word model for rough terrain planning. The first step is to aggregate each single LIDAR scan into a 3D point cloud given an aggregation horizon in seconds. In our case we use only data which was collected since the robot has not moved due to missing robot state estimation and correction.

Afterwards, noise is filtered out from the resulting point cloud by applying the *Voxel Grid* and *Moving Least Square* filters of the *Point Cloud Library* (PCL)[3]. Additionally, a *Pass Through* filter is used to reduce the computational expense by cropping the region of interest. All parameters of the filtering process were adjusted offline by comparing the resulting surface reconstruction of the *Greedy Projection* approach which is provided by PCL. Figure 1 emphasizes the difference between noisy and filtered point clouds.


(a) Noisy point cloud and 3D surface reconstruction due to missing filtering

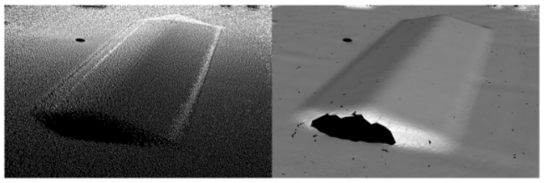

(b) Filtered point cloud and 3D surface reconstruction

Fig. 1: Examples for 3D surface reconstruction

[3]`http://www.pointclouds.org/`

Given the smoothed point cloud we can extract the height and normal of the surface. The height map is obtained easily by storing the z-component of each point of the cloud in a discretized 2D grid map. Each cell of the height map contains the highest value discovered in the point cloud. Unfortunately the height map contains many small holes due to sparse LIDAR data. This issue may let the planner fail unnecessarily due to missing height information. Therefore, missing height information $\hat{p}_z$ is estimated by the weighted average of the point's $\hat{p} = (\hat{p}_x, \hat{p}_y, \hat{p}_z)^T$ k-nearest neighbors $p^{(i)} = (p_x^{(i)}, p_y^{(i)}, p_z^{(i)})^T$ in the point cloud:

$$\hat{p}_z = \frac{1}{k}\sum_{i=1}^{k} p_z^{(i)} \cdot \left(1 - \frac{\left\| \begin{pmatrix} p_x^{(i)} \\ p_y^{(i)} \end{pmatrix} - \begin{pmatrix} \hat{p}_x \\ \hat{p}_y \end{pmatrix} \right\|^2}{\sum_{j=1}^{k} \left\| \begin{pmatrix} p_x^{(j)} \\ p_y^{(j)} \end{pmatrix} - \begin{pmatrix} \hat{p}_x \\ \hat{p}_y \end{pmatrix} \right\|^2} \right) \quad (1)$$

which is demonstrated in figure 2.

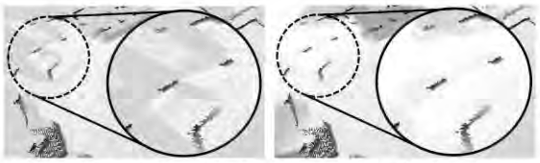

(a) Cluttered pixels indicate missing height information (b) Homogeneous height data is now available due to estimation

Fig. 2: Example for gap filling in a height map

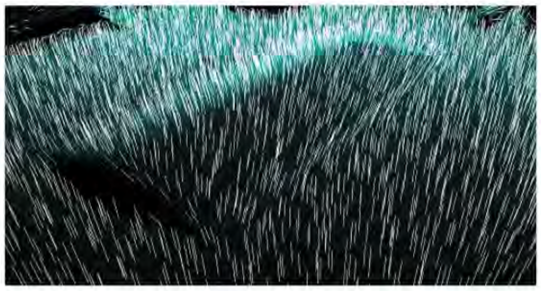


Fig. 3: Resulting normal estimation of a pitch ramp using PCA-based approach: The white lines visualize the estimated normals while the underlying point cloud consists of the turquois points.

The normal is extracted by applying a Principle Component Analysis (PCA)[4] based approach from PCL on the neighborhood of the query point. This operation is performed on every point of the cloud which is surrounded by a sufficient amount of neighbors in a given distance. As the solution for the direction is not unique, all normals are converted to point upright relative to gravity (see figure 3).

[4]`http://pointclouds.org/documentation/tutorials/normal_estimation.php`

Given a target position and orientation the normal can be obtained and used to determine the roll and pitch angle for this query point. In order to determine a normal for any query point efficiently, all of them are saved in a k-d-tree, which is already implemented in PCL. In our case we use the position as query key and the normal is resolved with $O(\log n)$ in average.

For simple body collision checks the LIDAR scans are additionally used to generate an octomap [20] which results in a 3D world representation composed of cubes. This octomap is used to generate 2D occupancy grid maps by projecting a slice of the octomap.

V. Footstep Planning

Our novel approach efficiently generates suitable plans for rough terrain tasks using existing black box walking controllers without deep knowledge about their implementation that requires decoupling of planning and execution layers. For this purpose the planner must be able to determine feasible paths with the given world model as fast as possible to operate the robot efficiently.

The presented work is based on Hornung's footstep planning framework [9][10][11]. In this paper we want to highlight the major extensions and all necessary steps to perform footstep planning for rough terrain tasks under real-world applications. Finally, the approach is evaluated with the Atlas robot but may be easily transferred to general classes of humanoid robots providing a walking controller.

The basic footstep planner by Hornung et. al. defines the state $\boldsymbol{s}$ as the pose of the foot $\boldsymbol{s} = (x, y, \theta, f)$, where $\{x, y\}$ denote the 2D-position, θ the orientation and $f \in \{left, right\}$ the corresponding foot. All successor states $\boldsymbol{s'}$ of $\boldsymbol{s}$ are generated by applying the transition function $t(\boldsymbol{s}, a)$ where $a \in A$ are actions describing the displacement vector $(\Delta x, \Delta y, \Delta\theta, f)$ and f denotes the supporting foot. Therefore the successor state $\boldsymbol{s'}$ is simply derived by:

$$\boldsymbol{s'} = t(\boldsymbol{s}, a) = \boldsymbol{s} \overset{a}{\mapsto} \boldsymbol{s'} \quad (2)$$

The set A consists of all possible actions a which are also denoted as "footstep primitives". For simplicity we omit the foot f in all following representations as we assume that the planner will generate an alternative sequence of footsteps and every action can be mirrored in following way: $(\Delta x, \Delta y, \Delta\theta, left) = (\Delta x, -\Delta y, -\Delta\theta, right)$.

Finally this representation is used to generate a graph with states as nodes and actions as transition condition between the nodes. Here, the Anytime A* (ARA*) [21] algorithm is used to determine the shortest path from start to goal which is transformed to a sequence of footsteps afterwards.

A. States, Actions and Transition Model

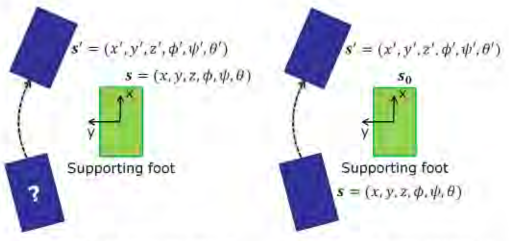
The key parts of graph-based planning are the states, actions and the transition model. These components have to be adapted in a way they become suitable for rough terrain tasks requiring full 3D planning space.

1) States: Rough terrain traveling requires a 3D state space representation. Thus, the original approach was extended by z, ϕ (roll) and ψ (pitch) to the 3D state space $\boldsymbol{s} = (x, y, z, \phi, \psi, \theta) = (s_x, s_y, s_z, s_\phi, s_\psi, s_\theta)$. Fortunately, these new components are constrained by the underlying surface and can easily be obtained by the generated height map and normals as described in section IV. For this purpose the corresponding normal $\boldsymbol{n} = (n_x, n_y, n_z)^T$ has to be converted to Euler angles given the target orientation θ of the foot:

$$\phi = -\arcsin(n_x \sin(-\theta) + n_y \cos(-\theta)) \quad (3)$$
$$\psi = \arcsin(n_x \cos(-\theta) - n_y \sin(-\theta)) \quad (4)$$

which fits a plane tangential to the surface at the query point. The Euler angle representation simplifies the comprehension and design of cost functions. Although the extended state space now has six DoF, the search space itself remains at three DoF. For this reason the computational effort for searching an optimal solution is not increased by this modification, provided the world model can be queried efficiently.

With the growth of larger, more complex humanoid robots, consideration of dynamics has become more important and cannot be neglected for the Atlas robot. For this reason the cost functions were extended to evaluate the cost $c(\boldsymbol{s}, \boldsymbol{s}')$ for full step representation (see Figure 4b) instead of the original half step representation (see Figure 4a) where the origin of the swing foot is unknown. This allows us to consider the dynamic behavior estimate of the robot executing this step sequence. We define a new representation for step sequences consisting solely of the start state $\boldsymbol{s}$ and goal state $\boldsymbol{s'}$ of the moving foot which are given relative to supporting foot $\boldsymbol{s_0}$. This kind of representation enables a unique representation for any possible step, and permits the use of lookup tables instead of computational expensive function evaluations, to reduce planning time significantly.


$s' = (x', y', z', \phi', \psi', \theta')$
$s = (x, y, z, \phi, \psi, \theta)$
x
y
Supporting foot
?
$s' = (x', y', z', \phi', \psi', \theta')$
s_0
x
y
Supporting foot
$s = (x, y, z, \phi, \psi, \theta)$

(a) half step representation (b) full step representation

Fig. 4: Modification of step representation

2) Actions: Although the original approach was designed for small humanoid robots walking on flat ground, the search space remains the same (see Section V-A.1). Thus, an action a is still defined as displacement vector $(\Delta x, \Delta y, \Delta\theta)$. Rough terrain tasks demand higher accuracy in planning level due to unstructured debris on the floor. This problem is further exacerbated by larger humanoid robots whose reachable region $\mathcal{R}$ for a single step is significantly increased. Therefore, the action set A must be increased as well to maintain a sufficient variation of discrete actions and accuracy for crossing rough terrain. For this reason it is not feasible to extend the set A manually like the authors in [12] do. Thus, we define a reachability polygon $\mathcal{R}$ next to the supporting foot $\boldsymbol{s_0}$ from which the footstep primitive set A is sampled in a discrete way.

B. Cost Functions and Heuristics

The cost function used in combination with a heuristic has the maximum impact on the planner solution [10]. This makes designing cost functions and heuristics the most important and challenging task.

1) Cost functions: As we do not have any detailed insight into the robot's walking controller (here the *BDI Walk Behavior*) all cost functions can only be designed by systematic experiments, expertise or machine learning approaches. Besides the robot capabilities basic properties like minimal number of steps or shortest path has to be considered by cost functions too. Additionally, life-size humanoid robots have versatile walking capabilities which should be utilized in rough terrain, so the footstep planner has to minimize multiple, possibly competing costs e.g. shortest path and minimal fall risk. For this reason a hierarchical system of cost functions was implemented, dividing each criterion in a different layer which is finally conquered individually by a specific cost function. This design was realized by using the Decorator Pattern [22], enabling the composition of cost functions in a flexible manner.

In this section we will only introduce briefly the implemented cost functions.

- *Constant*: This functions adds the constant value c_{step} on top of all cost regardless of the given step. Using this function enforces the planner to generate a step-minimal solution.
- *Euclidean*: The euclidean cost function computes the 2D distance traversed by the robot's torso. This way the planner is enforced to find the shortest path.
- *GPR*: This cost function wraps a *Gaussian Progress Regression* implementation estimating step cost by previously collected and offline learned observation.
- *Map*: In Section V-A.1 we have pointed out that instead of evaluation of computational expensive cost function a simple lookup table may be used which is implemented by this cost function,
- *Boundary*: This cost function implements the robot specific capabilities and invalidates all step sequences which can probably not performed by the robot. This cost function was determined by systematic experiments with the real robot to discover the limits of the system which are tightened to decrease the risk of failure. In sum, this cost function penalize violation of these tightened limits and has the most influence on $\hat{A}(\boldsymbol{s})$.
- *Dynamics*: The dynamic behavior e.g. accelerations are modeled as additional cost and prevents high accelerations of the robot torso.

- *Ground Contact*: This cost function utilizes the ground contact estimation (see Section V-C) and adds artificial cost, when the given step position does not fit the underlying terrain perfectly.

In the current state following hierarchy is used (top-down): *Constant*, *Euclidean*, *Boundary*, *Dynamics* and *Ground Contact*. Finally the accumulated cost represents the effort to execute the step sequence.

2) Risk Measurement: The reachability polygon $\mathcal{P}$ introduced in Section V-A.2 implies the risk of including bad footstep placements. For this reason we introduce in our novel approach a quantity denoted as risk to distinguish between effort (cost) and feasibility (risk) of a step. Merging risk and cost in a single value would not prevent the planner from using bad steps in a plan when only a small number of step options are available. Furthermore, risk evaluation allows to generate dynamically a subset of valid actions $\tilde{A}(s) \subseteq A$ depending on current state s which improves the quality of generated plans. Analogous to evaluation of cost, the risk is simultaneously computed in every layer. Thus, feasibility of a step can now be determined based on the accumulated risk while the effort to execute this step is represented solely by the cost.

3) Heuristic: Finding the global best solution is not guaranteed by using the ARA*-planner in *anytime* mode. While the cost function defines the shape of the resulting (local) cost-minimal plan, the heuristic influences which local minimum is found by the planner; thus, the heuristic may not be neglected. The original heuristics uses

$$h(\boldsymbol{s}) = \|\boldsymbol{s} - \boldsymbol{s_{goal}}\| + c_\theta \cdot |\Delta\theta| + c_{step} \cdot \frac{\|\boldsymbol{s} - \boldsymbol{s_{goal}}\|}{d_{max}} \quad (5)$$

to estimate remaining cost with d_{max} as maximal step distance, $\Delta\theta$ the shortest angle between θ, θ_{goal} and c_θ, c_{step} as predefined cost per angular difference or step. In combination with the cost functions defined in Section V-B.1 the planner has a poor planning performance (see figure 5a), because the heuristic does not take into account that the maximum step distance d_{max} should depend on the direction of movement. Hence, we distinguish between transversal Δx_{max} and lateral Δy_{max} step distance limits leading to following modified heuristic:

$$n_{steps} = \lfloor \frac{\Delta\hat{x}}{\Delta x_{max}} + \frac{\Delta\hat{y}}{\Delta y_{max}} \rfloor \quad (6)$$

$$\hat{h}(\boldsymbol{s}) = \|\boldsymbol{s} - \boldsymbol{s_{goal}}\| + c_\theta \cdot |\Delta\theta| + c_{step} \cdot n_{steps}, \quad (7)$$

where $\Delta\hat{x}$ and $\Delta\hat{y}$ denote the needed transversal and lateral movement distance without using any rotational movement. Thus, n_{steps} is just an estimation of minimal number of total transversal and lateral steps needed to reach the goal. In usual cases $\Delta x_{max} > \Delta y_{max}$ holds which means the robot can move faster forwards than sidewards. Thus, the modified heuristic $\hat{h}(x)$ prefers moving directly towards the goal why $\hat{h}(x)$ performs better than $h(x)$ in Figure 5b.

C. Collision Check

Rough terrain tasks demand the ability to detect obstacles and walkable surfaces and utilize these information for

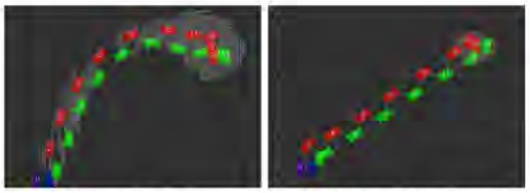

(a) Sight line euclidean step cost heuristic: 101.659 expanded states, 95s planning time (b) Modified euclidean step cost heuristic: 10.102 expanded states, 9s planning time

Fig. 5: Comparison of heuristics

navigation. Our approach is focused on solving the terrain task of the DRC Trials which has fairly sloped terrain without walls and the robot doesn't need to perform multi-contact-planning. Thus, we may assume if two sequent states are collision free then the trajectory will be collision free too.

1) Occupancy Grid Map: Hornung et. al. already implemented an efficient collision check strategy for rectangular bodies like the feet which is proposed by Sprunk et al. [23] using an occupancy grid map. They assume that all not surmountable obstacles are artificially inflated in the occupancy grid map [10] to keep the upper body collision free. But this approach is insufficient for our purposes because the robot has to traverse narrow doors. Therefore, we split collision checking into two layers: One layer is exclusively responsible for upper body and the other for feet collision check. Analogously, we extend the footstep planner to use different occupancy grid maps for each layer which are generated by slicing the octomap as described in Section IV.

2) Ground Contact Estimation: In the domain of 3D planning collision checking using 2D occupancy grid maps has the drawback to take not the terrain height into account. When using the occupancy grid map, we can not distinguish between holes and hills. Furthermore, while performing as expected, occupancy grid maps have the drawback of wasting space by encircling each obstacle with non-traversable cells. As a result, collision checks with occupancy grid maps are too strict because they enforce placing each foot avoiding obstacles completely. This results in longer paths and the ARA*-planner takes a lot of more computational time trying to find shorter solutions. Furthermore the occupancy grid map based collision checks do not allow any kind of overhang when climbing upstairs.

This motivates our new approach to use solely the height map introduced in Section IV. Given a target foot state s the undersurface of the footprint is sampled equally spaced to compute the vertical distance $\Delta z = h(s_x, s_y) - s_z$ relative to the terrain surface $h(x, y)$ which is shown in Figure 6. Each sampling point is classified by using thresholds into *collision* ($\Delta z > 0.75cm$), *overhung* ($\Delta z < -1.0cm$) or otherwise *contact*. If at least one sampling point is classified as *collision* the foot state will be treated as invalid. Otherwise the ground contact support is given as percentage of sampling points classified as *contact*. These narrow thresholds are feasible because of well filtered LIDAR data (see Section IV) and all surfaces in the competition are flat. In general the thresholds must be scaled towards data quality. The ground contact support estimate becomes worse with increasing noise in case of many overhanging points and the planner performance decreases due to many false positive collision detections. In future work we plan to use the convex hull of *contact* points as ground contact estimation that promises more robustness against false positive *overhung* classifications. Furthermore, it enables the planner to generate steps having only at least three *contact* points (e.g. stepping on three poles). An estimate of surface friction is neglected for now because all tasks are built up with slip-proofed surfaces. But friction estimation will be added when required by future competition.

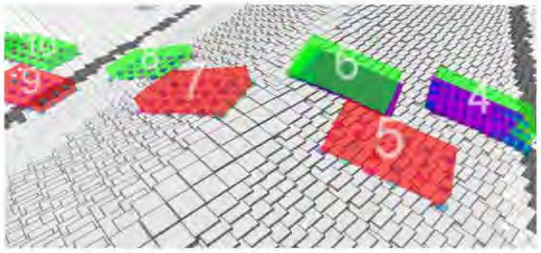


Fig. 6: This example shows the sampling points at the undersurface of the footprint. The steps 4 and 6 have been sampled in higher resolution.

The ground contact estimation superfluous the usage of discretized occupancy grid maps. Furthermore the planner is now able to plan overhanging steps which increases flexibility of foot placement. For this reason the planner is able to find and optimize solutions quicker improving significantly the resulting plan. In Figure 7 two examples of the DRC Trials are given where the ground contact support outperforms the usage of occupancy grid maps by generating shorter and straighter paths.

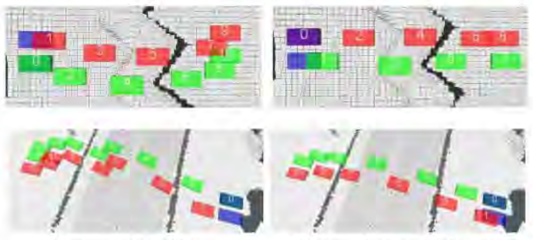


Fig. 7: Comparison of collision checking approaches: In both examples the ground contact estimation outperforms the occupancy grid map.

D. Miscellaneous modifications

1) Start foot selection: As the first step is not explicitly defined by the planning request, the planner has to select the starting foot by itself. The original footstep planner by Hornung et. al. selects always the left foot. In contrast we add the logic to select the foot which is closer to the goal pose. This behavior allows the planner to turn quicker towards the goal position which results in shorter paths.

2) Automatic goal pose refinement: The operator may define the planning goal by pose. In this case the planner itself has to define the final feet position at the end of the plan. The planner is able to autonomously project both feet on the sloped surface. If the initial feet configuration collides, the planner is able to shift slightly the feet pose to a collision free configuration within the given reachability polygon $\mathcal{R}$.

3) Post-Processing: Walking controllers are supposed to define further parameters for each step e.g. lift height. We are interested in getting as fast as possible a solution, so optimizing these parameters during planning process is not feasible due to increased dimension of search space. Thus, these parameters are improved in a post-process step after the final solution was found. In the current state we determine only the lift height by searching the maximum terrain height in sight line between origin and target pose of the step.

VI. Operator Interaction

For supervised robot operation different abstraction layers for controlling the footstep planner is available. With increasing task complexity a more detailed operator interface can be used. The most simple way to interact with the footstep planner is by moving the ghost robot to the desired goal position demonstrated in Figure 8. Alternatively, the operator may define directly a target pose by clicking on the grid map.

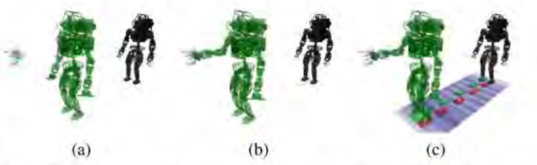


Fig. 8: Use case of footstep planning: The operator places a template (a). The ghost robot (green) will be aligned with the template (b) and the footstep plan is generated (c).[24]

For more complex task like rough terrain several widgets are available which we want to introduce briefly:

- *Parameter Widget*: This widget provides access to the major planner parameters.
- *Step Widget*: The widget provides the option to define manually single steps or whole patterns.
- *Terrain Request Widget*: 3D terrain model generation is unfortunately computational expensive; thus, it should only be generated by operator request.

Transmission bandwidth is very limited during the competition. Thus, the operator receives only a thinned out version of the point cloud consisting of a very small fraction of the original point cloud data to verify the generated footstep plan which is visualized with boxes which can be seen in Figures 6 and 7.

VII. Results

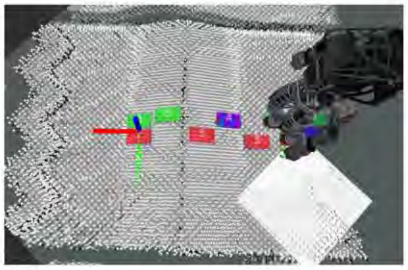


Fig. 9: OCS perspective of the ramp (DRC Trials)

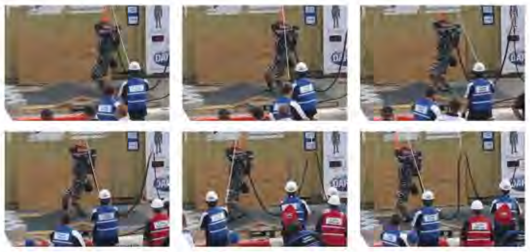


Fig. 10: Robot traversing the pitch ramp (DRC Trials)

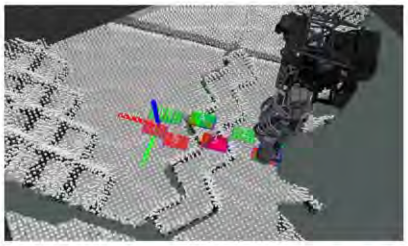


Fig. 11: OCS perspective of the chevron hurdle (DRC Trials)

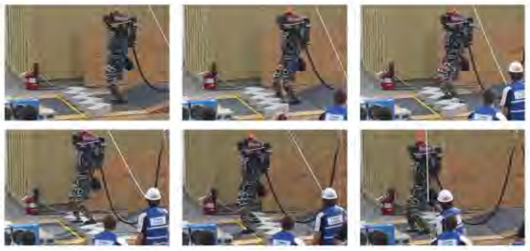


Fig. 12: Robot traversing the chevron hurdle (DRC Trials)

At an early development stage we tried to learn a suitable step cost function using Gaussian Progress Regression (GPR) for the simulated robot. For this purpose random step pattern has been generated and executed in simulation. After each experiment the step pattern was rated by successful execution or fall. In total over 40k experiments were performed in simulation whose data was used to learn a step cost function which estimates the risk of failure. In `https://www.youtube.com/watch?v=hT-_n4icg54` the footstep planner is able to utilize the black box walking controller much better compared to the manual defined footstep primitives in `https://www.youtube.com/watch?v=JMCREHzimpM`. However, it turned out that offline learning approaches like GPR takes too much effort to train. All recorded data and the learned cost functions are supposed to be invalidated every time the robot is modified, which happened often because Atlas was still in development. Thus, extensive (re)training must be performed, which is not feasible for a real robot system. Furthermore, simulation-based learning is not feasible because of significant differences between the model and real robot.

As we got only a limited developing time until the DRC Trials, we had to investigate in an approach which is easier to maintain in case of changes of the robot system or the walking controller. For this reason the final solution is to define a hierarchical step cost function determined and optimized by systematic experiments (see Section V-B.1).

This approach was successfully used during the DRC Trials in all tasks without any falls caused by the footstep planner. While the BDI walking controller provided a very stable walk, we demonstrated how to utilize this walking controller close to the limits. During the DRC Trials almost all teams using the Atlas robot with the provided BDI walking controller performed significant smaller step sizes and with it slower travel speed.

The terrain task was the supreme task for locomotion where the planner performed well. It took only a few minutes and very few interaction steps by the operator to cross the pitch ramp and the chevron hurdle. The resulting execution is summarized in the Figures 10 and 12 while the corresponding perspective of the operator control station which contains the 3D world model and footstep plan is showed in the Figures 9 and 11. We also provide videos of stepping over the pitch and the chevron hurdle at `https://www.youtube.com/watch?v=7Qv__bLa3j4` and `https://www.youtube.com/watch?v=vAtqVKGWvFM`.

In general we are able to generate plans on flat surfaces within a few seconds using a single core of a Core i7 computer. Going into 3D space, the initial solution is found within a few seconds too, but the improvement of the plan takes around ten seconds depending on terrain complexity. The terrain model generation time depends on the selected size of the region of interest; but at the DRC Trials it took a maximum of one second for a region of the size which is visible in Figures 9 and 11. The plan for the pitch ramp in Figure 9 could be generated online in 0.7 seconds and for the chevron hurdle in Figure 11 in 1.8 seconds. To our knowledge almost all participating teams haven't performed the terrain task by using a high-level footstep planner. Instead each step was defined by the operator manually which takes obviously more planning time.

As discussed in Section IV the footstep planner is also required to navigate through narrow doorways. The video at `https://www.youtube.com/watch?v=BlUfl5iSAkU` shows the successful attempt of the robot walking autonomously through a very small doorway without any collisions using our footstep planner.

VIII. Conclusion

In this work a complete supervised 3D footstep planning system covering perception, world modeling, full 3D (6 DoF) planning and operator interaction was introduced. We demonstrated a graph-based footstep planning approach utilizing an existing black box walking controller to generate whole sequences of steps in rough terrain scenarios. The 3D state of each step is determined by the planner automatically. Thus, the operator has not to define manually each step to be executed which increases the operator efficiency and mission performance.

Many features like the used full step representation with reachability polygon as well as the novel ground contact estimation based on height map and normal estimation improve the planner performance in uneven terrain significantly. The generated terrain model provides sufficient data to perform 3D footstep planning in unstructured environments and a compressed version can be sent to the operator station to provide situational awareness. Furthermore, we described a couple of user interfaces that were implemented to provide flexible footstep planning control given in different abstraction layers.

The first simulation-based experiments show a proof of concept for learning cost functions for black box walking controller. However, it turned out that the investigated offline learning approaches are not feasible for robots which are still in development due to required excessive training.

In future work we would like to improve planning results by generating the reachability polygon (see Section V-A.2) online by using inverse kinematics and respecting in parallel the center of mass dynamics including all carried objects. Experience has shown us that the terrain model is crucial for reliable 3D footstep planning. For this reason we will investigate how to improve the model quality, while reducing the computational expense. The effectiveness of the supervising operator depends on the provided options to interact with the footstep planner. Therefore we are going to implement an interactive footstep planner interface which will allow the operator to fine tune the planned steps. The advanced planner will also incorporate plan stitching to allow more complex and flexible "human-in-the-loop" planning which mitigates the lack of intelligence of the planning system.

IX. Acknowledgment

The authors would like to thank all members of Team ViGIR for their contribution and support which enabled the realization of this work. This work has been funded by DFG Research Training Group 1362 and Defense Advanced Research Projects Agency (DARPA) under Air Force Research Lab (AFRL) contract FA8750-12-C-0337; the views expressed in this paper are those of the authors.

References

[1] Kohlbrecher, S., Romay, A., Stumpf, A., Gupta, A., von Stryk, O., Bacim, F., Bowman, D., Goins, A., Balasubramanian, R., Conner, D.: Human-robot teaming for rescue missions: Team ViGIR's approach to the 2013 DARPA Robotics Challenge Trials. Journal of Field Robotics (to appear)

[2] Hirukawa, H., Hattori, S., Kajita, S., Harada, K., Kaneko, K., Kanehiro, F., Morisawa, M., Nakaoka, S.: A pattern generator of humanoid robots walking on a rough terrain. In: IEEE Intl. Conf. on Robotics and Automation. (2007) 2181–2187

[3] Takubo, T., Imada, Y., Ohara, K., Mae, Y., Arai, T.: Rough terrain walking for bipedal robot by using ZMP criteria map. In: IEEE Intl. Conf. on Robotics and Automation. (2009) 788–793

[4] Ratliff, N., Zucker, M., Bagnell, J.A., Srinivasa, S.: CHOMP: Gradient optimization techniques for efficient motion planning. In: IEEE Intl. Conf. on Robotics and Automation. (2009) 489–494

[5] Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., Abbeel, P.: Finding locally optimal, collision-free trajectories with sequential convex optimization. In: Robotics: Science and Systems. (2013)

[6] Bouyarmane, K., Kheddar, A.: Multi-contact stances planning for multiple agents. In: IEEE ICRA. (2011) 5246–5253

[7] Bouyarmane, K., Vaillant, J., Keith, F., Kheddar, A.: Exploring humanoid robots locomotion capabilities in virtual disaster response scenarios. In: IEEE-RAS HUMANOIDS. (2012) 337–342

[8] Chestnutt, J., Lau, M., Cheung, G., Kuffner, J., Hodgins, J., Kanade, T.: Footstep planning for the honda ASIMO humanoid. In: IEEE Intl. Conf. on Robotics and Automation. (2005) 629–634

[9] Garimort, J., Hornung, A.: Humanoid navigation with dynamic footstep plans. In: IEEE ICRA. (2011) 3982–3987

[10] Hornung, A., Dornbush, A., Likhachev, M., Bennewitz, M.: Anytime search-based footstep planning with suboptimality bounds. In: IEEE-RAS Intl. Conf. on Humanoid Robots. (2012) 674–679

[11] Hornung, A., Maier, D., Bennewitz, M.: Search-based footstep planning. In: ICRA Workshop Progress & Open Problems in Motion Planning & Navigation for Humanoids, Karlsruhe, Germany. (2013)

[12] Maier, D., Lutz, C., Bennewitz, M.: Integrated perception, mapping, and footstep planning for humanoid navigation among 3d obstacles. In: IEEE/RSJ IROS. (2013) 2658–2664

[13] Kalakrishnan, M., Buchli, J., Pastor, P., Schaal, S.: Learning locomotion over rough terrain using terrain templates. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems. (2009) 167–172

[14] Kolter, J.Z., Rodgers, M.P., Ng, A.Y.: A control architecture for quadruped locomotion over rough terrain. In: IEEE Intl. Conf. on Robotics and Automation. (2008) 811–818

[15] Parlaktuna, O., Ozkan, M.: Adaptive control of free-floating space manipulators using dynamically equivalent manipulator model. Robotics and Autonomous Systems **46**(3) (2004) 185–193

[16] Zucker, M., Bagnell, J.A., Atkeson, C.G., Kuffner, J.: An optimization approach to rough terrain locomotion. In: IEEE Intl. Conf. on Robotics and Automation. (2010) 3589–3595

[17] Deits, R., Tedrake, R.: Footstep planning on uneven terrain with mixed-integer convex optimization. In: IEEE-RAS Intl. Conf. on Humanoid Robots. (2014) to appear

[18] Chilian, A., Hirschmuller, H.: Stereo camera based navigation of mobile robots on rough terrain. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems. (2009) 4571–4576

[19] Nelson, G., Saunders, A., Neville, N., Swilling, B., Bondaryk, J., Billings, D., Lee, C., Playter, R., Raibert, M.: Petman: A humanoid robot for testing chemical protective clothing. Advanced Robotics **30**(4) (2012) 372–377

[20] Wurm, K.M., Hornung, A., Bennewitz, M., Stachniss, C., Burgard, W.: OctoMap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In: ICRA Workshop best practice in 3D perception & modeling for mobile manipulation. Volume 2. (2010)

[21] Likhachev, M., Gordon, G.J., Thrun, S.: ARA*: Anytime a* with provable bounds on sub-optimality. In: NIPS. (2003)

[22] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Pearson Ed. (1994)

[23] Sprunk, C., Lau, B., Pfaffz, P., Burgard, W.: Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In: IEEE Intl. Conf. on Robotics and Automation. (2011) 72–77

[24] Romay, A., Kohlbrecher, S., Stumpf, A., von Stryk, O.: Template-based manipulation for supervised semi-autonomous humanoid robots. In: IEEE-RAS Intl. Conf. on Humanoid Robots. (2014) to appear

G. BEHAVIOR EXECUTIVE SYSTEM

This appendix presents the details of FlexBE, Team ViGIR's behavior engine and high-level executive. In addition to the behavior engine, which acts as a "back-end", the appendix presents FlexBE's graphical user interface (GUI), which serves as a "front-end" to the Behaviors subsystem. The text is from Chapters 3 through 5 of [7], which is available online in its entirety[56].

Chapter 3 focuses on the underlying concepts and discusses the theoretical background in an abstract manner. After summarizing the available basis provided by previous work in a uniform way, concepts regarding operator interaction and runtime modifications are added on top. Finally, consequences for behavior development are discussed. Chapters 4 and 5, presents various aspects of the implemented software based on the developed concepts.

Chapter 4 targets the onboard behavior engine and shows how execution of behaviors is solved by FlexBE and how the process of behavior switching during runtime is integrated. Subsequently, after an initial discussion regarding the detailed approach specific to the user interface, chapter 5 presents the behavior control system, including code generation of behaviors and controlling their execution.

[56] http://www.sim.informatik.tu-darmstadt.de/publ/da/2015_Schillinger_MA.pdf (accessed August 11, 2015)

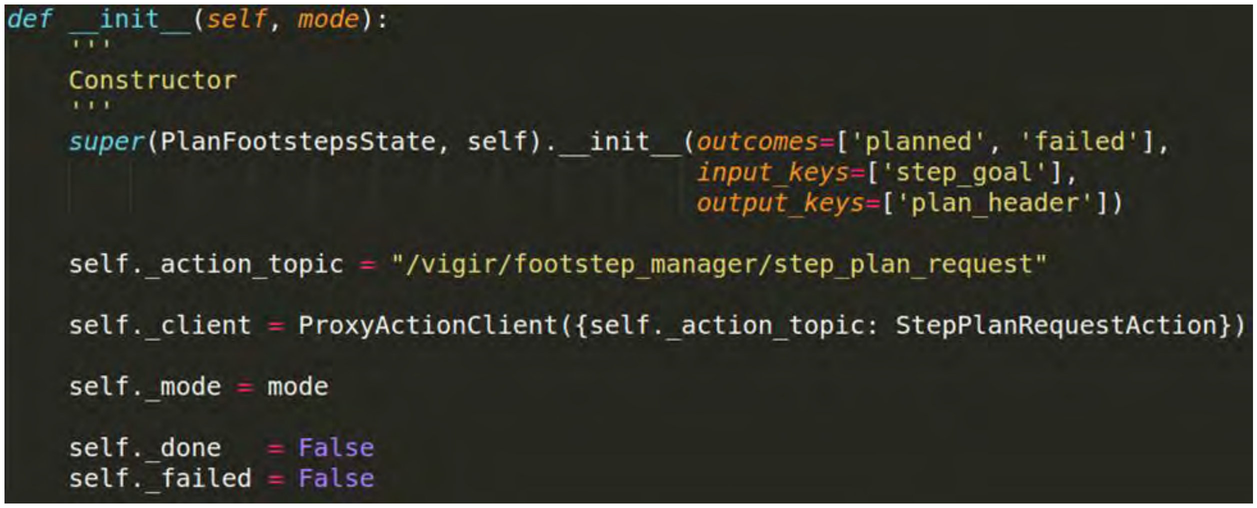
This page intentionally blank.

H. BEHAVIOR EXAMPLES

This appendix presents details of the construction of states used in the behaviors presented in Section 4.2.4.

H.1. State Details

In the following section we will enumerate all states that were included in a behavior that was used during the DRC Finals or the experimental demonstrations in the lab. Since the list will not include any details, we will first present the inner working of one state, PlanFootstepsState, which is representative of a large class of states that interface with a ROS action server. Figure 75 shows the python constructor for the state's class definition.

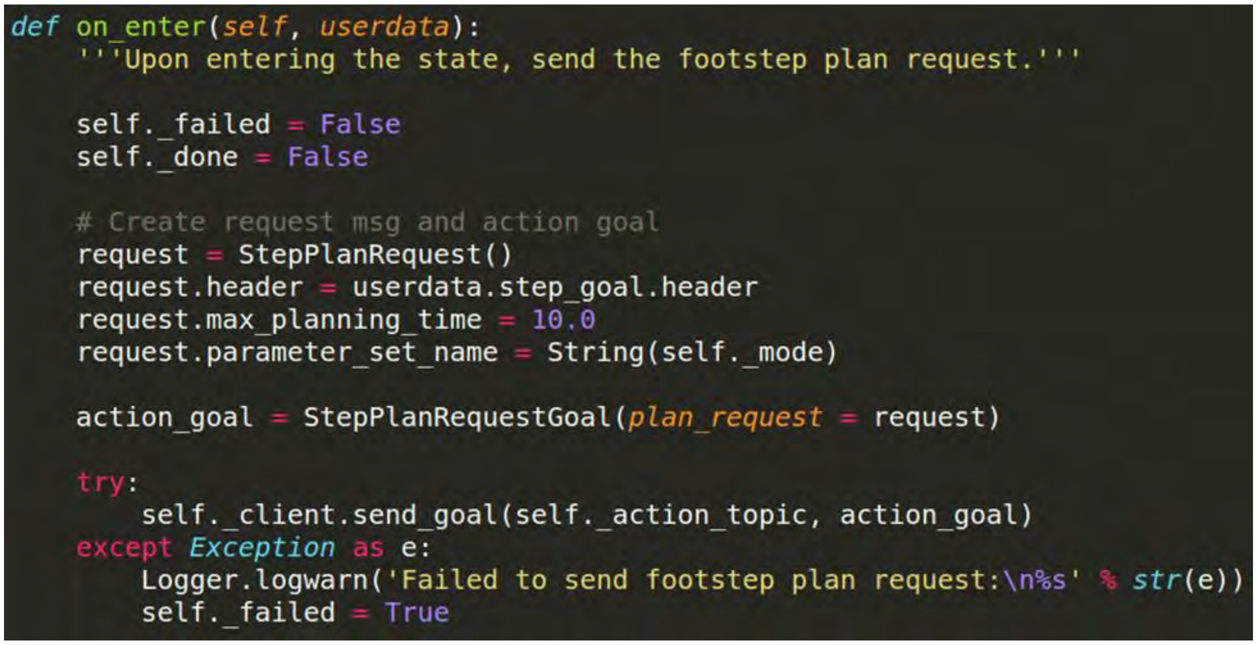


```
def __init__(self, mode):
    '''
    Constructor
    '''
    super(PlanFootstepsState, self).__init__(outcomes=['planned', 'failed'],
                                              input_keys=['step_goal'],
                                              output_keys=['plan_header'])

    self._action_topic = "/vigir/footstep_manager/step_plan_request"

    self._client = ProxyActionClient({self._action_topic: StepPlanRequestAction})

    self._mode = mode

    self._done   = False
    self._failed = False
```

Figure 75. The PlanFootstepsState's constructor.

This is where the outcomes, input keys, and output keys are defined for all states. In this example, the constructor handles the initialization of an action client that will later send footstep plan requests to the onboard footstep manager (which will in turn contact the onboard footstep planner). As with all states, the attributes (done, failed) that correspond to the two outcomes are initialized.

Figure 76 shows the python code for the state's on_enter method that is responsible for initializing the state before each execution.

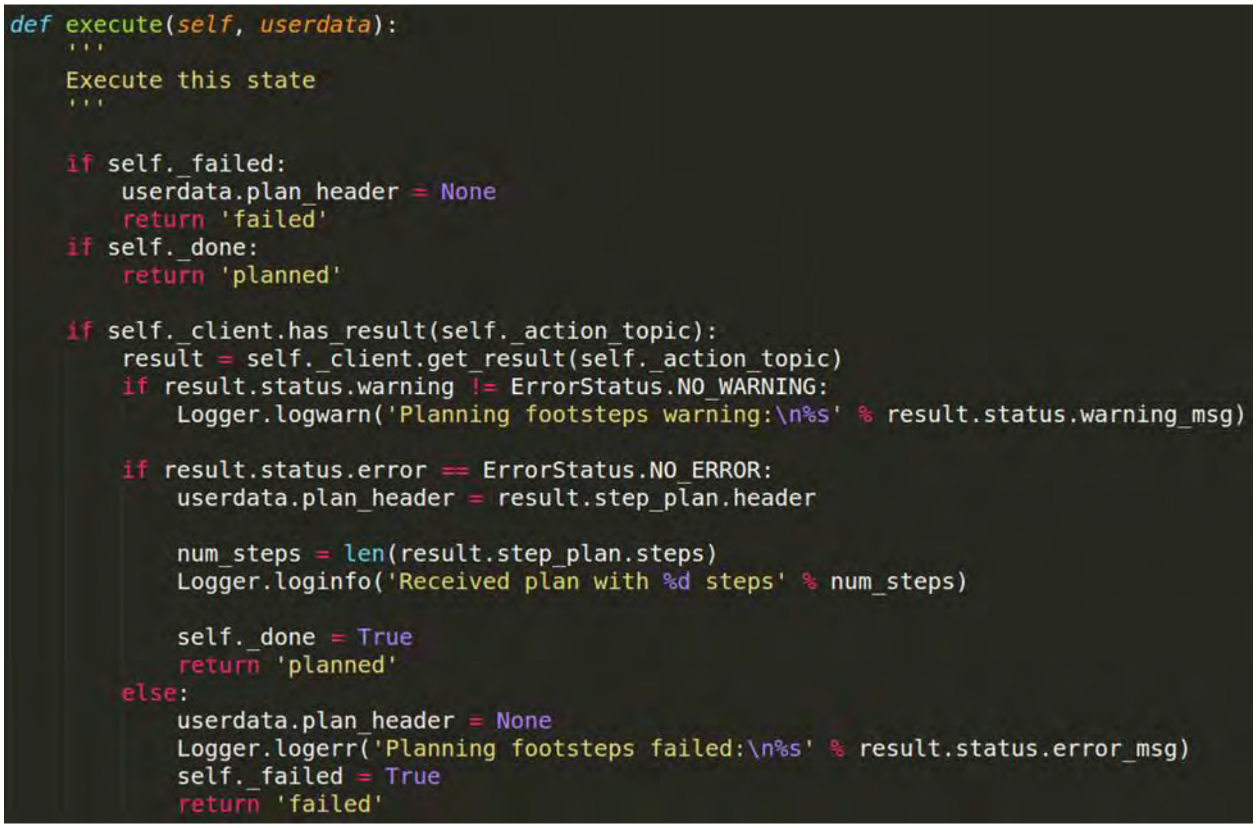


```python
def on_enter(self, userdata):
    '''Upon entering the state, send the footstep plan request.'''

    self._failed = False
    self._done = False

    # Create request msg and action goal
    request = StepPlanRequest()
    request.header = userdata.step_goal.header
    request.max_planning_time = 10.0
    request.parameter_set_name = String(self._mode)

    action_goal = StepPlanRequestGoal(plan_request = request)

    try:
        self._client.send_goal(self._action_topic, action_goal)
    except Exception as e:
        Logger.logwarn('Failed to send footstep plan request:\n%s' % str(e))
        self._failed = True
```

Figure 76. The PlanFootstepsState's on_enter method.

The two aforementioned attributes, done and failed, are reset (since a state can be entered many times during behavior execution). The main purpose of this state's on_enter method is to create and send a footstep plan request. The request is populated with information provided to this state via its input key, "step_goal", as well as the constructor's input argument, "mode".

Figure 77 shows the python code for the states' execute method that is called every update cycle for which the state is active.



```
def execute(self, userdata):
    '''
    Execute this state
    '''

    if self._failed:
        userdata.plan_header = None
        return 'failed'
    if self._done:
        return 'planned'

    if self._client.has_result(self._action_topic):
        result = self._client.get_result(self._action_topic)
        if result.status.warning != ErrorStatus.NO_WARNING:
            Logger.logwarn('Planning footsteps warning:\n%s' % result.status.warning_msg)

        if result.status.error == ErrorStatus.NO_ERROR:
            userdata.plan_header = result.step_plan.header

            num_steps = len(result.step_plan.steps)
            Logger.loginfo('Received plan with %d steps' % num_steps)

            self._done = True
            return 'planned'
        else:
            userdata.plan_header = None
            Logger.logerr('Planning footsteps failed:\n%s' % result.status.error_msg)
            self._failed = True
            return 'failed'
```

Figure 77. The PlanFootstepsState's execute method.

The PlanFoostepsState's execute method is executed until the onboard footstep manager has responded with a result. If planning was successful, it writes the result to its output key, "plan_header", and returns the outcome "planned". If planning was unsuccessful, it notifies the operator and returns the outcome "failed".

H.2. List of States

All states that were included in a behavior that was used during the DRC Finals or the experimental demonstrations in the lab are enumerated below, in groups of related functionality:

- **Footstep Planning and Execution states**
  - CreateStepGoalState
  - PlanFootstepsState
  - FootstepPlanRelativeState
  - ExecuteStepPlanActionState
- **Object Templates -related states**
  - GetTemplateAffordanceState
  - GetTemplateFingerConfigState
  - GetTemplateGraspState
  - GetTemplatePoseState
  - GetTemplatePregraspState
  - GetTemplateStandPoseState
  - GetTemplateUsabilityState
  - AttachObjectState
  - DetachObjectState
- **Motion Planning and Execution states**

    - PlanAffordanceState
    - PlanEndeffectorCartesianWaypointsState
    - PlanEndeffectorPoseState
    - ExecuteTrajectoryMsgState
    - MoveitPredefinedPoseState
    - FingerConfigurationState
    - HandTrajectoryState
    - TiltHeadState
- **ATLAS-specific states**
    - ChangeControlModeActionState
    - CheckControlModeActionState
    - RobotStateCommandState
- **Various helper states**
    - GetPoseInFrameState
    - GetWristPoseState
    - CurrentJointPositionsState
    - UpdateJointCalibrationState
- **Generic States**
    - CalculationState
    - FlexibleCalculationState
    - CheckConditionState
    - DecisionState
    - OperatorDecisionState
    - InputState
    - LogState
    - WaitState

H.3. List of Behaviors

We briefly present all behaviors that were used in the DRC Finals or the experimental demonstrations in the lab. In many behaviors, groups of states are placed together in a state machine (gray blocks) in a hierarchical fashion. We do not show the contents of those state machines here, in the interest of space.

Calibration and Startup Behaviors

These behaviors were used during the initial robot start for checkout and to calibrate joint position sensors.

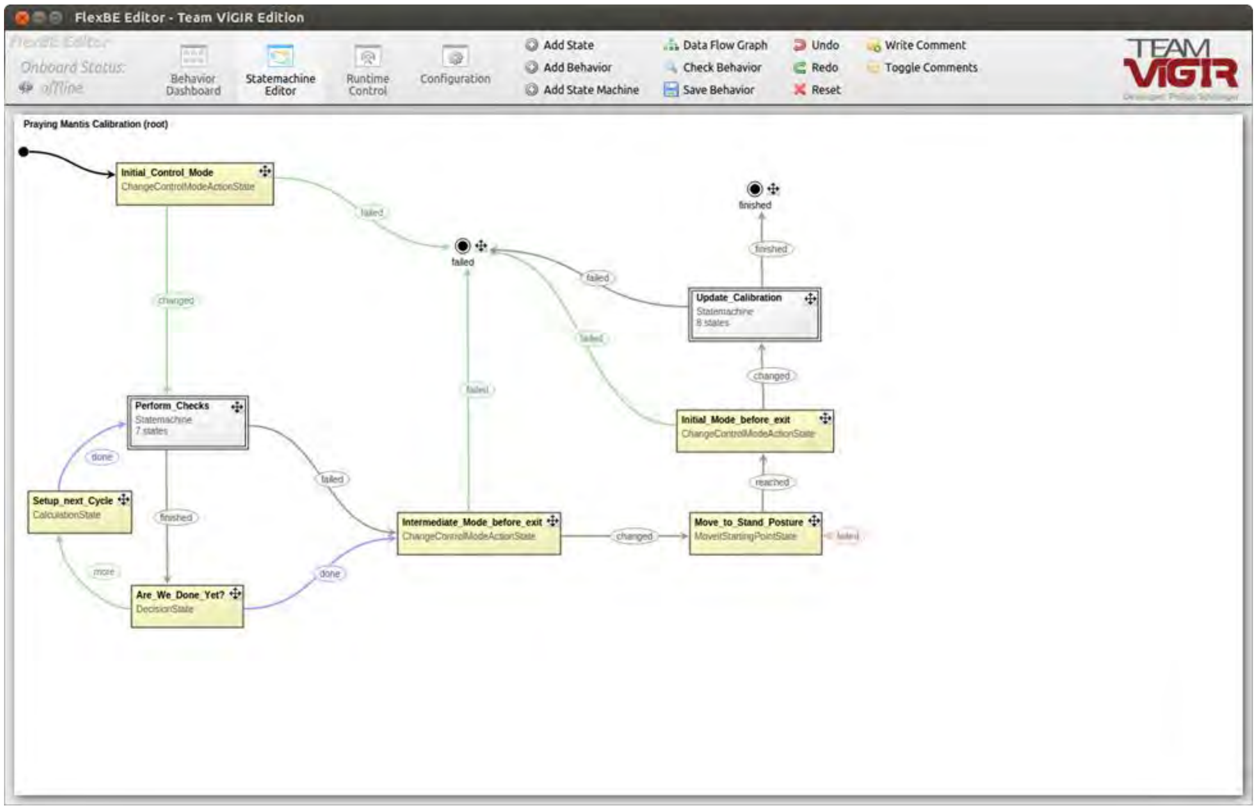


Figure 79. "Praying Mantis Calibration" Behavior.

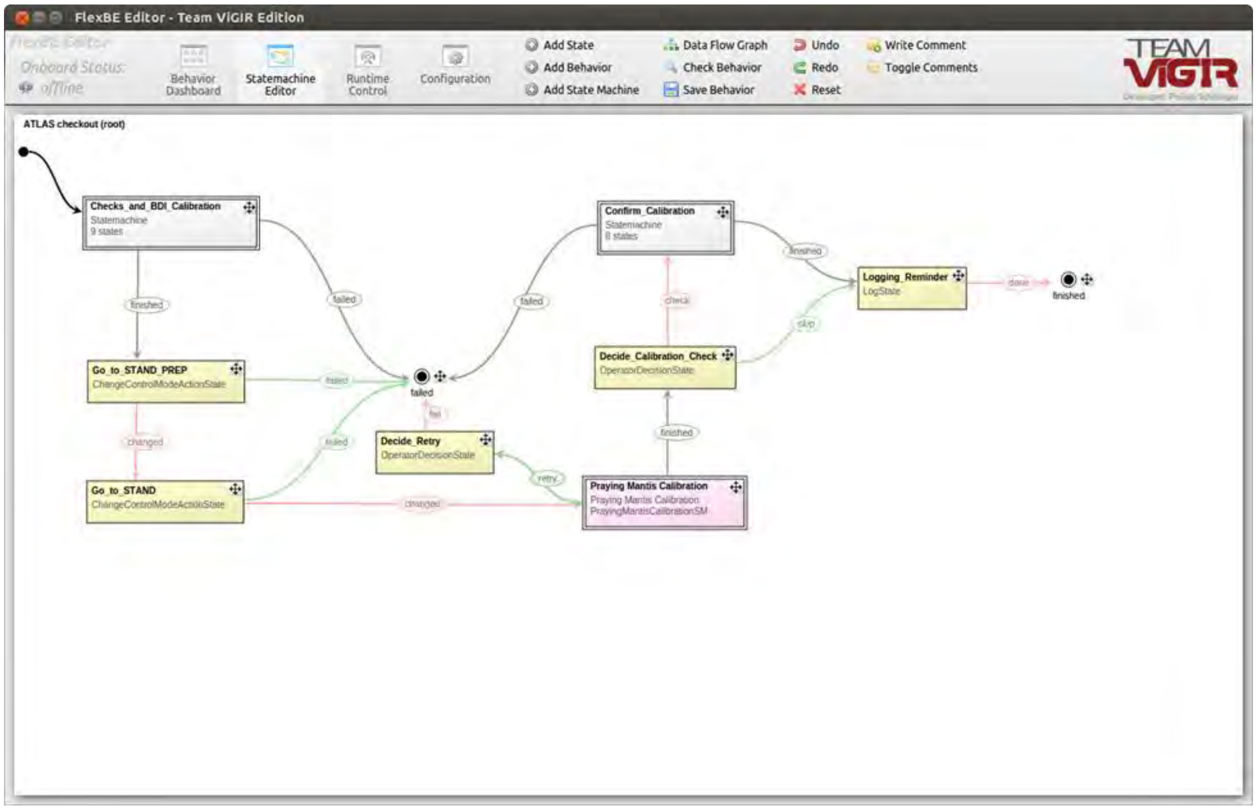


Figure 78. "Atlas Checkout" Behavior.

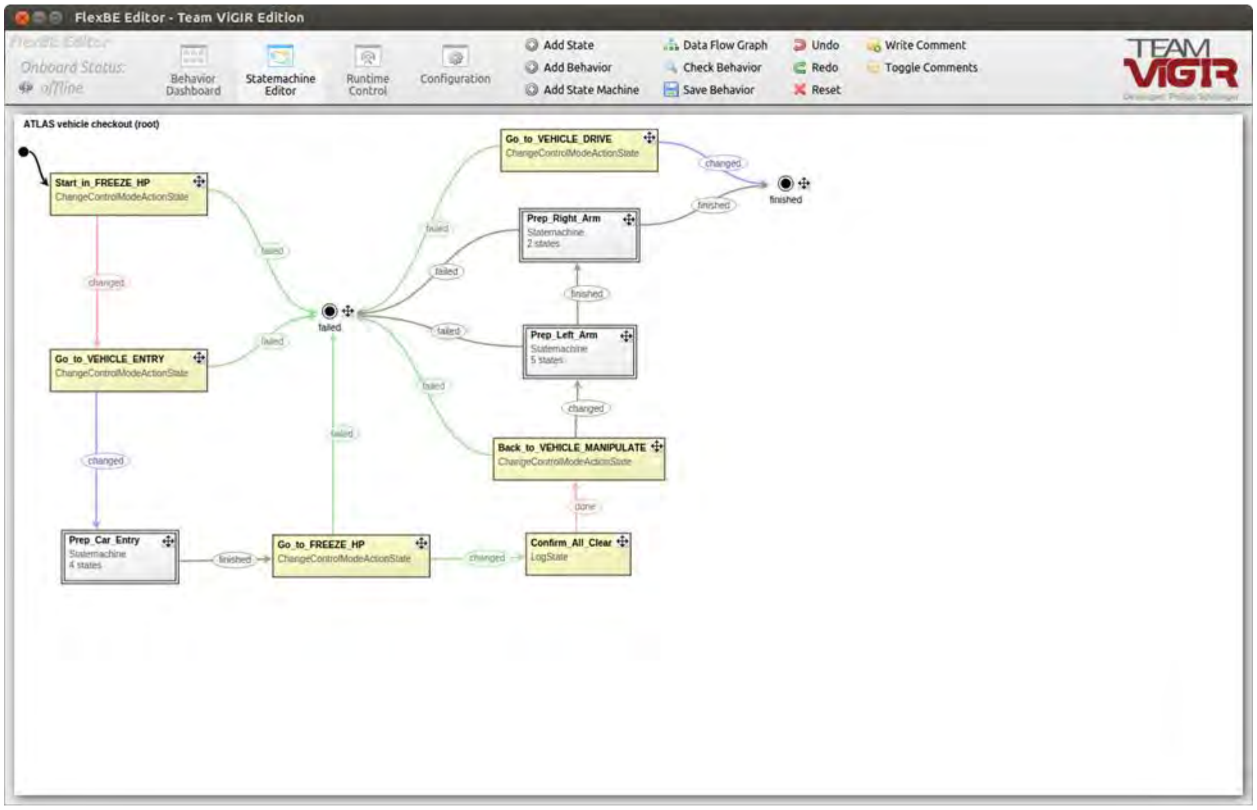


Figure 80. “Atlas Vehicle Checkout” Behavior (used before Driving Task)

Helper Behaviors

Helper behaviors are behaviors that are developed to be embedded into larger task-level behaviors. In other words, these are lower-level states within the hierarchical state machine.

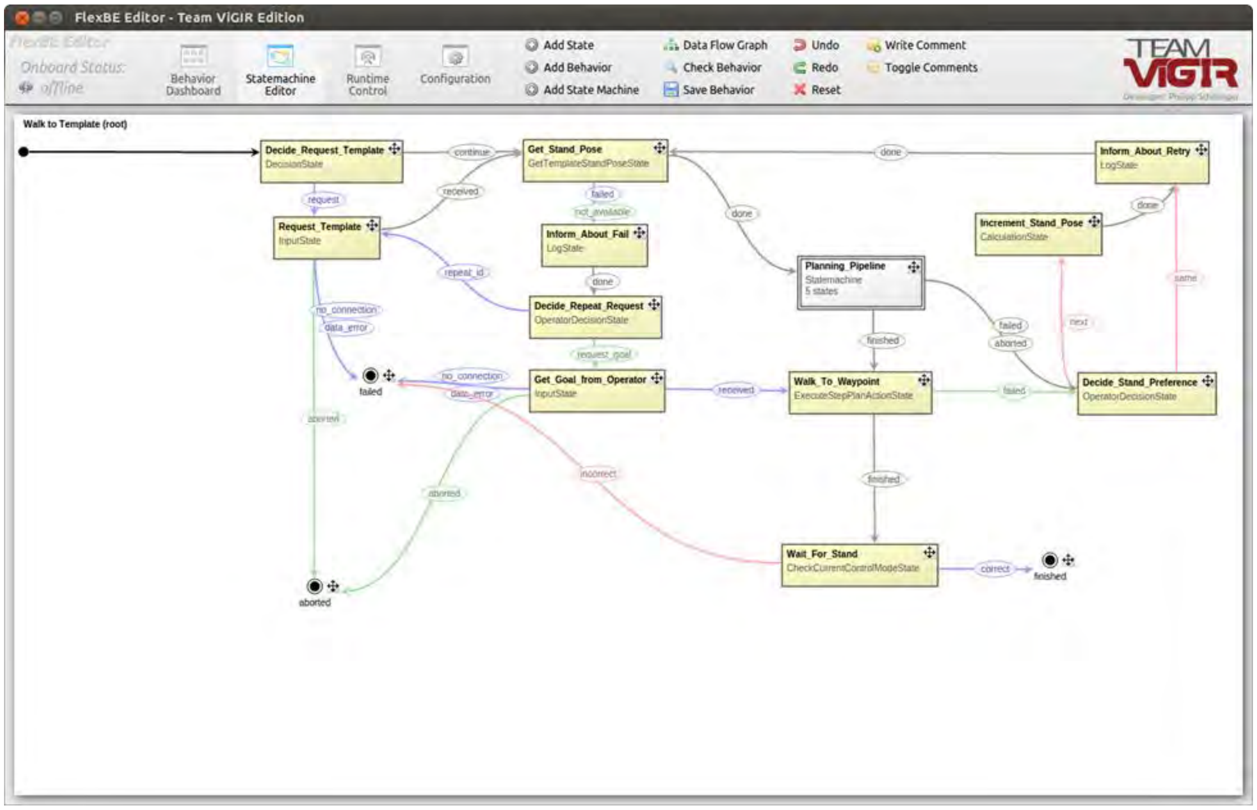
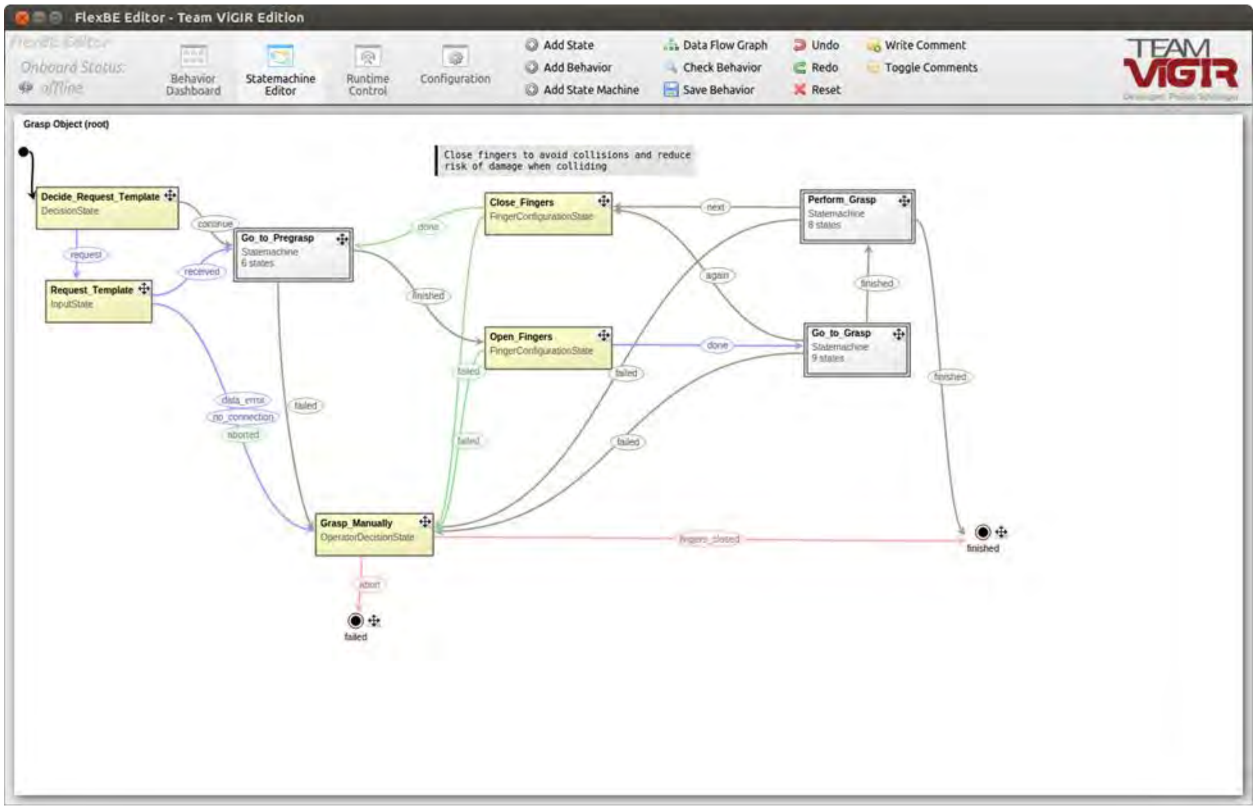


Figure 81. “Walk to Template” Helper Behavior



Close fingers to avoid collisions and reduce risk of damage when colliding

Figure 82. “Grasp Object” Helper Behavior

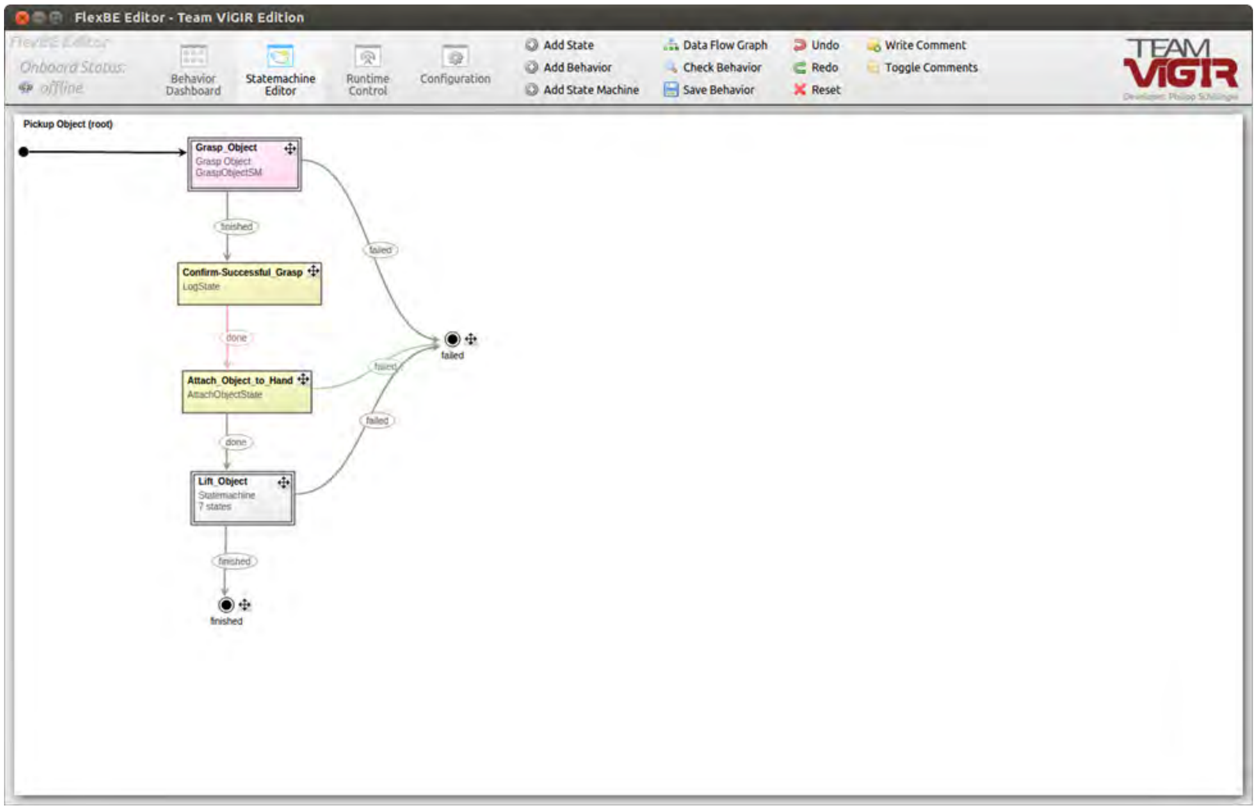


Figure 83. "Pickup Object" Helper Behavior

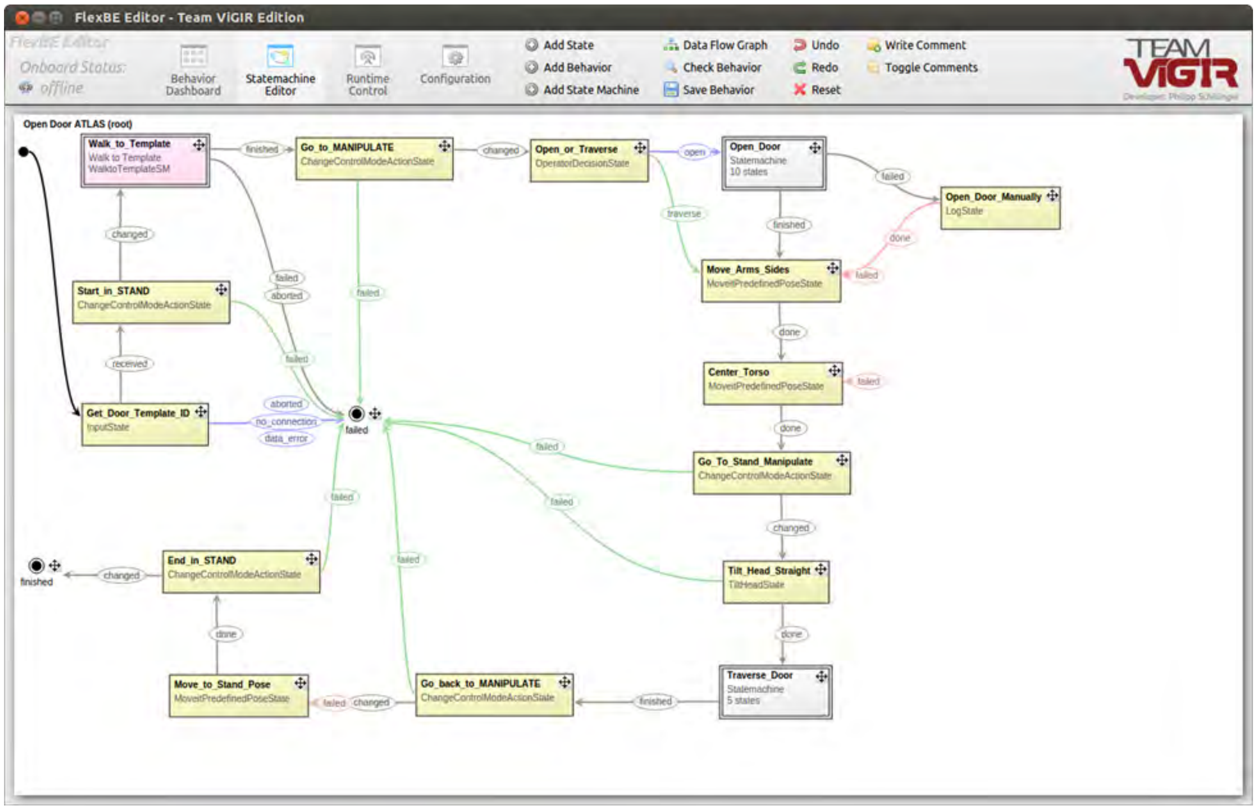


Figure 84. "Open Door" Helper Behavior (DRC Task #3)

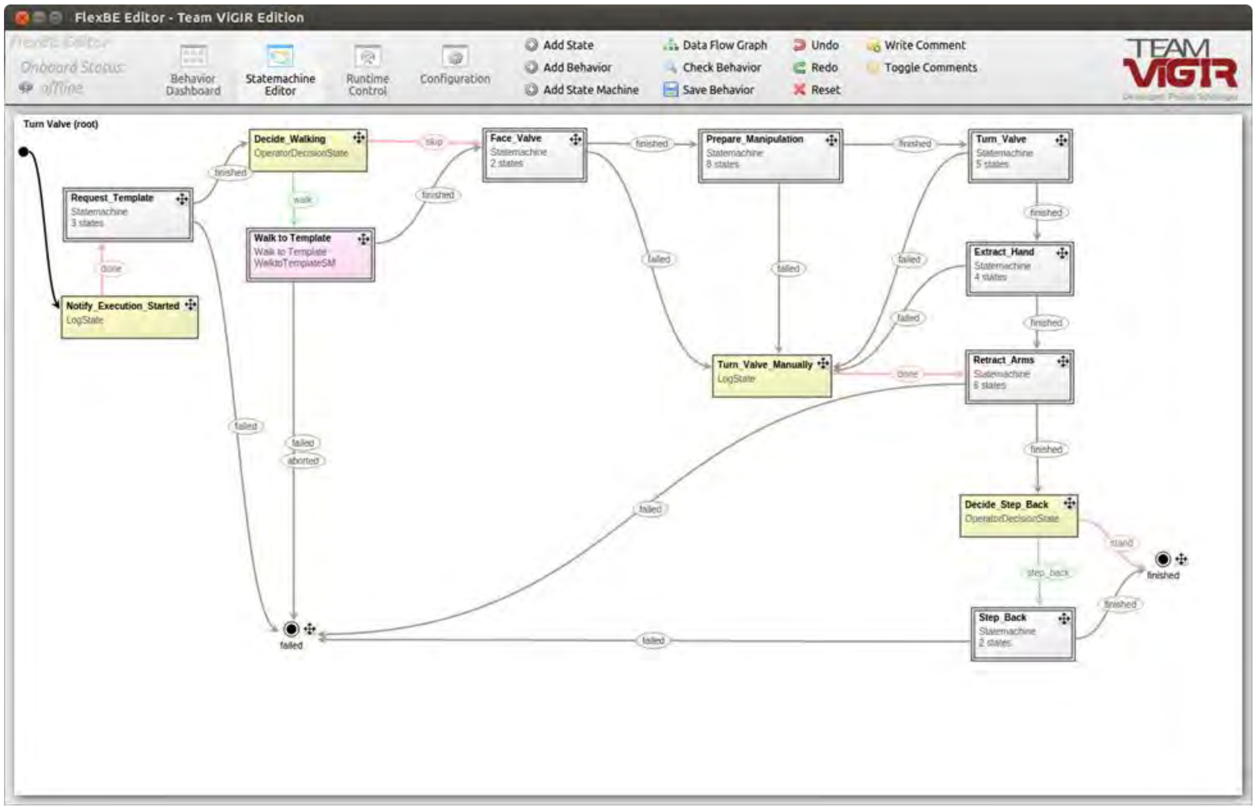


Figure 85. “Turn Valve” Helper Behavior (DRC Task #4)

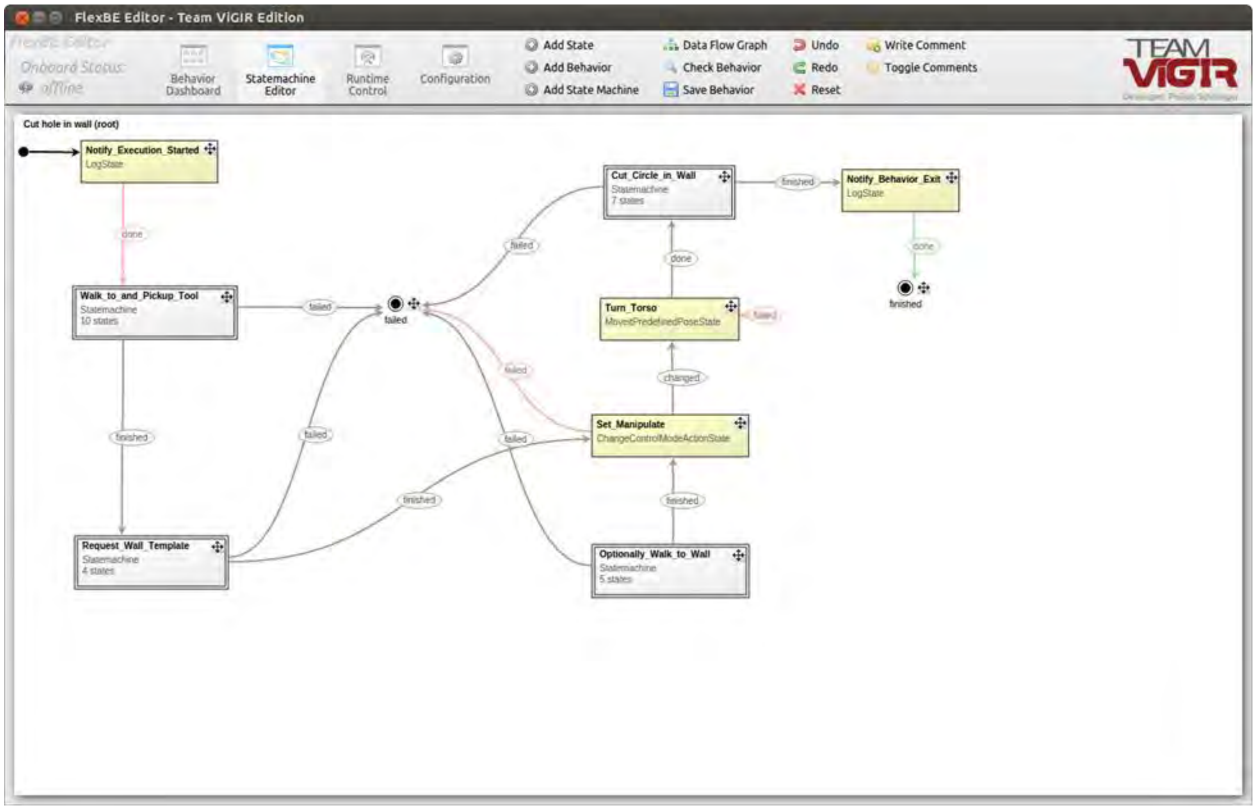


Figure 86. “Cut Hole in Wall” Helper Behavior (DRC Task #6)

H.4. Experimental Demonstration of Behaviors

The lab setup for the task-specific behaviors demonstrations was as follows. ATLAS was positioned in front of the object of interest (e.g. door, valve, wall), since a hardware issue with our ATLAS' left hip prevented any demo that involved walking or stepping. Calibration of the electric and hydraulic joints was performed in advance (using the "ATLAS Checkout" behavior above). Moreover, two operators were performing the behavior execution and perception tasks on a single OCS computer.

H.4.1. Demo #1: "Open Door" (by pushing the handle from below)

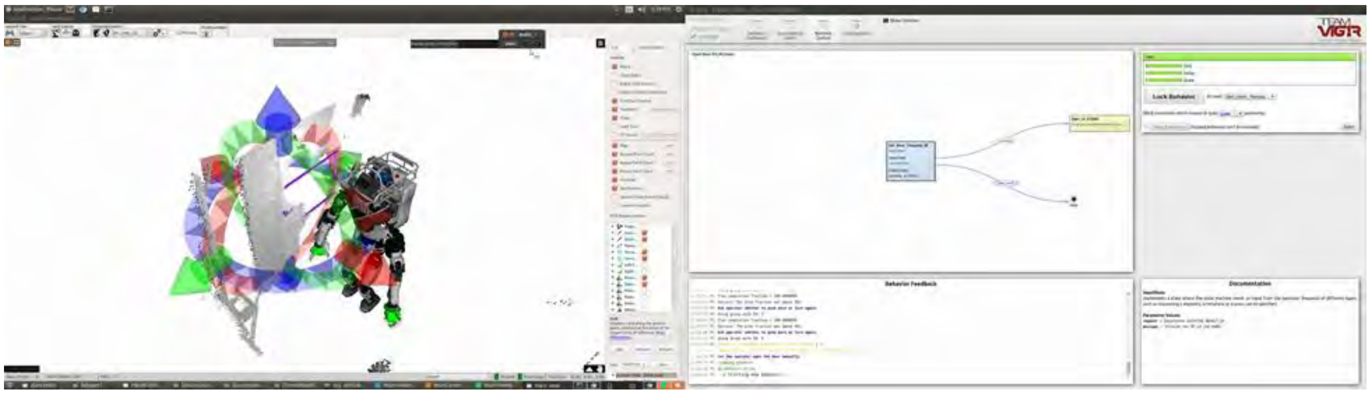


Figure 87. Requesting Door Object Template from Operator

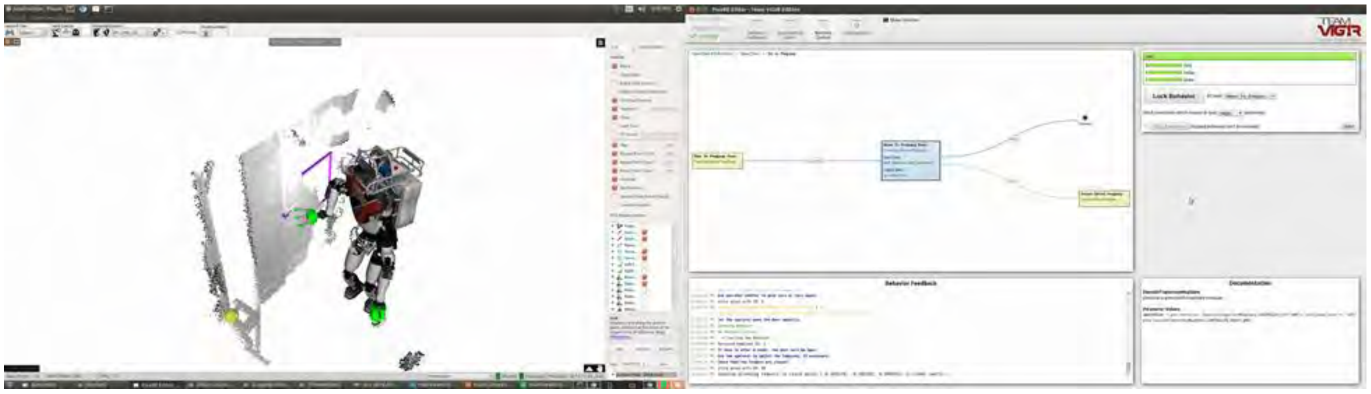


Figure 88. Behavior positions Atlas relative to template

With the template identifier available, the behavior can position ATLAS and guide its right arm to the template's "pre-grasp pose", which it obtained by querying the template server.

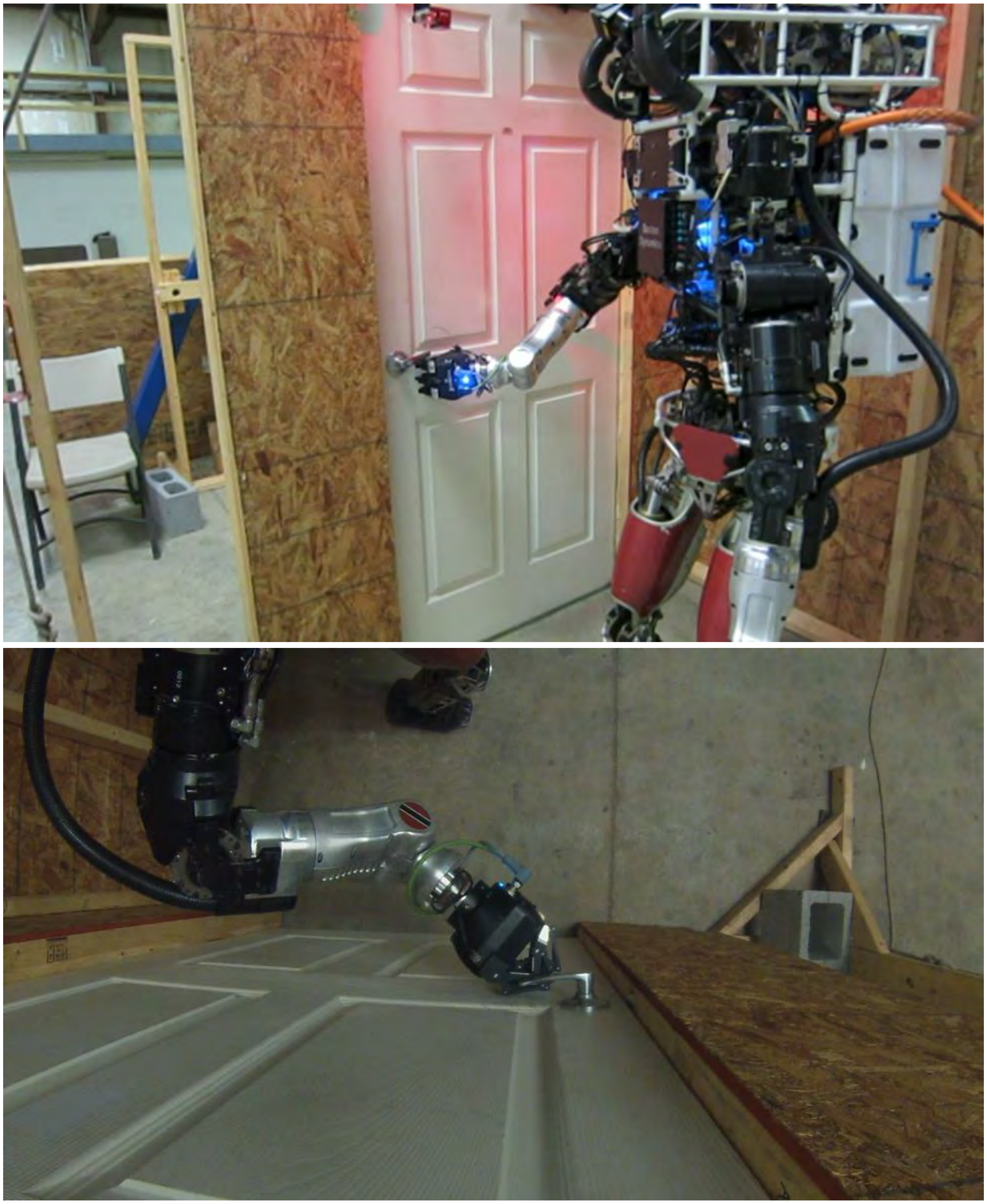


Figure 89. Atlas pushing the door handle from below

In this demo, we employed the tactic of pushing the door handle from below, with the fingers closed in a "fist". This tactic was more robust to inaccuracies in end effector position.

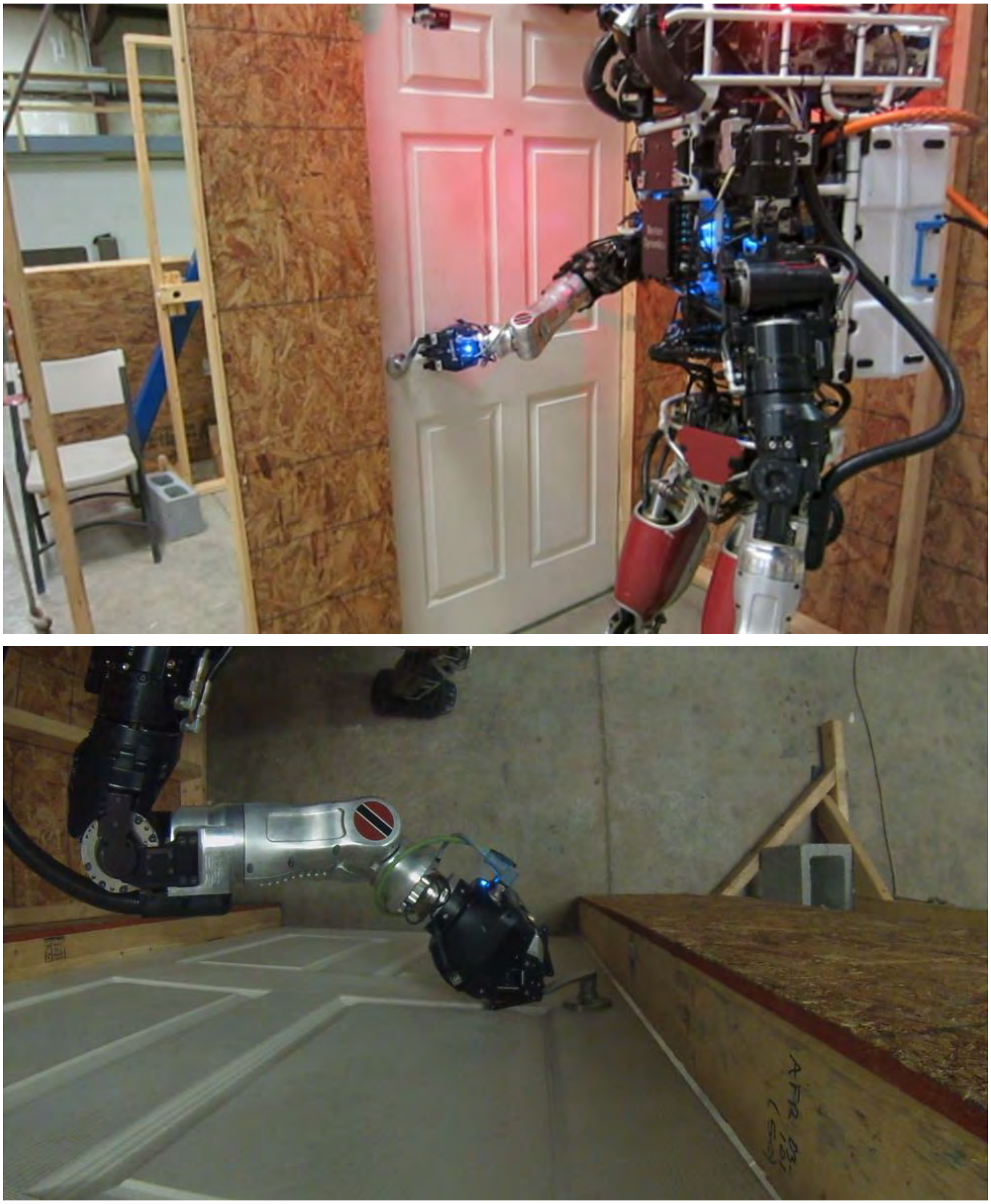


Figure 90. Atlas unlatching the door using “turnCCW” affordance

With the end effector in position, the behavior executes the “turnCCW” affordance of the door template, which results in a counterclockwise circular arc. It then executes the “push” affordance, which results in motion perpendicular to the door. As a result, the door is unlatched.

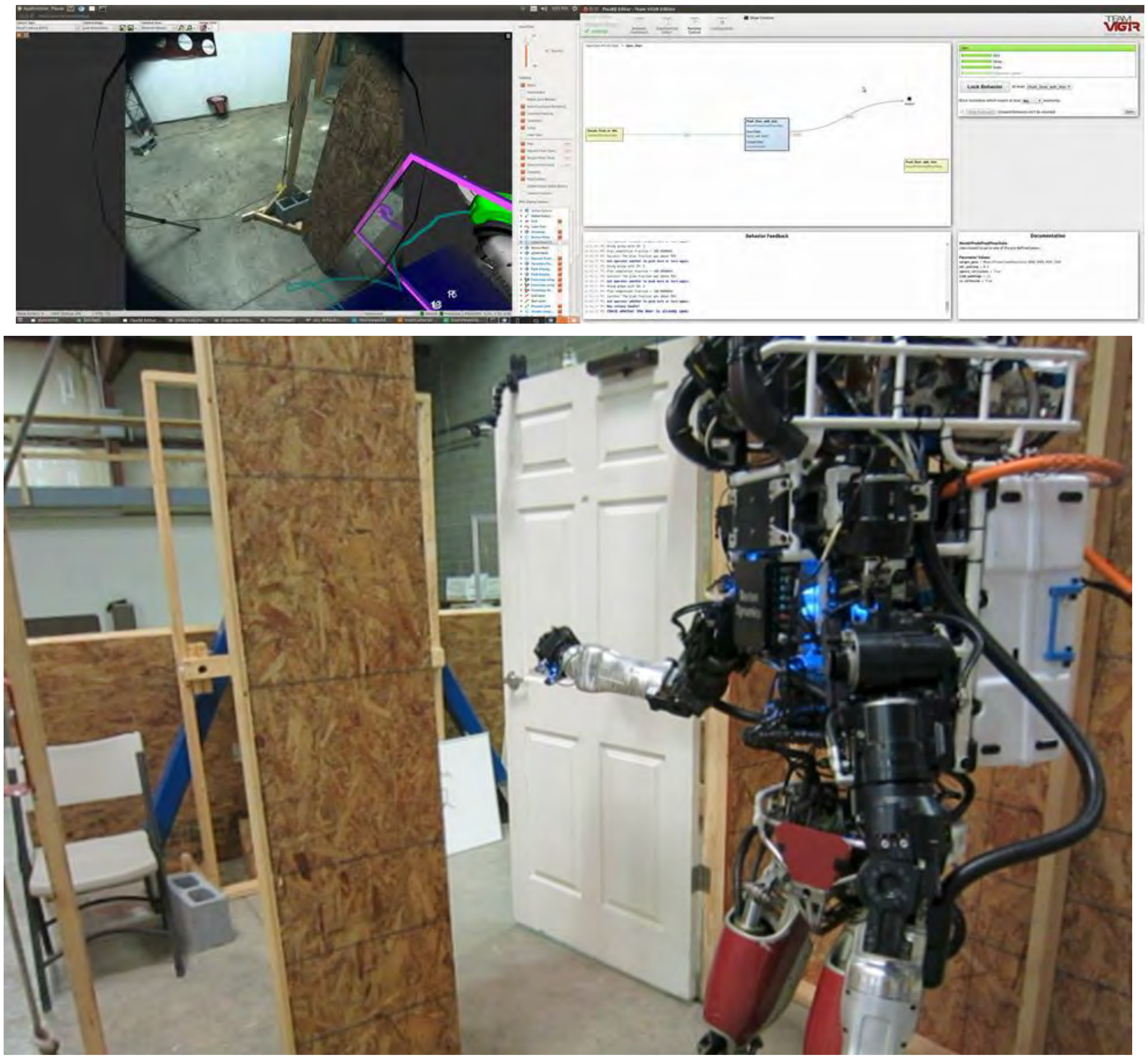


Figure 91. With the door unlatched, the behavior pushes the door completely open.

The next steps of this behavior would have been to bring the arms to the sides, center the torso, and then request a footstep plan in order to “strafe” (step sideways) through the doorway.

H.4.2. Demo #2: “Open Door” (by grasping and turning the handle)

This demo differs from Demo #1 only in the tactic employed for unlatching the door.

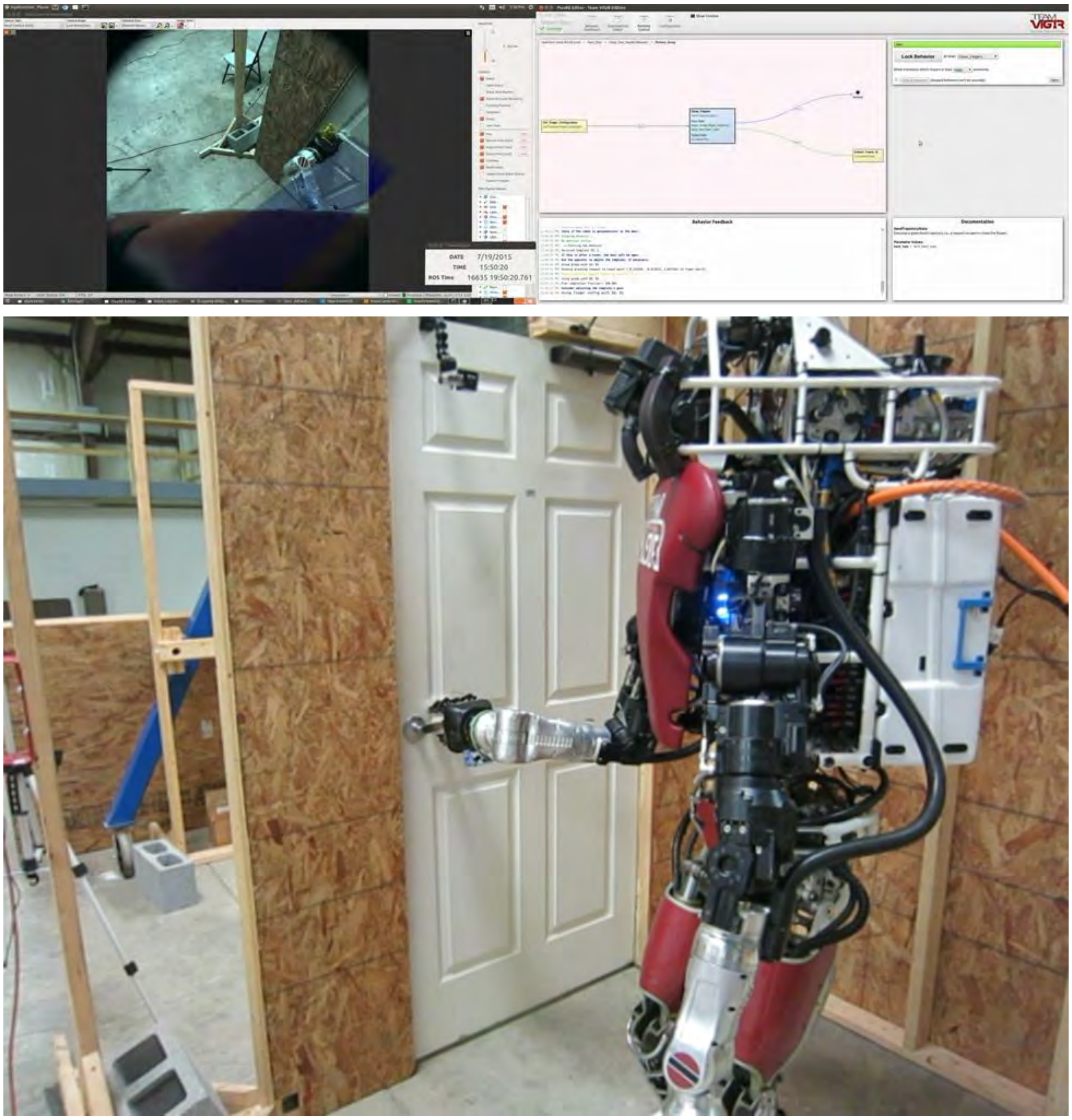


Figure 92. Different behavior used to grasp the door handle with fingers

In this demo, the behavior requests different “pre-grasp” and “grasp” poses. In between, it opens the fingers (top). The result is the fingers around the door handle (bottom).

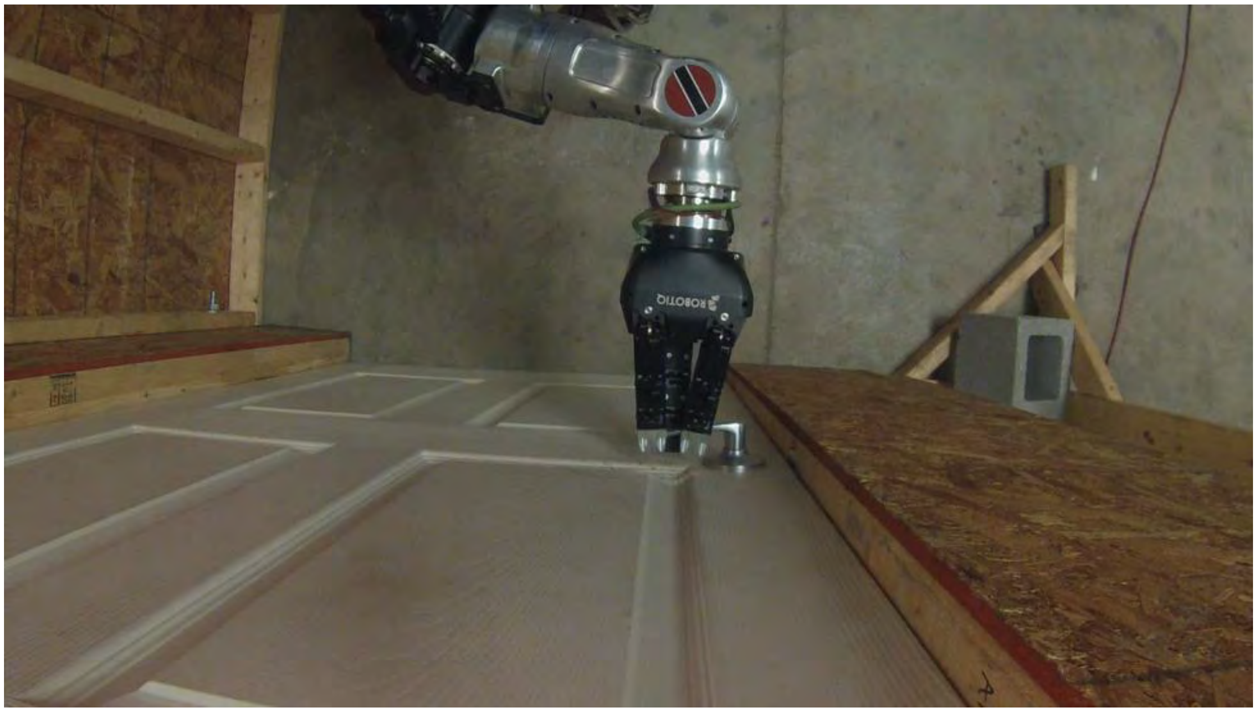


Figure 93. The behavior closes the fingers around the door handle.
But not in a "fist"-like manner like in Demo #1. Rather, it requests a specific grasp posture from the template server.

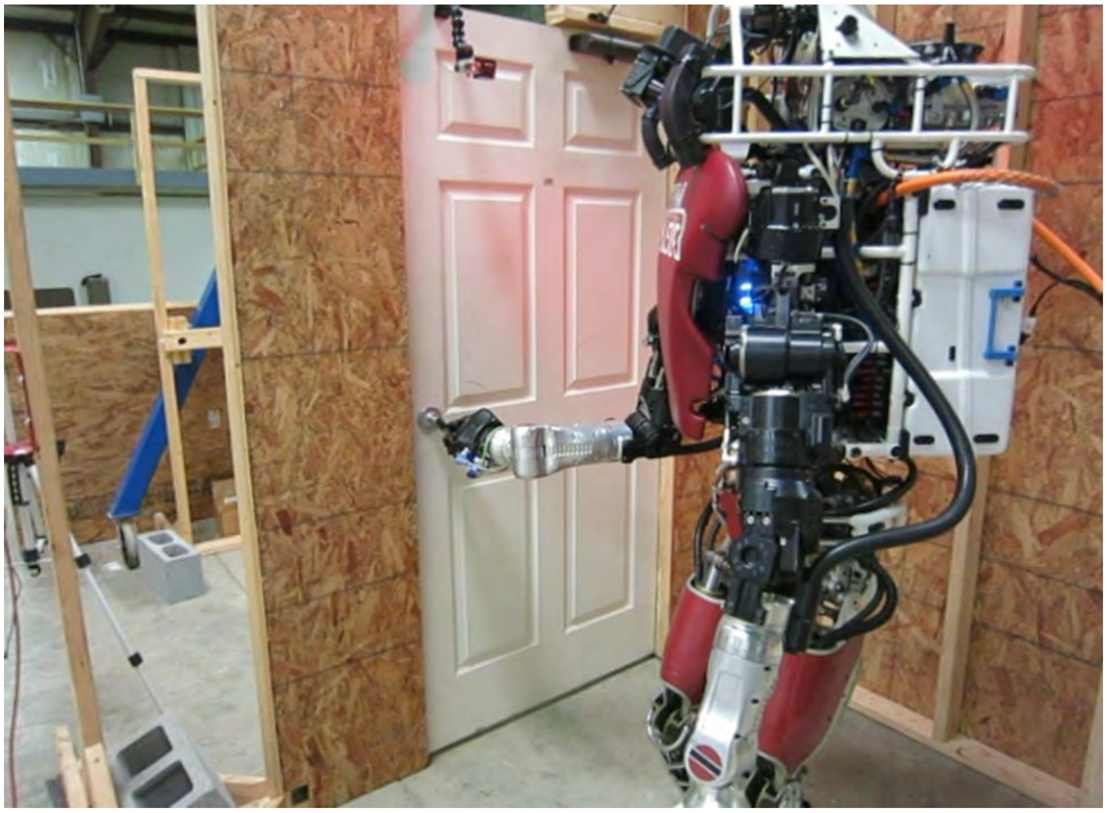


Figure 94. The behavior executes the "turn CW" affordance to unlatch the door.
With a firm grasp of the door handle, the behavior turns the handle in a clockwise circular arc. It then pushes the door as in Demo #1.

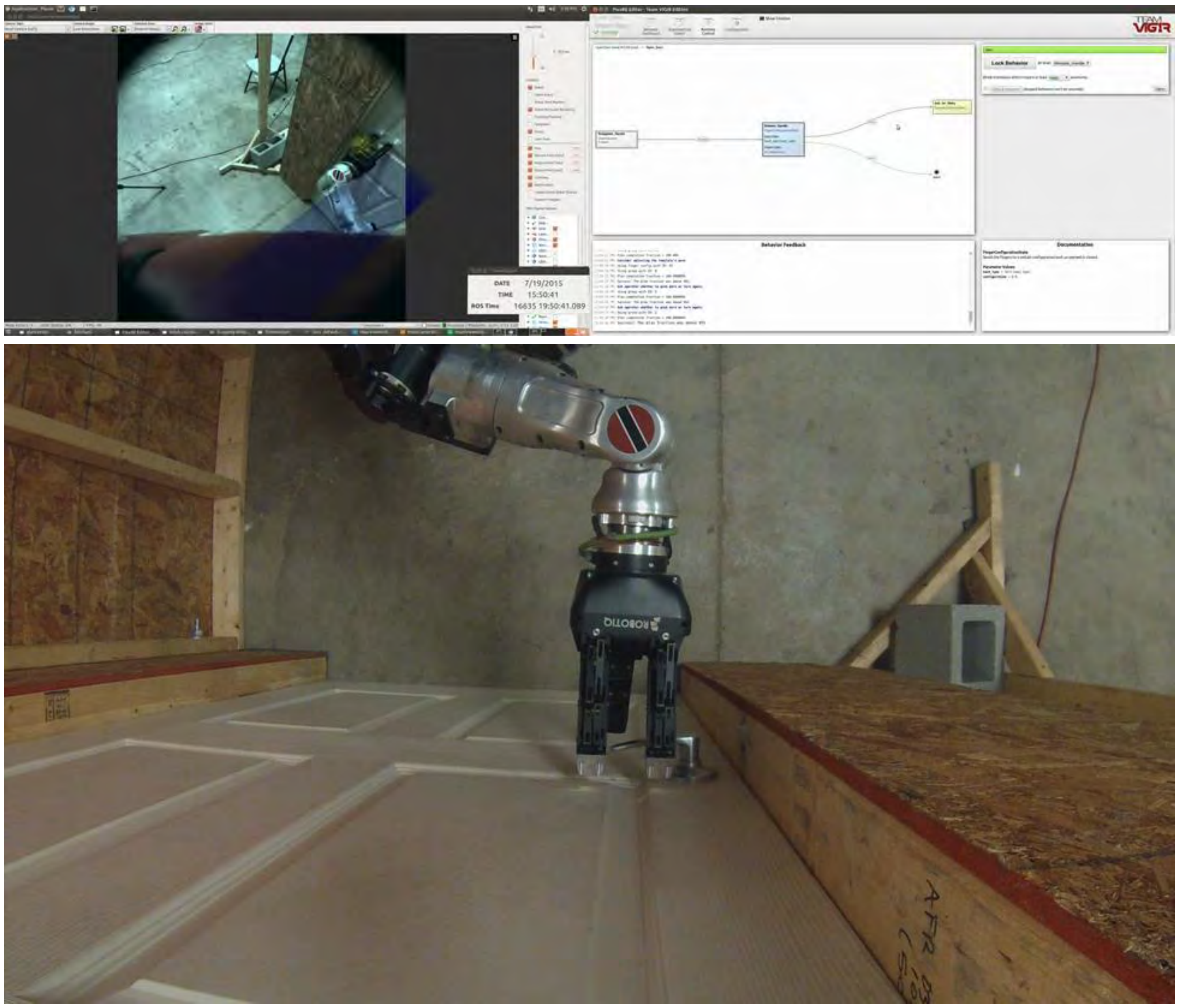


Figure 95. Atlas releases the door handle after unlatching.
This tactic requires that ATLAS releases its grasp on the door handle in between unlatching the door and pushing it wide open with its arm.Once the behavior starts, it requests a door object template from the operator (right). The operator places and aligns the door template, then sends its identifier to the behavior (left).

H.4.3. Demo #3: "Turn Valve"

This behavior is employing the strategy of turning the valve by inserting a "poke stick" attached to ATLAS' left wrist (see Figure 98). We used this strategy during Day 1 of the DRC Finals.

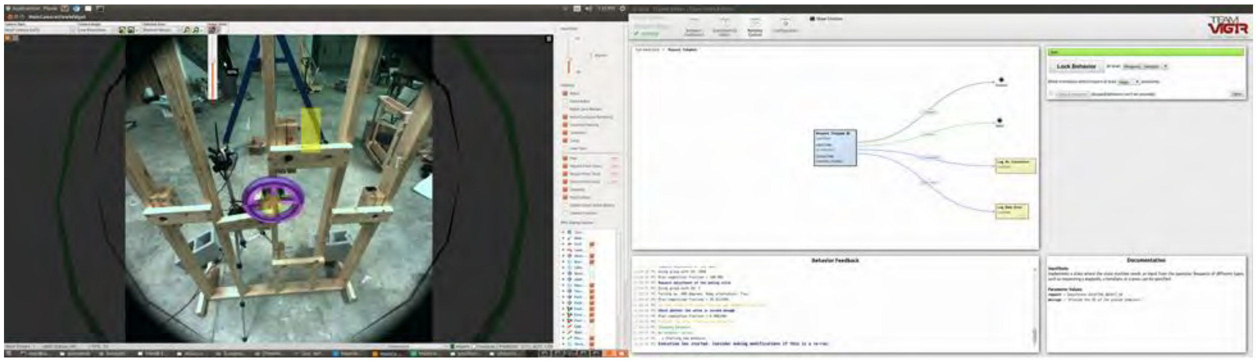


Figure 96. First, request an object template (purple valve) from the operator

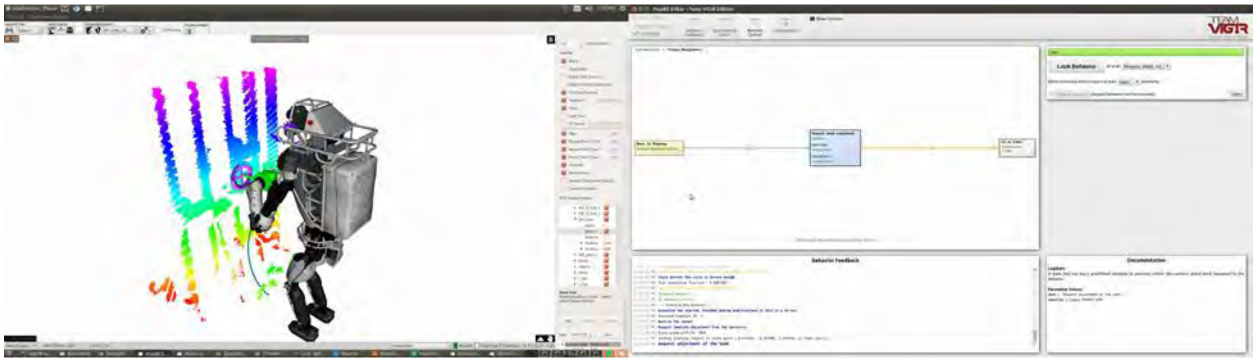


Figure 97. Operator verifies relative position of “poke stick” and valve.

Once the end effector, the “poke stick” in this case, is in front of the valve, the behavior asks the operator to check whether it is clear for insertion. If not, the operator has a chance to manually adjust the end-effector’s position and then let the behavior proceed (the transition is “blocked”).

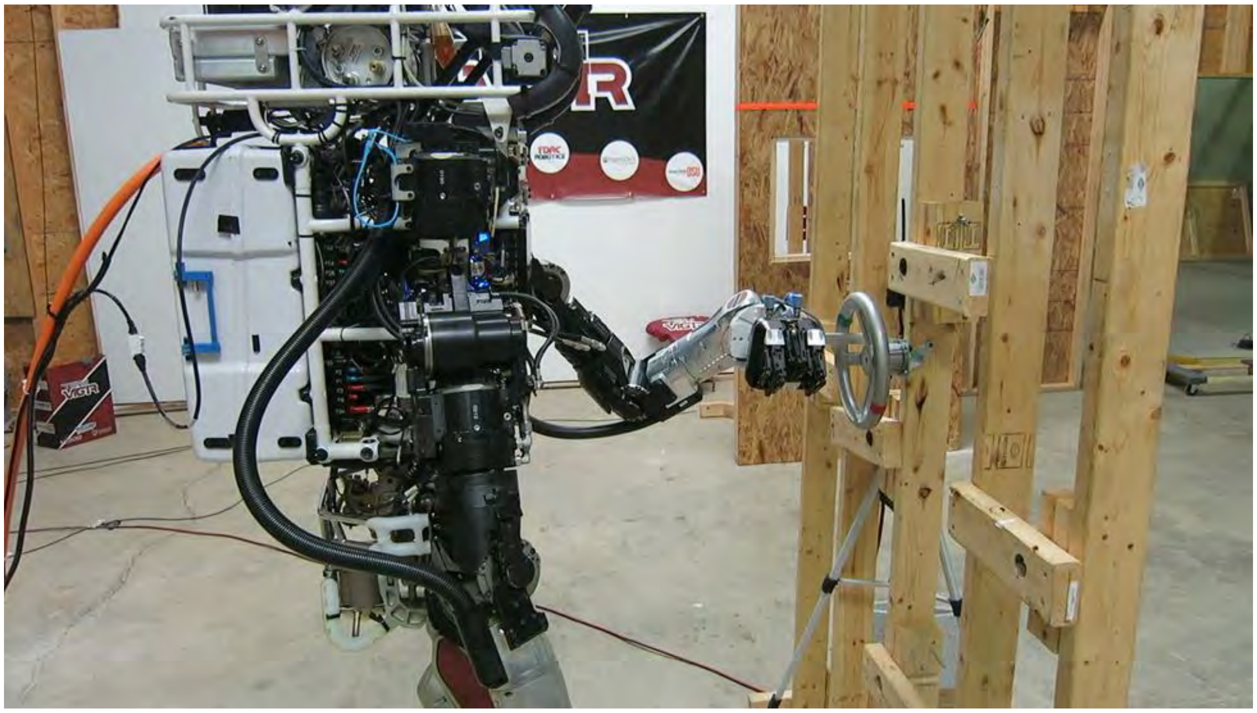


Figure 98. The behavior then executes the “insert” affordance of the valve template.

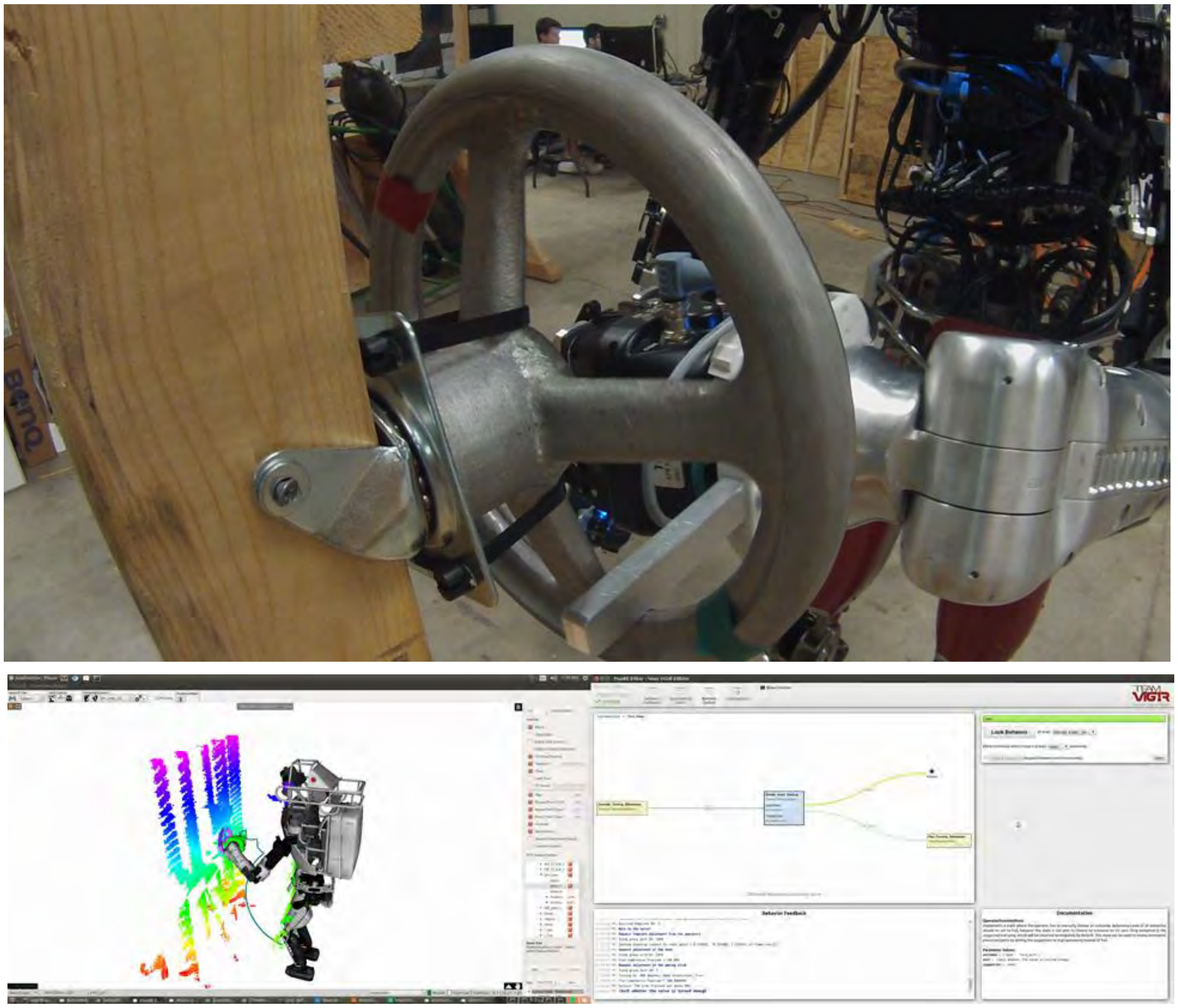


Figure 99. the behavior executes the “open” valve affordance

With the end effector inserted, the behavior executes the “open” valve affordance, which results in counter-clockwise rotation around the valve’s axis (top). If the desired amount of rotation is not achieved by one execution of the affordance, the behavior gives the option of repeating the turning step (bottom right). Due to the end effector (“poke stick”) configuration, valve turning can be repeated ad infinitum. The kinematics do not impose any limits on rotation.

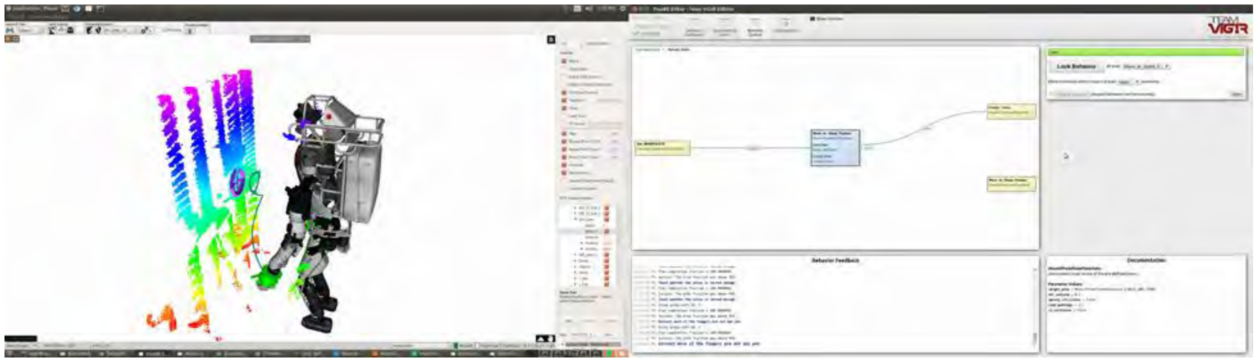


Figure 100. Once the valve is open, the behavior returns the arm to ATLAS' side.

H.4.4. Demo #4: "Cut Hole in Wall" (emulated by drawing circle with marker)

We chose to emulate the "cut hole in wall" task by drawing a circle on a whiteboard by attaching a dry erase marker at the tip of the cutting tool; however, the behavior did not have to be modified in any way to account for this new task setup. This task is similar to the one presented in Appendix E but using the advantages of the behavior engine.

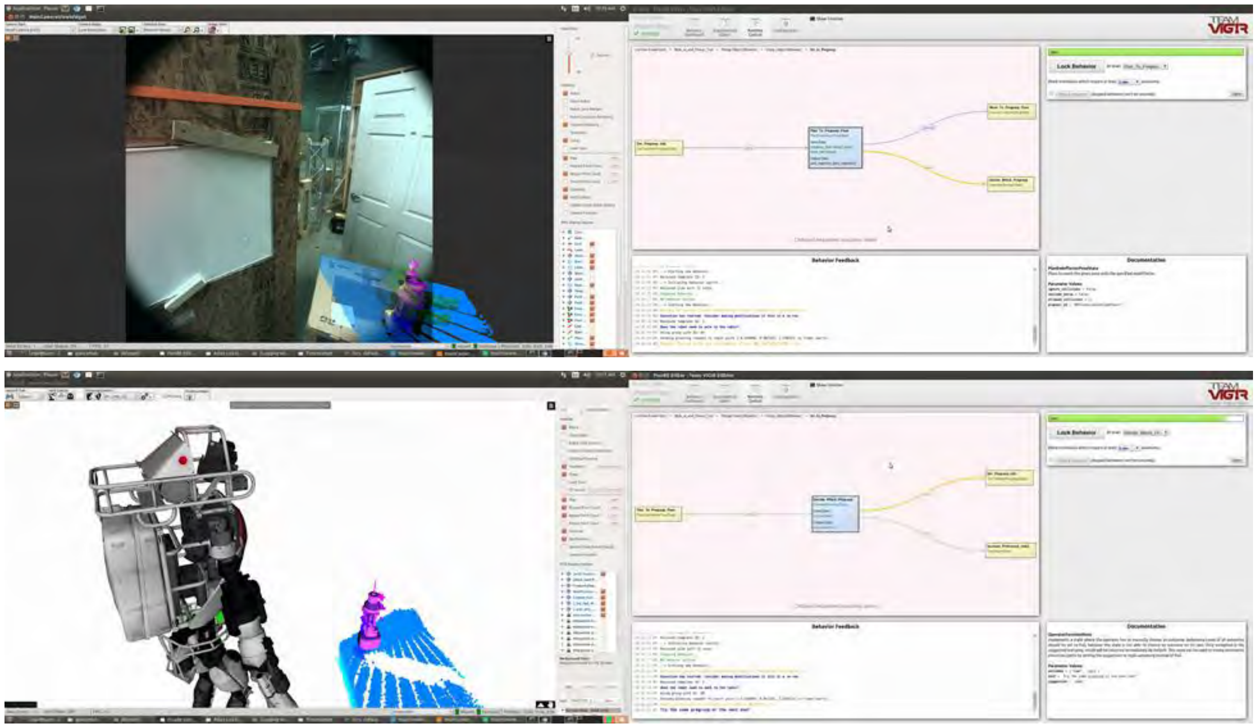


Figure 101. Executing the behavior and failure recovery.

When the behavior tries to move the right hand to the template's "pre-grasp" pose, planning fails (top right). The template had been misplaced, so the behavior allows the operator to properly align the template (bottom left) and then repeat the planning step with the same (or different) "pre-grasp" pose (bottom right).

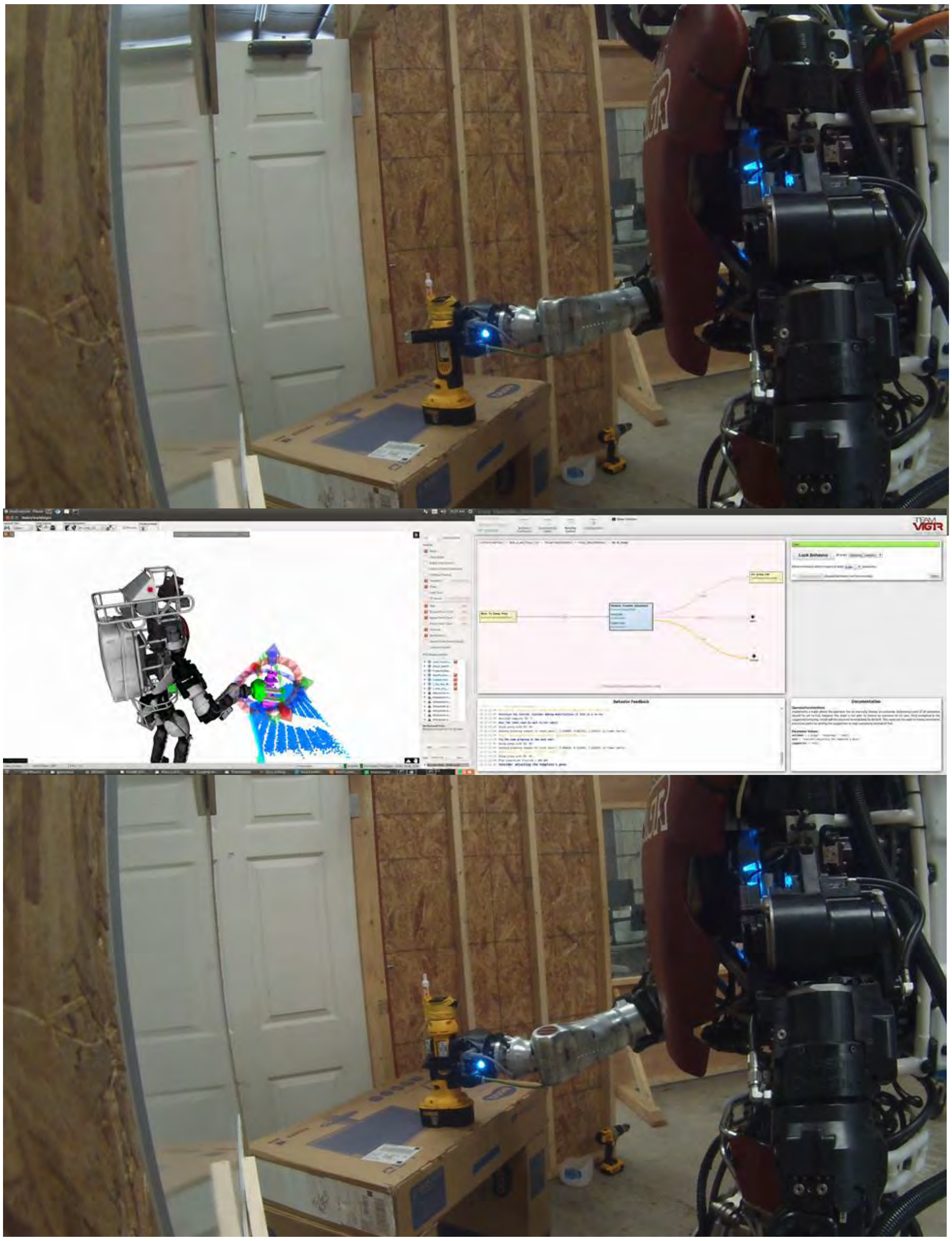


Figure 102. Atlas grasping tool after operator intervention.

Once ATLAS' hand moves to the "grasp" pose, the operator notices that the template's height is incorrect (top). Again, the behavior allows the operator to make adjustments to the template's position (middle left) and then repeat the previous step (middle right). As a result, the hand has a proper grasp around the cutting tool (bottom).

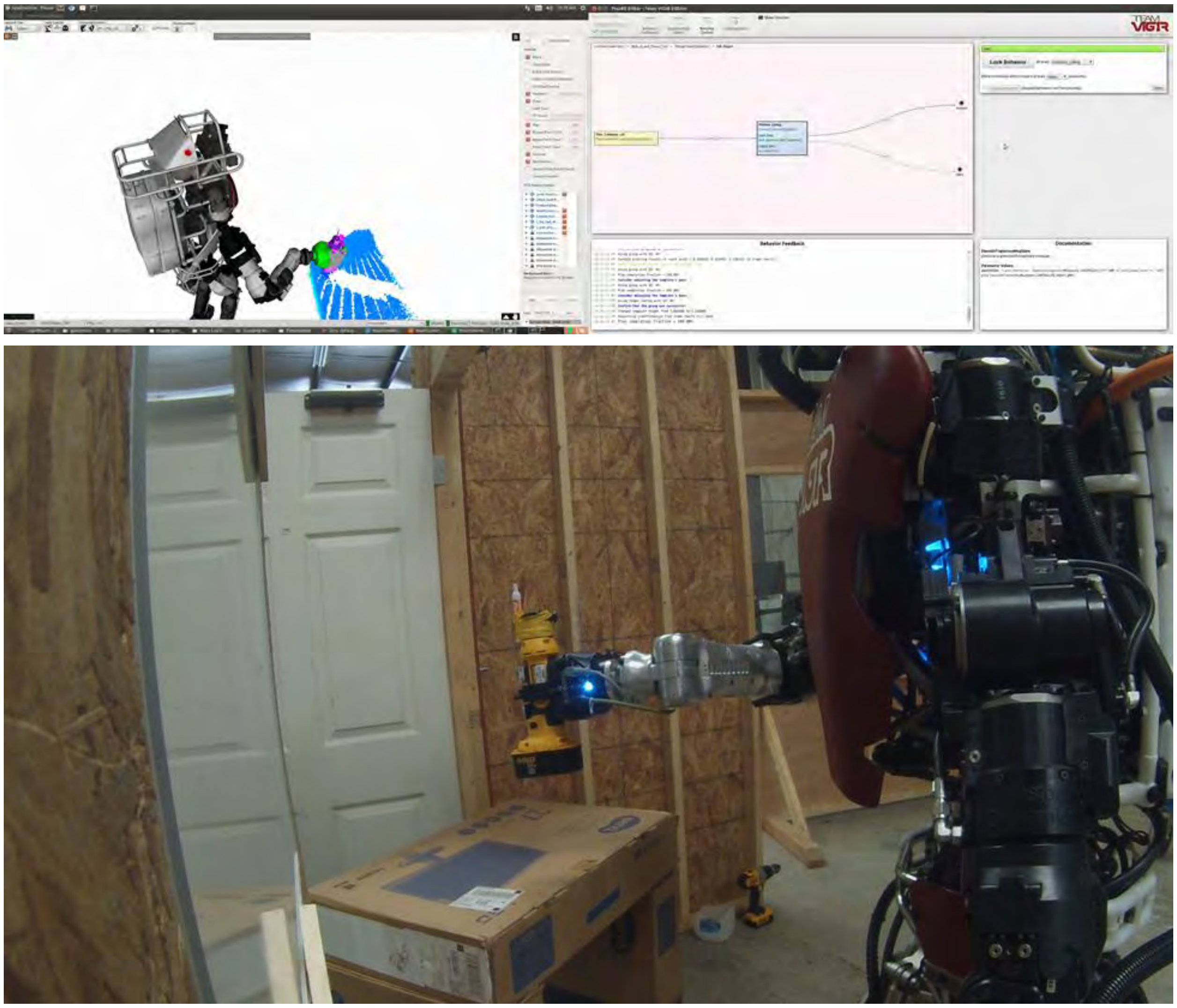


Figure 103. After grasping, the behavior "attaches" the object to the robot model in MoveIt!.

After grasping, the behavior "attaches" the object to the robot model in the MoveIt! Planning scene. It can then request a motion plan for lifting the object that accounts for the object in terms of collisions.

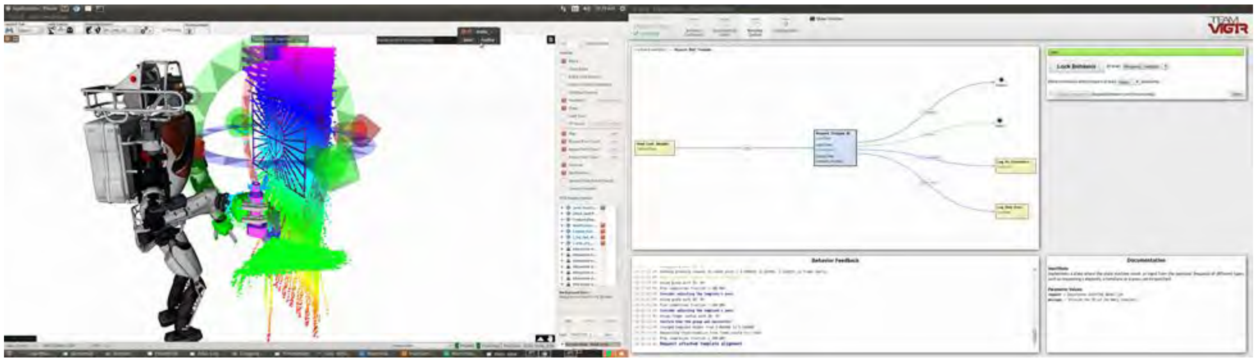


Figure 104. Inputting the wall cutting template.

This task involves two objects (cutting tool and wall with circular pattern), therefore the behavior is now asking the operator to provide the wall template.

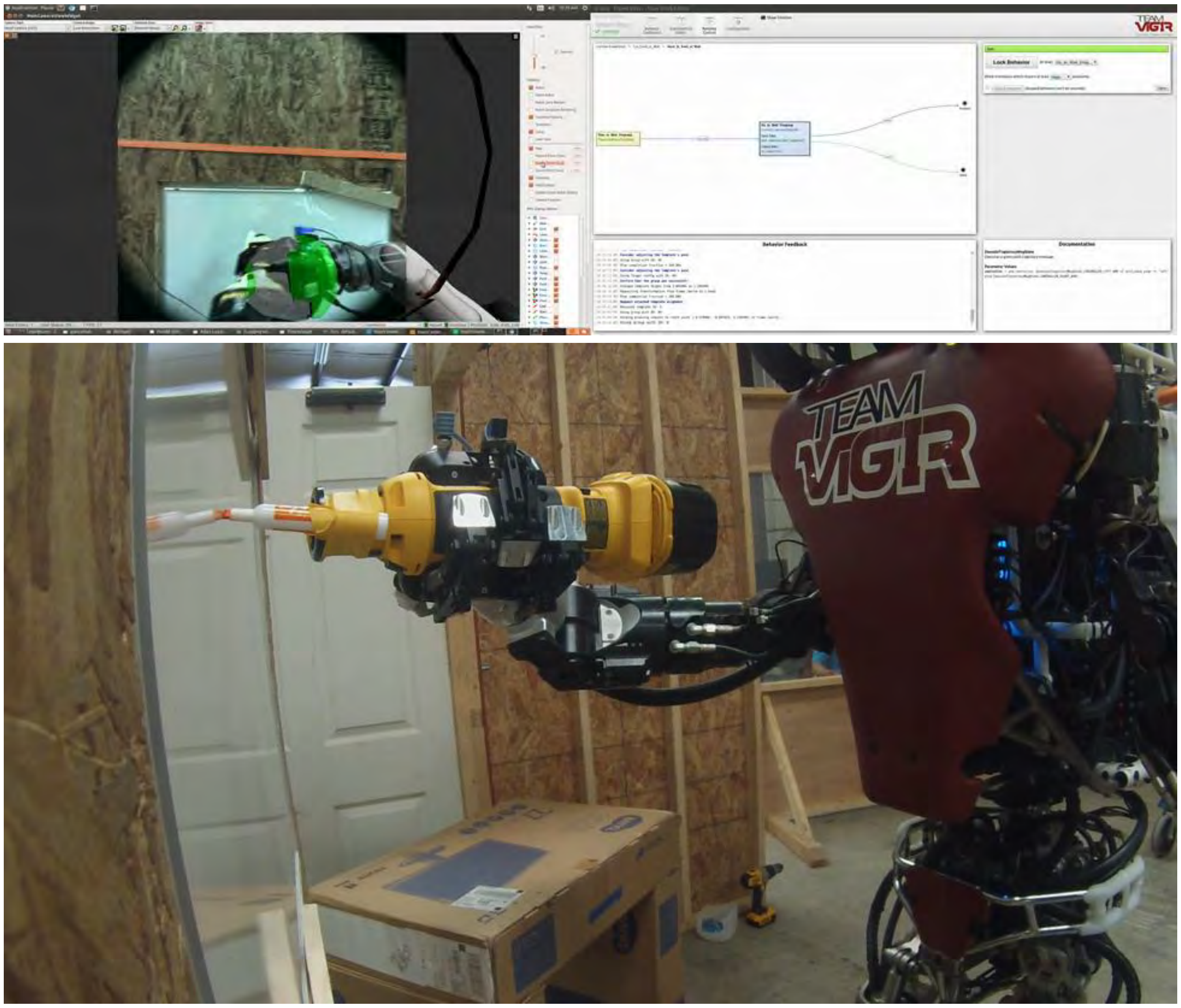


Figure 105. The behavior then moves the cutting tool to a pose in front of the wall

The behavior then moves the cutting tool to a pose in front of the wall, specifically at the top of the circular pattern. It then executes the wall template's "insert" affordance. (Normally, the drill would now penetrate the wall; the dry erase marker has to make contact with the whiteboard, but not push against it too hard. This was not taken into account by the behavior.)

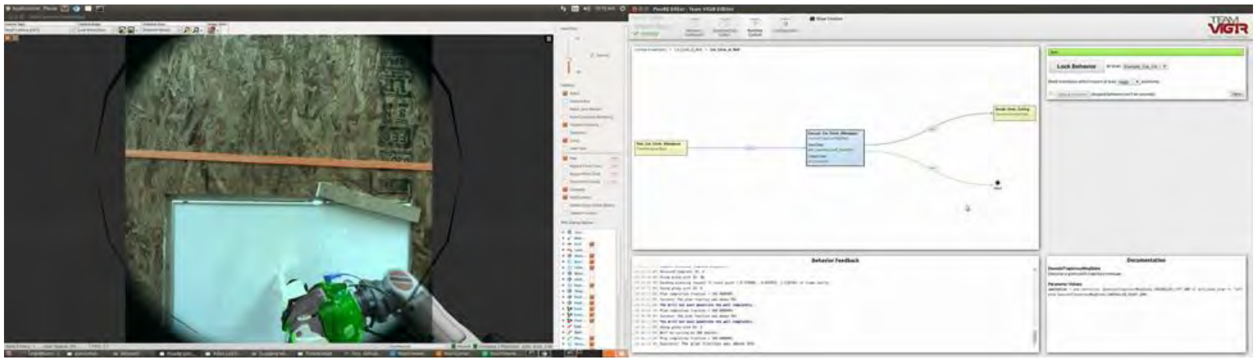


Figure 106. The behavior is executing the “cut_circle” affordance of the wall template.

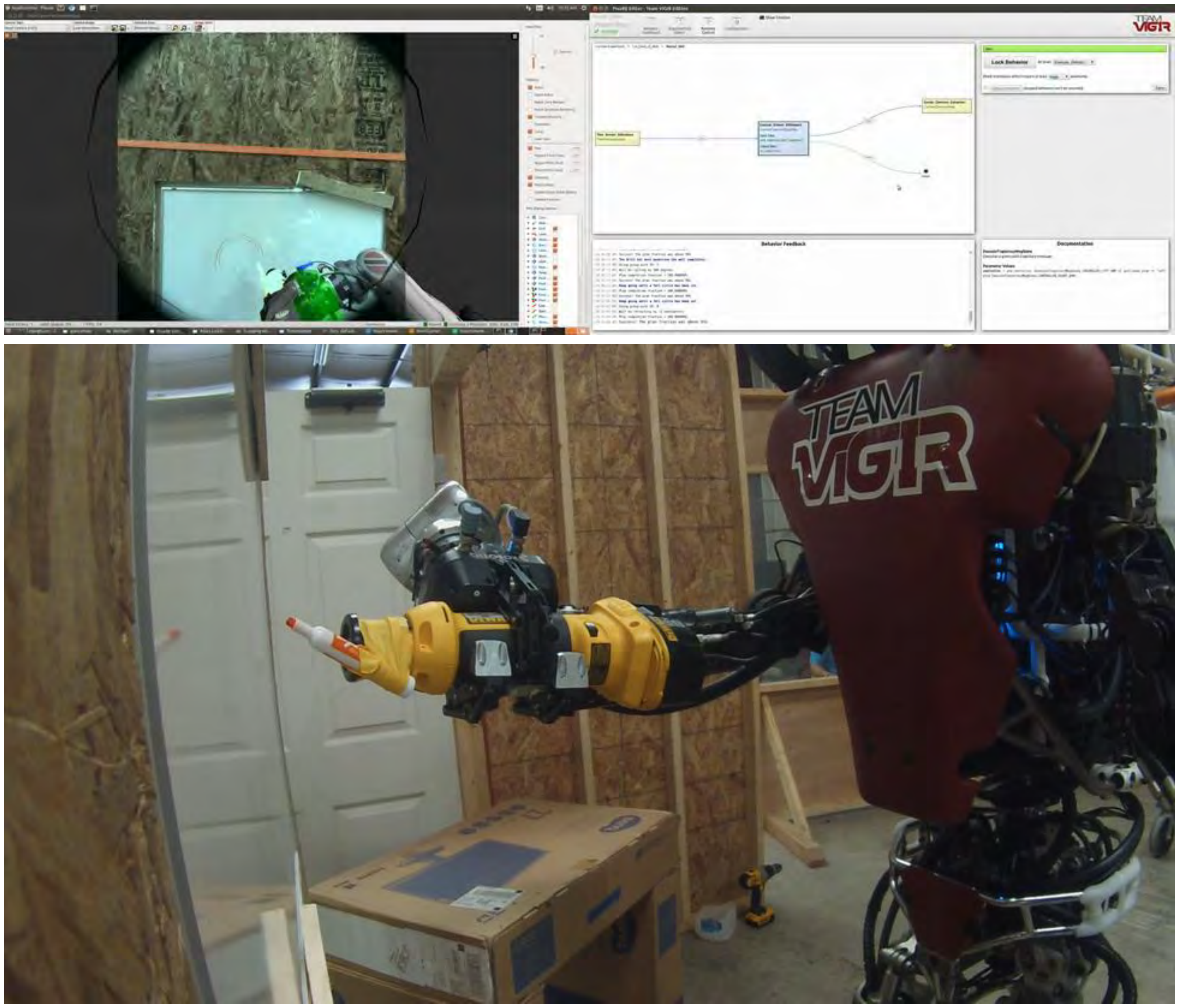


Figure 107. After “cutting”, the behavior executes the negative “insert” affordance.

Once the hole has been cut (circle has been drawn), the behavior executes the “insert” affordance (but with a negative displacement value) in order to retract the cutting tool (top right). The dry erase marker was pushed too hard against the whiteboard and got misaligned (bottom). This contributed to the drawing of only an incomplete circle (top left).

I. BEHAVIOR SYNTHESIS SYSTEM

I.1. Behavior Synthesis from High-level User Specifications

The attached technical report elaborates on the application of our “activation-outcomes” LTL specification paradigm to our ATLAS robot.

I.1.1. Technical Report

Behavior Synthesis for an ATLAS Humanoid Robot from High-level User Specifications

Spyros Maniatopoulos, Philipp Schillinger, Vitchyr H. Pong, David C. Conner, and Hadas Kress-Gazit

I. Problem Statement

Problem 1 (Discrete Abstraction): Given ATLAS' control mode transition constraints and the available actions $\mathcal{A}$, define a discrete abstraction $\mathcal{D}$ of the robot-plus-software system, S, that captures the execution and outcomes of the software implementation of control mode transitions and actions. In addition, maintain a mapping, $\mathcal{D}_M : S \rightarrow \mathcal{AP}$, between the elements of the discrete abstraction and the corresponding software components.

Problem 2 (Formal Task Specification): Given a task, in terms of goals $\mathcal{G}$, the task's initial conditions $\mathcal{I}$, and the discrete abstraction, $\mathcal{D}$, of the system S that is to carry out the task, automatically compile a specification, $\mathcal{T}_S$, that encodes, in a formal language, the task being carried out by S.

II. Discrete Abstraction

A. Control Modes & Actions

We model ATLAS' control mode interface as a transition system $(\mathcal{M}, \rightarrow)$, where $\mathcal{M}$ is the set of states, each corresponding to one control mode, $m \in \mathcal{M}$, and $\rightarrow$ is a set of valid control mode transitions (subset of $\mathcal{M} \times \mathcal{M}$). In addition, we define $Adj(m) = \{m' \in \mathcal{M} \mid (m, m') \in \rightarrow\}$ and also allow self-transitions, i.e., $m \in Adj(m)$, $\forall m \in \mathcal{M}$.

ATLAS can perform actions. Each action, $a \in \mathcal{A}$, is considered discrete and it corresponds to a software component, e.g., generation of a footstep plan or closing the robot's fingers. Actions may also have one or more preconditions. Action preconditions can be control modes or other actions, i.e., $Pre(a) \in 2^{(\mathcal{A} \cup \mathcal{M})}$, $a \notin Pre(a)$, $\forall a \in \mathcal{A}$.

B. Atomic Propositions

We adopt a paradigm that generalizes the one in [1]. We abstract the discrete actions, $a \in \mathcal{A}$, that ATLAS can perform using one system proposition, π_a, per action and one environment proposition, π_a^o, per possible outcome of that action, $o \in Out(a)$. Similarly,[1] for each control mode, $m \in \mathcal{M}$, we have a system proposition π_m and a number of outcome propositions π_m^o. For both actions and control mode transitions, the outcomes that are of most interest in the context of this paper are completion (c) and failure (f) of the action. That is, $Out(a) = Out(m) = \{c, f\}$ Therefore, the set of atomic propositions AP is given by Eq. (1):

$$\mathcal{Y} = \bigcup_{a \in \mathcal{A}} \pi_a \bigcup_{m \in \mathcal{M}} \pi_m, \tag{1a}$$

$$\mathcal{X}' = \mathcal{X} \bigcup_{a \in \mathcal{A}} \bigcup_{o \in Out(a)} \pi_a^o \bigcup_{m \in \mathcal{M}} \bigcup_{o \in Out(m)} \pi_m^o, \tag{1b}$$

where $\mathcal{X}$ are environment propositions other than outcome propositions, e.g., ones that abstract sensors, as per [2].

III. Formal Task Specification

A. Multi-Paradigm Specification

Specifying a robot task in a formal language can be time consuming and error prone. It also requires an expert user. To alleviate these issues, we employ a multi-paradigm specification approach. We first observe that there are portions of the task specification $\mathcal{T}_S$ that are going to be system-specific and portions that are going to be task-specific, such as the task's goals. Intuitively, a user should only have to specify the goals without worrying about the internals of the robot and the software it is running. For ATLAS, we can automatically infer which control modes and actions are pertinent to a task. Finally, the initial conditions are either specified by the user or detected at runtime.

Referring to Problem 2, we get the goals, $\mathcal{G}$, and initial conditions, $\mathcal{I}$, from the user. The discrete abstraction, $\mathcal{D}$, is ATLAS-specific and we assume that it has been created a priori from expert developers, according to Section II. We can now automatically compile the task specification $\mathcal{T}_S$ in (the GR(1) fragment [3] of) Linear Temporal Logic. Since LTL is compositional, we can generate individual LTL formulas and then conjunct them to get the full LTL specification.

[1]The distinction between action and control mode propositions is purely for the sake of clarity of notation. There is nothing special about either.

B. *Specification of Actions and Control Mode Constraints*

1) Generic Formulas: The system safety requirements (2) dictate that an activation proposition should turn `False` once an outcome has been returned.

$$\bigwedge_{o \in Out(a)} \square \left(\pi_a \wedge \bigcirc \pi_a^o \Rightarrow \bigcirc \neg \pi_a \right) \tag{2}$$

The environment safety assumptions (3) dictate that the outcomes of an action are mutually exclusive. For example, an action cannot both succeed and fail.

$$\bigwedge_{o \in Out(a)} \square \left(\bigcirc \pi_a^o \Rightarrow \bigwedge_{o' \neq o} \bigcirc \neg \pi_a^{o'} \right) \tag{3}$$

2) Action-specific Formulas: The environment safety assumptions (4) govern the value of outcomes in the next time step. Specifically, formula (4a) says that if an outcome has been returned, and the corresponding action is re-activated, then any outcome can become `True`. Formula (4b) dictates that, if an outcome is `False` and the corresponding action is not activated, then that outcome should remain `False`. This pair of formulas is a generalization of the "fast-slow" formulas (3) and (4) in [1].

$$\square \left(\bigvee \pi_a^o \wedge \pi_a \Rightarrow \bigvee \bigcirc \pi_a^o \right) \tag{4a}$$

$$\bigwedge_{o \in Out(a)} \square \left(\neg \pi_a^o \wedge \neg \pi_a \Rightarrow \bigcirc \neg \pi_a^o \right) \tag{4b}$$

The environment safety assumptions (5) dictate that the value of an outcome should not change if the corresponding action has not been activated again. In other words, the outcome persists.

$$\bigwedge_{o \in Out(a)} \square \left(\pi_a^o \wedge \neg \pi_a \Rightarrow \bigcirc \pi_a^o \right) \tag{5}$$

The environment liveness assumption (6c) is a fairness condition. It states that, (always) eventually, either the activation of an action will return an outcome, (6a), or that the robot will "change its mind", (6b). Formula (6a) is a generalization of $\varphi_a^{completion}$ in [1], whereas formula (6b) is exactly the same as φ_a^{change} in [1], since it consists of activation propositions only.

$$\varphi_a^{return} = \left(\pi_a \wedge \bigvee \bigcirc \pi_a^o \right) \vee \left(\neg \pi_a \wedge \bigwedge \bigcirc \neg \pi_a^o \right) \tag{6a}$$

$$\varphi_a^{change} = \left(\pi_a \wedge \bigcirc \neg \pi_a \right) \vee \left(\neg \pi_a \wedge \bigcirc \pi_a \right) \tag{6b}$$

$$\square \diamond \left(\varphi_a^{return} \vee \varphi_a^{change} \right) \tag{6c}$$

The system safety requirement (7) demonstrates how a formula encoding the preconditions of an action, $Pre(a)$, looks like in the activation–outcomes paradigm.

$$\square (\bigvee_{p \in Pre(a)} \neg \pi_p^c \Rightarrow \neg \pi_a) \tag{7}$$

where the superscript $c \in Out(p)$ stands for "completion".

3) Control Mode Formulas: For brevity of notation, let $\varphi_m = \pi_m \wedge \bigwedge_{m' \neq m} \neg \pi_{m'}$. Activating φ_m takes into account the mutual exclusion between the control modes $m \in \mathcal{M}$.

The system safety requirements (8) encode a topological transition relation, such the BDI control mode transition system.

$$\bigwedge_{m \in \mathcal{M}} \square \left(\bigcirc \pi_m^c \Rightarrow \bigvee_{m' \in Adj(m)} \bigcirc \varphi_{m'} \vee \bigcirc \varphi_{\mathcal{M}}^{none} \right) \tag{8}$$

where $\varphi_{\mathcal{M}}^{none} = \bigwedge_{m \in \mathcal{M}} \neg \pi_m$ being `True` stands for not activating any control mode transitions, i.e., staying in the same control mode.

The environment safety assumptions (9) enforce mutual exclusion between the BDI control modes.

$$\bigwedge_{m \in \mathcal{M}} \square \left(\bigcirc \pi_m^c \Leftrightarrow \bigwedge_{m' \neq m} \bigcirc \neg \pi_{m'}^c \right) \tag{9}$$

The environment safety assumptions (10) govern how the active control mode can change in a single time step in response to the activation of a control mode transition.

$$\bigwedge_{m\in\mathcal{M}} \bigwedge_{m'\in Adj(m)} \square\left(\pi^c_m \wedge \varphi_{m'} \Rightarrow \left(\bigcirc \pi^c_m \bigvee_{o\in Out(m')} \bigcirc \pi^o_{m'}\right)\right) \tag{10}$$

The environment safety assumptions (11) constrain the outcomes control mode transitions.

$$\bigwedge_{m\in\mathcal{M}} \bigwedge_{o\in Out(m)} \square\left(\neg\pi^o_m \wedge \neg\pi_m \Rightarrow \bigcirc \neg\pi^o_m\right) \tag{11}$$

The environment safety assumptions (12) dictate that the value of the outcomes of control mode transitions must not change if no transition is being activated, i.e., they must persist.

$$\bigwedge_{m\in\mathcal{M}} \bigwedge_{o\in Out(m)} \square\left(\pi^o_m \wedge \varphi^{none}_{\mathcal{M}} \Rightarrow \bigcirc \pi^o_m\right) \tag{12}$$

The environment liveness assumption (13c) is the equivalent of the fairness condition (6c) for control mode propositions.

$$\varphi^{return}_{\mathcal{M}} = \bigvee_{m\in\mathcal{M}} \left(\varphi_m \wedge \bigvee_{o\in Out(m)} \bigcirc \pi^o_m\right) \tag{13a}$$

$$\varphi^{change}_{\mathcal{M}} = \bigvee_{m\in\mathcal{M}} \left(\varphi_m \wedge \bigcirc \neg\varphi_m\right) \tag{13b}$$

$$\square\Diamond\left(\varphi^{return}_{\mathcal{M}} \vee \varphi^{change}_{\mathcal{M}} \vee \varphi^{none}_{\mathcal{M}}\right) \tag{13c}$$

C. Specification of Initial Conditions

For each action, a, and control mode, m, in the initial conditions, $\mathcal{I}$, the completion proposition should be `True` in the environment initial conditions (14). All other outcome propositions corresponding to those actions and control modes, as well as all outcome propositions corresponding to any other actions and control modes, should be `False`.

$$\varphi^e_i = \bigwedge_{i\in\mathcal{I}} \left(\pi^c_i \bigwedge_{o\in Out(i)\setminus\{c\}} \neg\pi^o_i\right) \wedge \bigwedge_{j\notin\mathcal{I}} \bigwedge_{o\in Out(j)} \neg\pi^o_j \tag{14}$$

Activation propositions are `False` regardless of whether that action or control mode is in the initial conditions or not (15). The reason being that, intuitively, if we want something to be an initial condition, then we shouldn't have the resulting controller re-activate it at the beginning of execution.

$$\varphi^s_i = \bigwedge_{i\in\mathcal{I}} \neg\pi_i \wedge \bigwedge_{j\notin\mathcal{I}} \neg\pi_j \wedge \bigwedge_{g\in\mathcal{G}} \neg\mu_g \tag{15}$$

D. Specification of Task Goals

The system initial condition (16), safety requirements (17) and (18), and liveness requirement (19) are used to reason about the satisfaction of the system's goals, $g \in \mathcal{G}$, in a finite run (as opposed to infinite execution, which is what LTL is defined over). In this finite run paradigm, the synthesized state machine (SM) itself has outcomes, $o \in Out(SM)$. The propositions corresponding to the SM's outcomes, π^o_{SM}, are system, not environment, propositions. The system propositions, μ_g, serve as memory of having accomplished each goal (c.f. [4]).

$$\bigwedge_{g\in\mathcal{G}} \neg\mu_g \tag{16}$$

$$\bigwedge_{g\in\mathcal{G}} \square\left(\bigcirc \pi^c_g \Rightarrow \bigcirc \mu_g\right) \tag{17a}$$

$$\bigwedge_{g\in\mathcal{G}} \square\left(\mu_g \Rightarrow \bigcirc \mu_g\right) \tag{17b}$$

$$\bigwedge_{g\in\mathcal{G}} \square\left(\neg\mu_g \wedge \bigcirc \neg\pi^c_g \Rightarrow \bigcirc \neg\mu_g\right) \tag{17c}$$

$$\square\left(\pi^{c}_{SM} \Leftrightarrow \bigwedge_{g\in\mathcal{G}} \mu_g\right) \tag{18a}$$

$$\square\left(\pi^{f}_{SM} \Leftrightarrow \bigvee_{\pi\in\mathcal{Y}} \pi^{f}\right) \tag{18b}$$

$$\bigwedge_{o\in Out(SM)} \square\left(\pi^{o}_{SM} \Rightarrow \bigcirc \pi^{o}_{SM}\right) \tag{18c}$$

$$\square\Diamond\left(\bigvee_{o\in Out(SM)} \pi^{o}_{SM}\right) \tag{19}$$

Formulas (17) do not guarantee that the goals will be achieved in a specific order. However, that is often desirable. To account for it, we can define the goals as an ordered set $\mathcal{G} = \{g_1, g_2, \dots, g_n\}$, where $g_i < g_j$ for $i < j$, and the relation $g_i < g_j$ means that goal g_i has to be achieved before g_j. With this definition, we can add the optional safety requirement (20), whenever strict goal order is desired.

$$\bigwedge_{i=1}^{n} \square\left(\neg\mu_{g_{i-1}} \Rightarrow \bigcirc \neg\mu_{g_i}\right),\ \mu_{g_0} \triangleq \texttt{True} \tag{20}$$

Finally, these auxiliary (memory and SM outcome) propositions have to be added to the system propositions:

$$\mathcal{Y}' = \mathcal{Y} \bigcup_{g\in\mathcal{G}} \mu_g \bigcup_{o\in Out(SM)} \pi^{o}_{SM}$$

References

[1] V. Raman, N. Piterman, and H. Kress-Gazit, "Provably Correct Continuous Control for High-Level Robot Behaviors with Actions of Arbitrary Execution Durations," in *IEEE Int'l. Conf. on Robotics and Automation*, 2013.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *VMCAI*, Charleston, SC, January 2006, pp. 364–380.

[4] V. Raman, B. Xu, and H. Kress-Gazit, "Avoiding forgetfulness: Structured English specifications for high-level robot control with implicit memory," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2012.

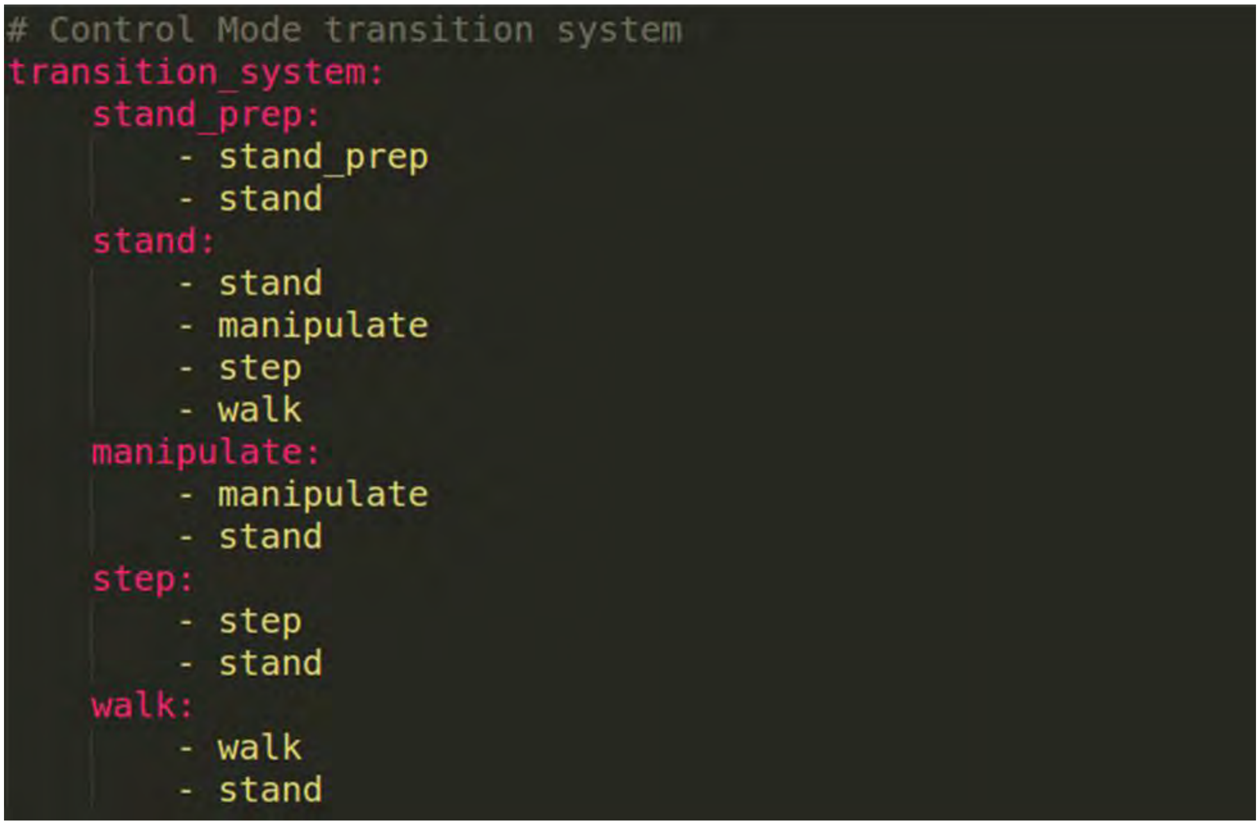
I.2. Experimental Demonstration of Behavior Synthesis

We first provide an overview of the lab setup and software configuration for the Behavior Synthesis experiments. Then, we present three experimental demos. Two demonstrate synthesis "starting from scratch", whereas the third demonstrates the use of synthesis to modify an existing behavior "on-the-fly", i.e., while the initial behavior is being executed on ATLAS.

I.2.1. Experimental Setup

The lab setup for the Behavior Synthesis demonstration was as follows. ATLAS was positioned in front of a table and a cutting tool was placed on the table. Calibration of the electric and hydraulic joints had been performed in advance. A hardware issue with our ATLAS' left hip prevented any demo that involved walking or stepping, so all of the demos below only involve manipulation. Moreover, a single operator was performing the synthesis, behavior execution, and perception tasks on a single OCS computer.

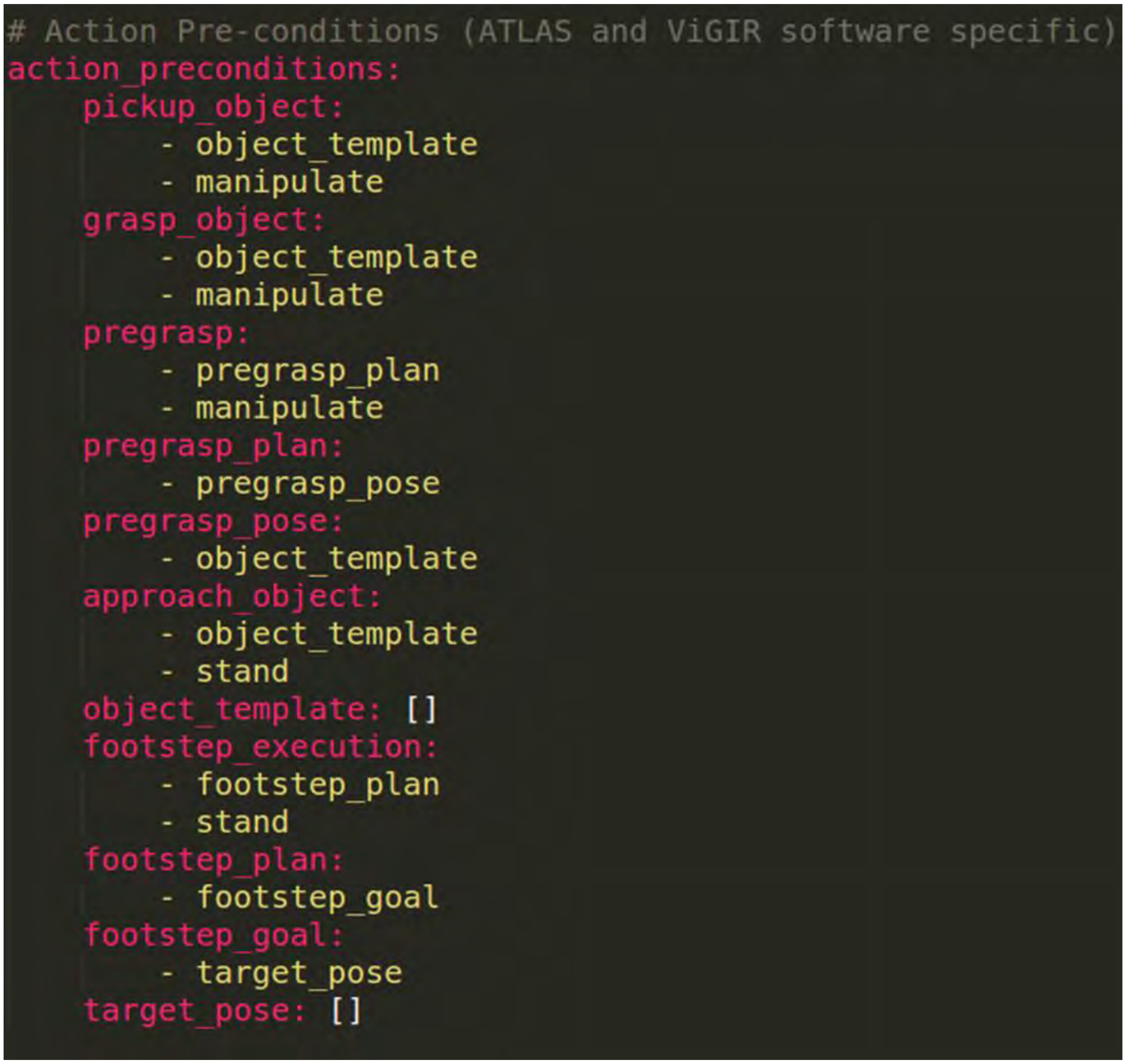
In addition to the partial specification provided by the user (initial conditions and goals), the LTL Compilation service takes into account the BDI control mode transition system as well as the preconditions of the various actions. For the purposes of these demos, these are specified in configuration files (see Figures Figure 108 and Figure 109). The configuration files were written a priori and did not have to change in between runs or demos. The user does have to use the same keywords as the configuration files when inputting the high-level specification (e.g. “stand”, “grasp_object”). Finally, a separate configuration file served as a mapping between these keywords (the atomic propositions) and the state primitives (see Appendix H). An excerpt is depicted in Figure 110.



```
# Control Mode transition system
transition_system:
    stand_prep:
        - stand_prep
        - stand
    stand:
        - stand
        - manipulate
        - step
        - walk
    manipulate:
        - manipulate
        - stand
    step:
        - step
        - stand
    walk:
        - walk
        - stand
```

Figure 108. BDI control mode constraints encoded as a transition system.

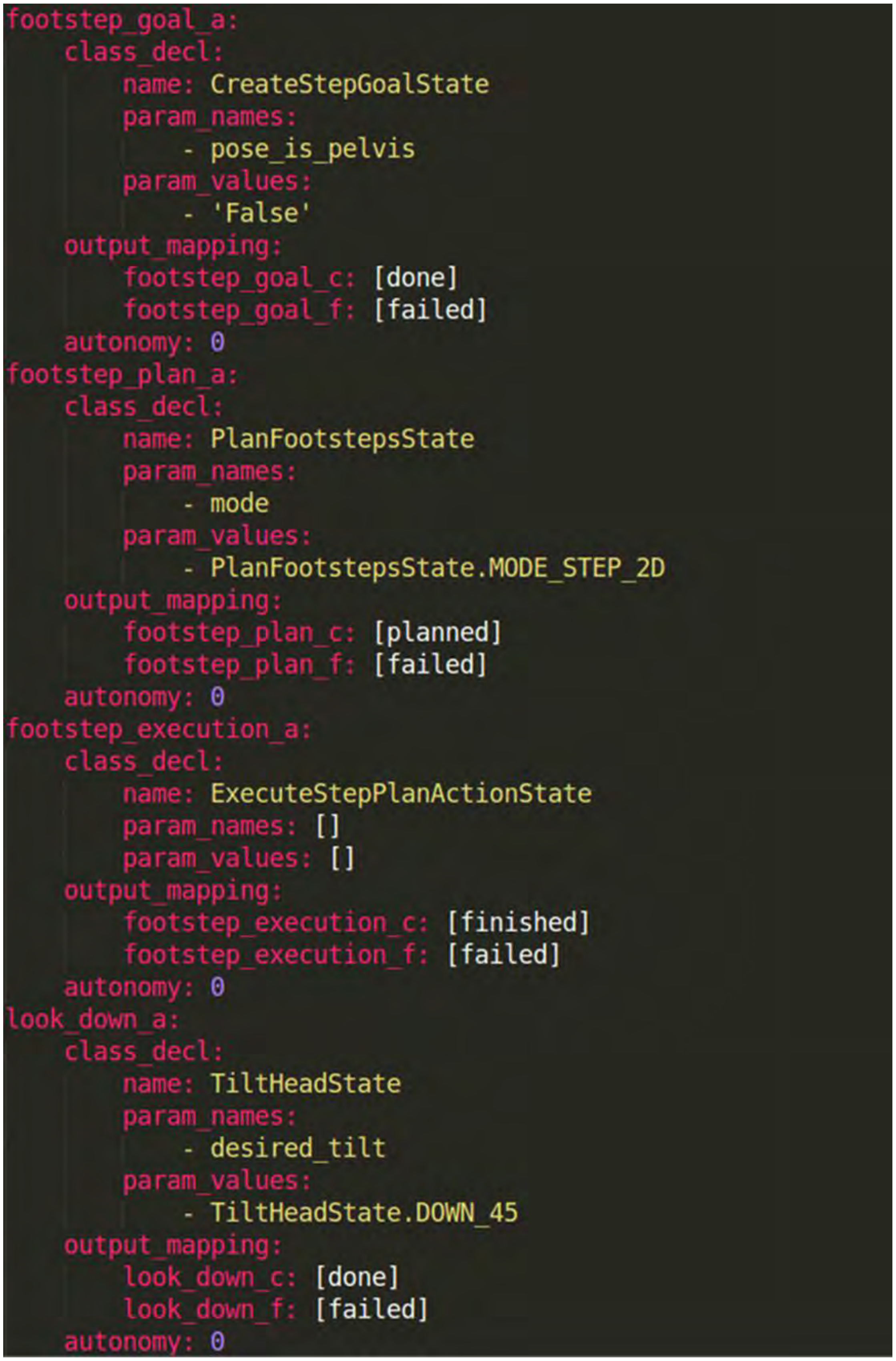
For each control mode (depicted in purple), the allowed control mode transitions are listed below (yellow).



```
# Action Pre-conditions (ATLAS and ViGIR software specific)
action_preconditions:
    pickup_object:
        - object_template
        - manipulate
    grasp_object:
        - object_template
        - manipulate
    pregrasp:
        - pregrasp_plan
        - manipulate
    pregrasp_plan:
        - pregrasp_pose
    pregrasp_pose:
        - object_template
    approach_object:
        - object_template
        - stand
    object_template: []
    footstep_execution:
        - footstep_plan
        - stand
    footstep_plan:
        - footstep_goal
    footstep_goal:
        - target_pose
    target_pose: []
```

Figure 109. Action preconditions.

The actions are depicted in purple and their preconditions are listed in yellow. The empty brackets ("[ ]") denote that these actions do not have any preconditions. Alternatively, they could have been omitted from this configuration file altogether.



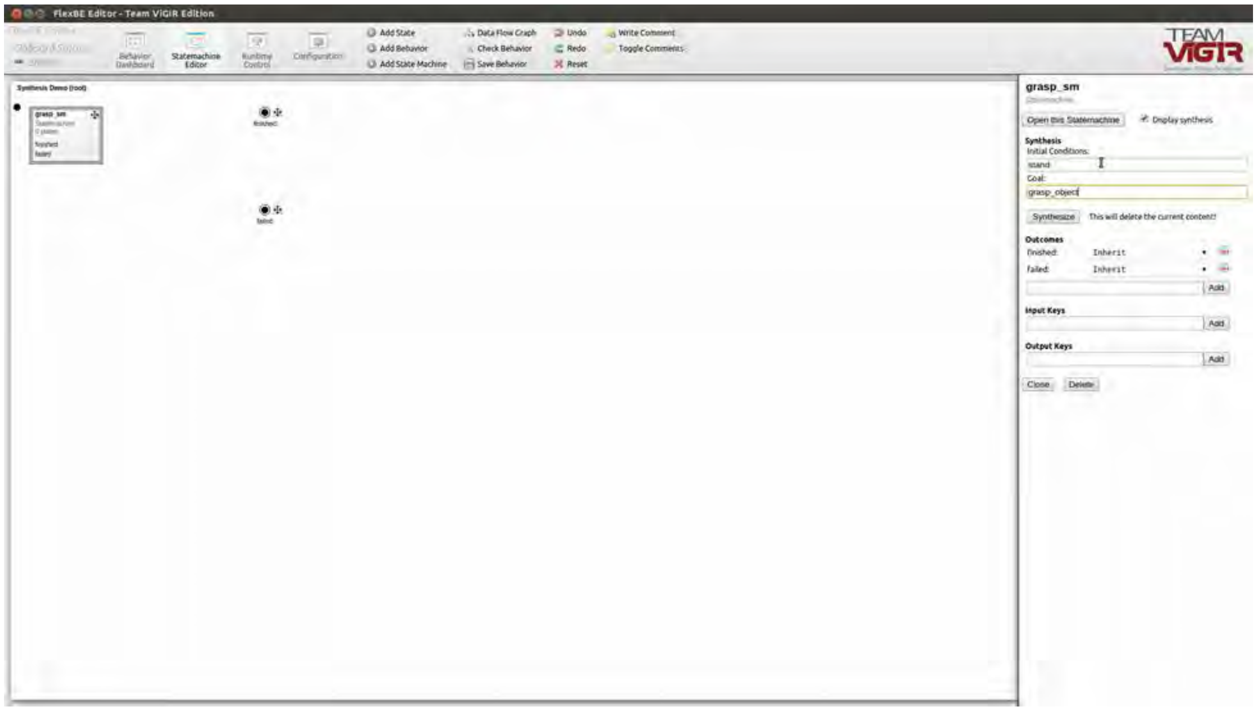
```yaml
footstep_goal_a:
    class_decl:
        name: CreateStepGoalState
        param_names:
            - pose_is_pelvis
        param_values:
            - 'False'
    output_mapping:
        footstep_goal_c: [done]
        footstep_goal_f: [failed]
    autonomy: 0
footstep_plan_a:
    class_decl:
        name: PlanFootstepsState
        param_names:
            - mode
        param_values:
            - PlanFootstepsState.MODE_STEP_2D
    output_mapping:
        footstep_plan_c: [planned]
        footstep_plan_f: [failed]
    autonomy: 0
footstep_execution_a:
    class_decl:
        name: ExecuteStepPlanActionState
        param_names: []
        param_values: []
    output_mapping:
        footstep_execution_c: [finished]
        footstep_execution_f: [failed]
    autonomy: 0
look_down_a:
    class_decl:
        name: TiltHeadState
        param_names:
            - desired_tilt
        param_values:
            - TiltHeadState.DOWN_45
    output_mapping:
        look_down_c: [done]
        look_down_f: [failed]
    autonomy: 0
```

Figure 110. Excerpt from the mapping between atomic propositions and FlexBE state primitives.

I.2.2. Demo #1: Behavior Synthesis with a single goal

Parameters:

- Initial control mode: STAND
- Goals: “grasp object”



grasp_sm

Synthesis
Initial Conditions: stand
Goal: grasp_object

Outcomes
finished: Inherit
failed: Inherit

Input Keys

Output Keys

Figure 111. The user is specifying the initial condition (STAND) and final goal (“grasp object”).

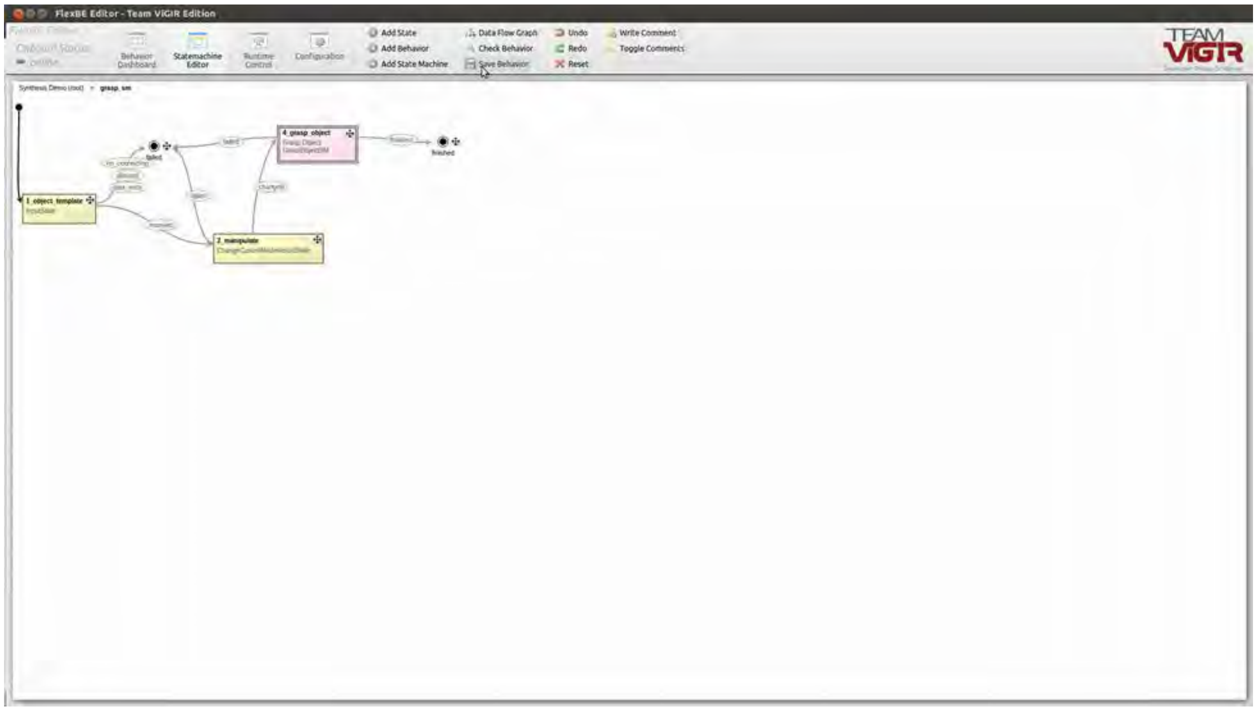


Figure 112. The resulting synthesized state machine includes the preconditions of grasping.

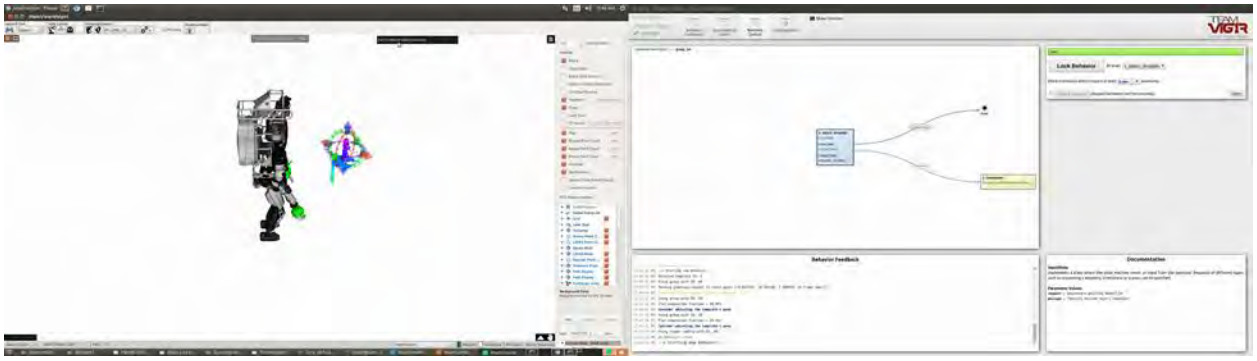


Figure 113. The synthesized state machine is ready to be executed.

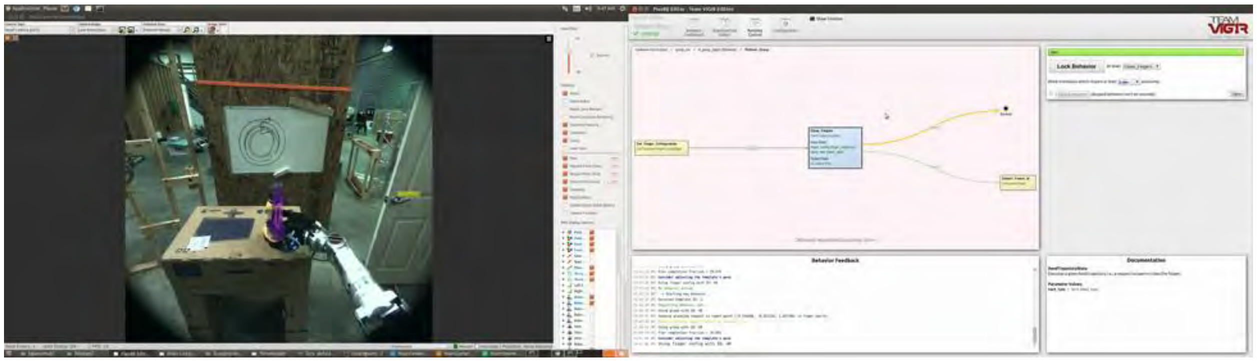
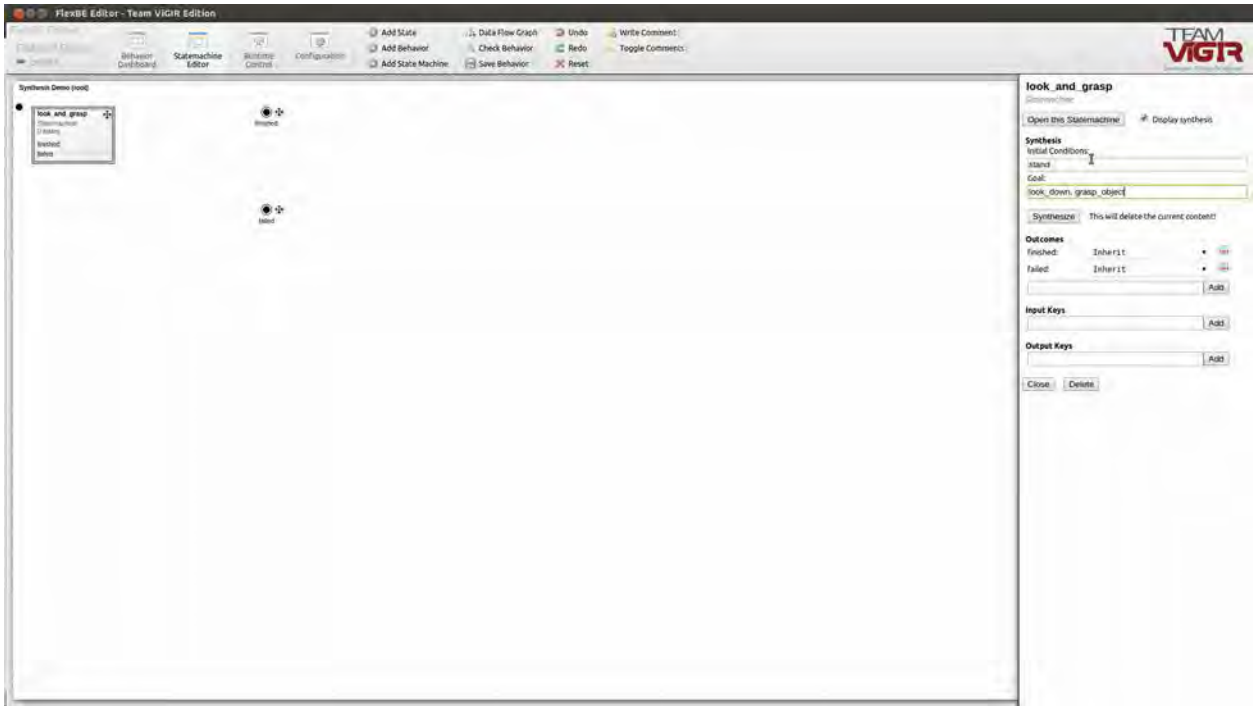


Figure 114. The final goal ("grasp object") has been accomplished.

I.2.3. Demo #2: Behavior Synthesis with multiple goals

Parameters:

- Initial control mode: STAND
- Goals: "look down", "grasp object"



look_and_grasp

Synthesis
Initial Conditions: stand
Goal: look_down, grasp_object

This will delete the current content!

Outcomes
finished: Inherit
failed: Inherit

Input Keys

Output Keys

Figure 115. The user is specifying two goals ("look down" and "grasp object").

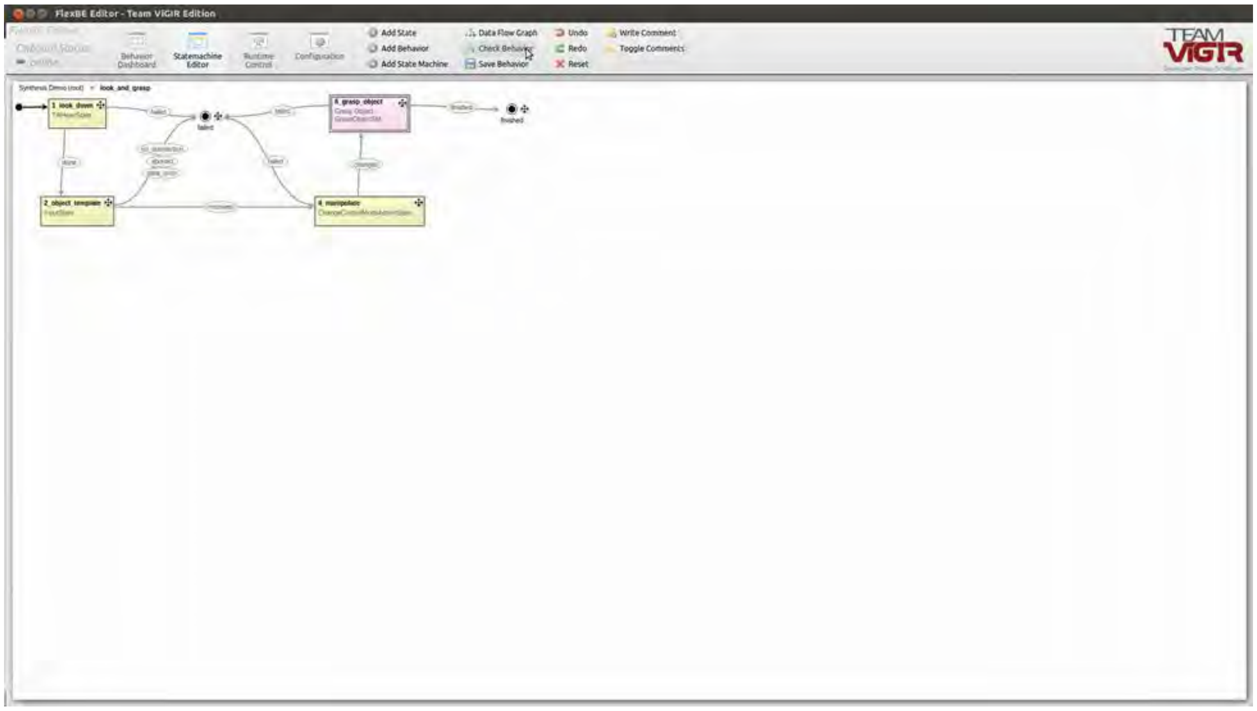


Figure 116. The resulting state machine starts with “look down”, then proceeds as in Demo #1.

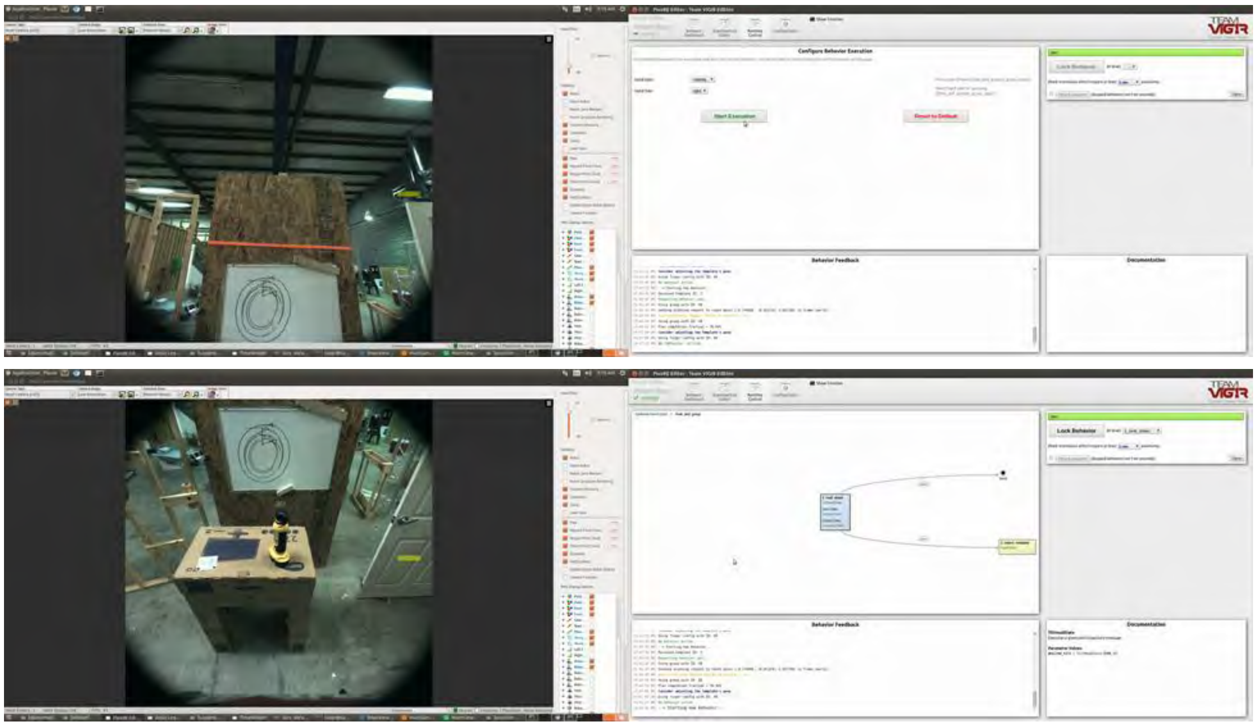

FlexBE Editor - Team ViGIR Edition
Configure Behavior Execution
Start Execution
Behavior Feedback
Documentation
Lock Behavior

Figure 117. Atlas executing the “look down” behavior.

ATLAS’ neck was tilted upwards before behavior execution (top). As specified by the user (“look_down”), the synthesized behavior first tilted the neck down, which brought the object of interest within the camera’s field of view (bottom).

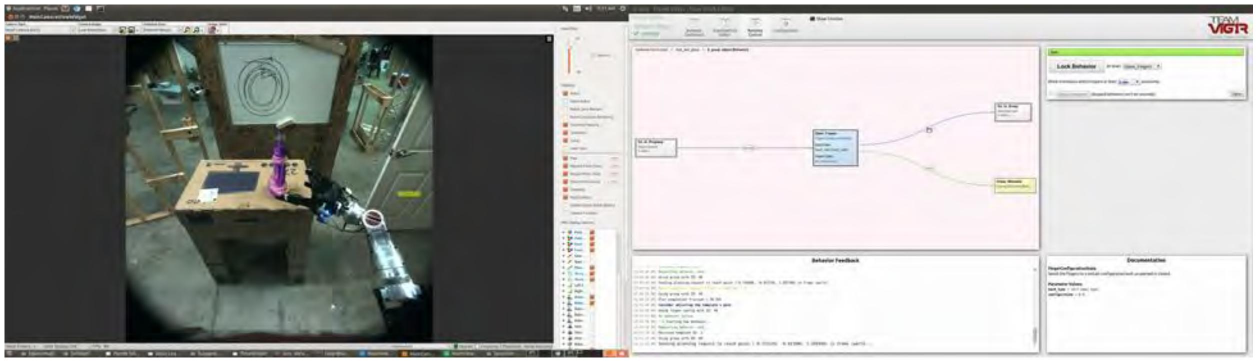


Figure 118. Execution of the synthesized state machine proceeds as in Demo #1.

I.2.4. Demo #3: Behavior Synthesis "on-the-fly" via Runtime Modification

Parameters:

- Initial behavior: "Pick up Tool"
- Initial control mode (when locked): MANIPULATE
- Goals: "footstep_execution"

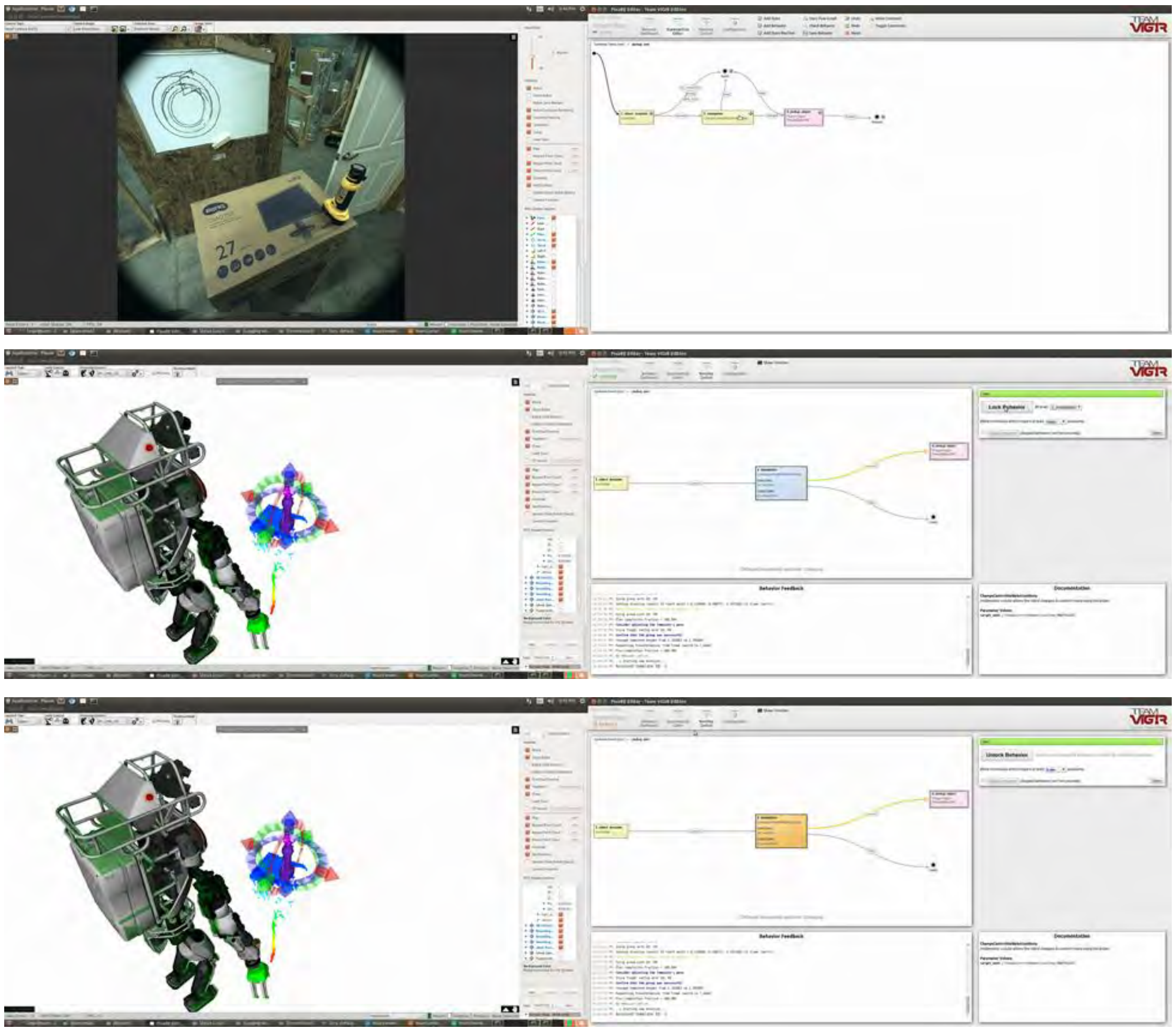


Figure 119. Changing behavior during execution

The initially executed behavior (top) involves picking up the cutting tool. Once execution reaches the transition to MANIPULATE, the behavior is "locked" (middle and bottom).

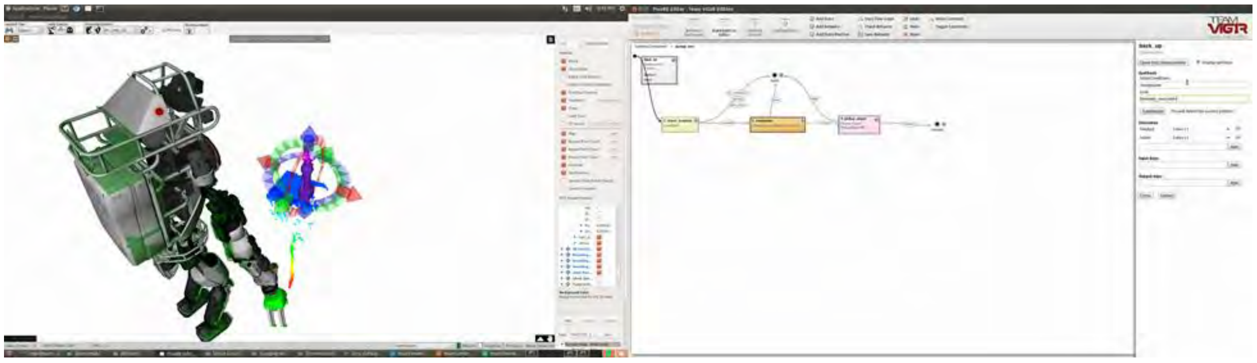


Figure 120. With behavior execution locked, the user switches to the Editor window

With behavior execution locked, the user switches to the Editor window and specifies the initial condition (MANIPULATE) and goal ("footstep execution") of a new state machine.

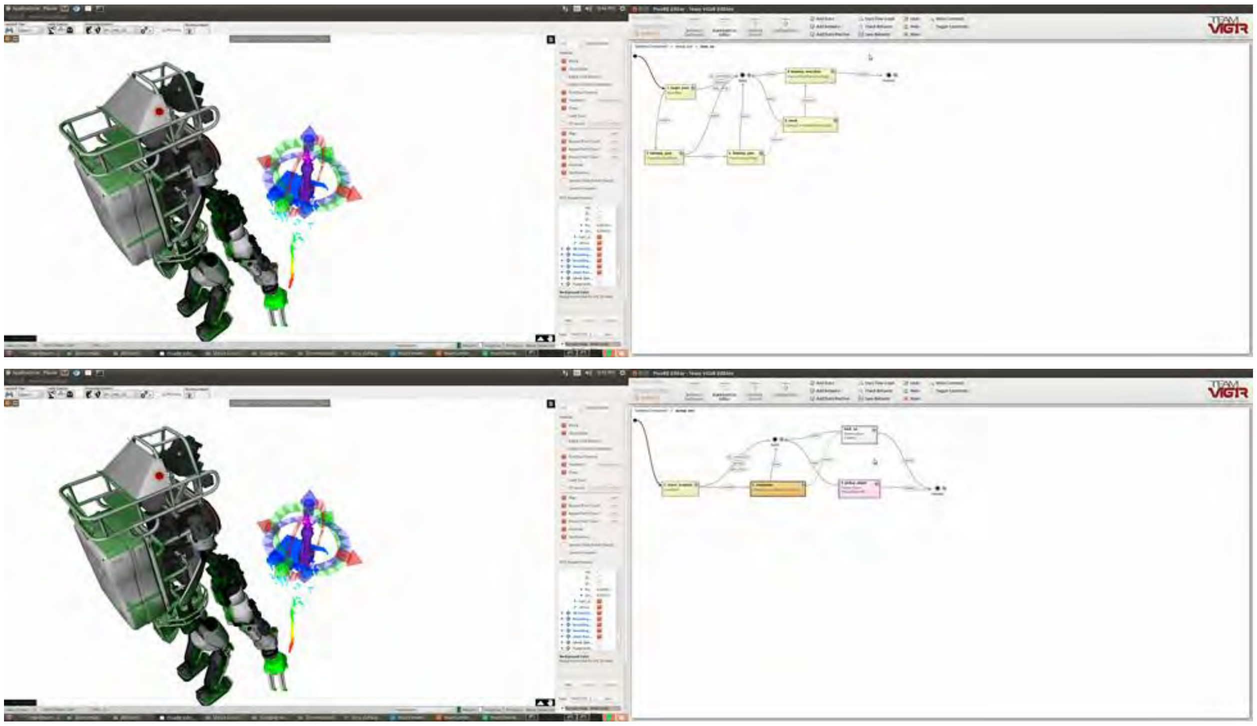


Figure 121. The new, synthesized state machine (top) is connected to the initial behavior (bottom).

Specifically, the transition leading from MANIPULATE (the "pivot" state, depicted in orange) to "pick up object" now leads to "back up", the synthesized state machine.

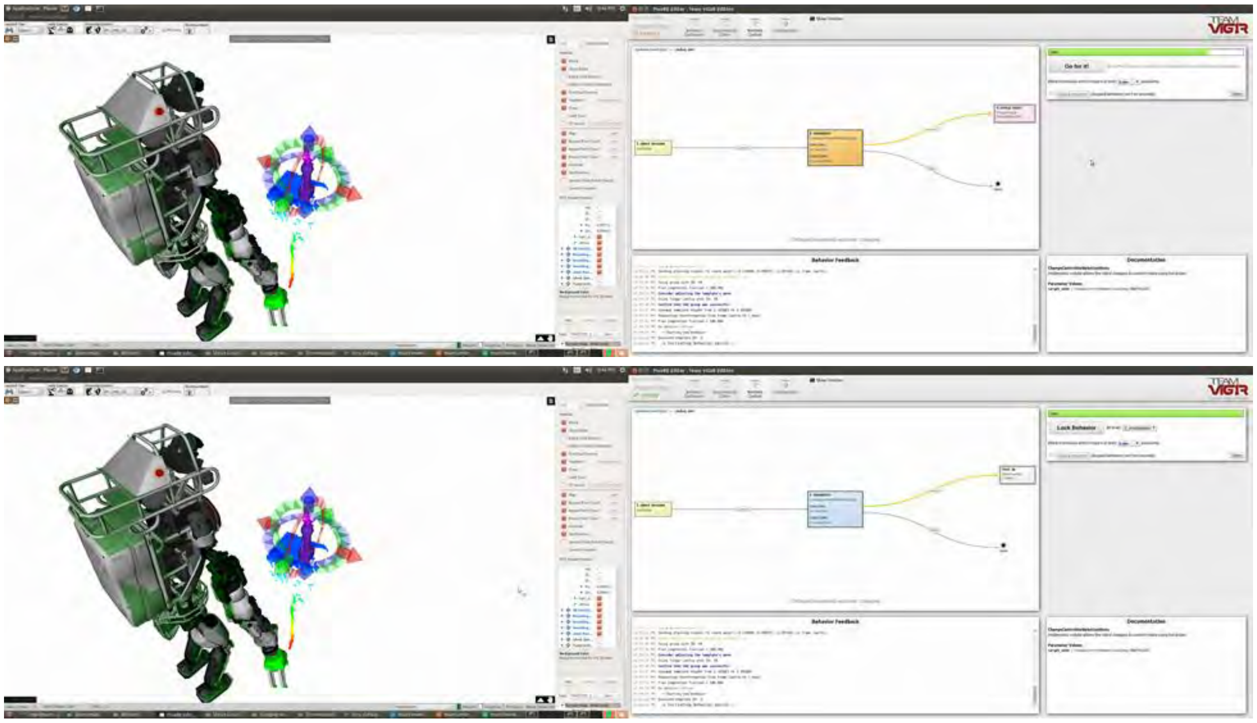


Figure 122. The modified behavior is saved and the user resumes execution.

The user resumes execution by clicking on the "Go for it!" button. Note how the transition that originally led to "pick up object" (top) now leads to "back up" (bottom).

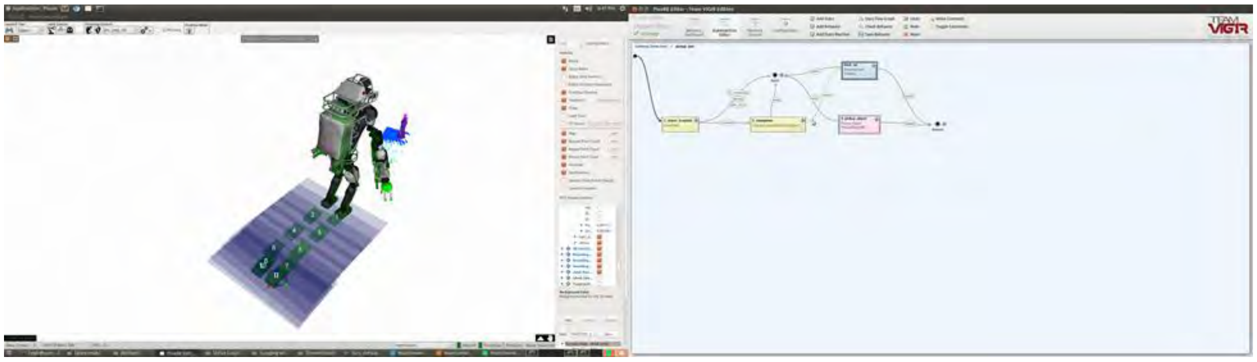


Figure 123. Execution has resumed and the synthesized state machine (blue) is executed.

J. OPEN SOURCE SOFTWARE GUIDE

J.1. Installation

Team ViGIR's open source software is provided with a focus on accessibility for interested developers. The complexity of comprehensive robot software can lead to deterrence of otherwise interested researchers if the usage instructions are very complex and/or error prone. For this reason, we provide a install and usage scripts that allow installing and running all software with few terminal commands. The most recent installation instructions can always be found in the vigir_rosinstall package.

J.2. Components

In the following, we provide a brief overview of important ROS packages and the provided capabilities.

J.2.1. Infrastructure

vigir_rosinstall: Provides the installation scripts that allow a quick and convenient install of all Team ViGIR software.

vigir_scripts: Provides helper scripts that support managing the Team ViGIR workspace

vigir_atlas_common: Provides the ATLAS robot model and related model data such as for hands and Multisense sensor head.

J.2.2. Robot Control

vigir_ros_control: Provides generic interfaces and tools for humanoid robot control. The packages in this repository are robot-agnostic and can be used as a toolbox to create controllers for humanoid robots. Examples of provided capabilities are inverse dynamics (gravity compensating) controllers, friction compensating controllers and trajectory smooting functionality.

vigir_atlas_ros_control: Provides ATLAS-specific robot control software. This repository contains the main ATLAS controller as used by Team ViGIR and controllers for impedance control of ATLAS, as well as tools related to low level control. This package is NOT opensourced due to proprietary BDI information.

J.2.3. Hardware Drivers

The hardware driver for the Multisense SL sensor head is used as provided by Carnegie Robotics, thus no dedicated repository is used for it.

vigir_pgr_camera: Provides a driver for the Blackfly cameras used as SA cameras on ATLAS. The driver provides a region of interest of the wide angle lens rotated to be upright and without black borders.

vigir_grasp_control: Provides an interface to allow trajectories and actions to be sent to the various hand types the robot supports.

vigir_manipulation_controller: Provides a controller for handling and interacting with various object templates. This package also manages the various types of hands themselves.

J.2.4. Perception

vigir_wide_angle_image_proc: Provides a ROS nodelet for rectifying high distortion fisheye camera images such as those provided by the ATLAS SA cameras.

vigir_perception: Provides major perception components. This includes tools for scan compression and transmission over constrained connections, generation of meshes from depth image and point cloud data as well as the main worldmodel server node.

J.2.5. Motion Planning

vigir_manipulation_planning: Provides main motion planning capabilities. this includes custom move_group components, the LIDAR octomap updater plugin and the affordance template system described in [Section Manipulation]

J.2.6. Behavior Control

flexbe_behvaior_engine: Contains the Flexbe Behavior Engine (FlexBE) ROS Packages.

flexbe_chrome_app: HTML/CCS/Javascript source code of the FlexBE GUI, a Google Chrome app.

vigir_behaviors: Team and robot specific part of FlexBE such as additional states for interfacing with Team ViGIR software. It also contains all of the implemented behaviors.

vigir_behavior_synthesis: ROS packages that enable the automatic synthesis of executable state machines.

ReSpeC: Reactive (LTL) Specification Construction kit. A ROS-independent Python framework used by vigir_behavior_synthesis.

J.2.7. Operator Control Station

vigir_ocs_common: Provides main operator control station capabilities. The three main widgets are contained in this repository, along with the various specialized task-specific user interface elements.

vigir_rviz: Provides a fork of the rviz visualization tool commonly used with ROS. The intention is to merge modifications by Team ViGIR upstream into the ROS version and remove the vigir_rviz repository afterwards.

This page intentionally blank.

K. LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS