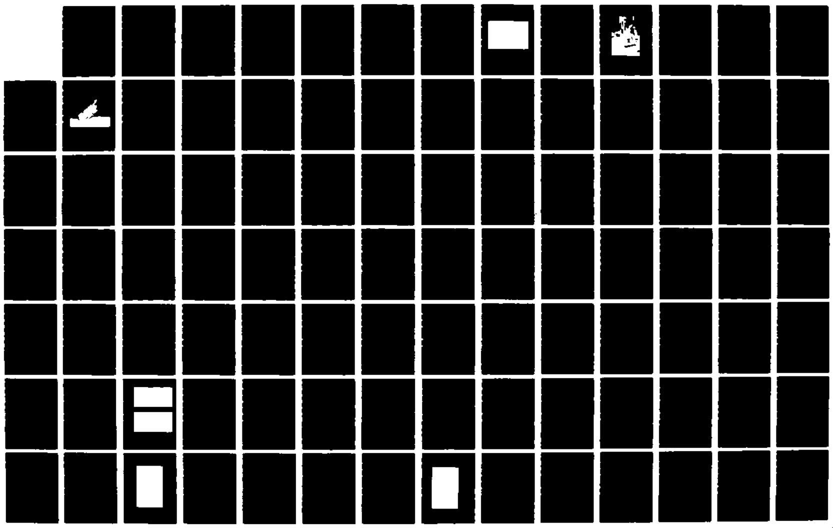
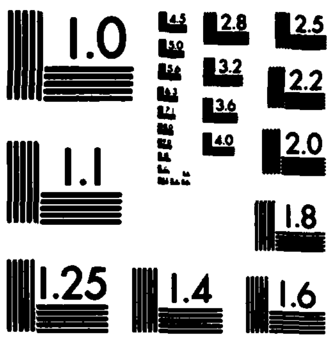
AD-A148 174 AN EXPERIMENTAL STUDY OF AN ULTRA-MOBILE VEHICLE FOR OFF-ROAD TRANSPORTATION(U) OHIO STATE UNIV RESEARCH FOUNDATION COLUMBUS R B MCGHEE ET AL. SEP 84 1/3

UNCLASSIFIED MDA903-82-K-0058 F/G 6/4 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

0

RF Project 762945/714250
Semi-Annual Report

AN EXPERIMENTAL STUDY OF AN ULTRA-MOBILE VEHICLE FOR OFF-ROAD TRANSPORTATION

Robert B. McGhee and Kenneth J. Waldron
College of Engineering

For the Period
October 1, 1983 - March 31, 1984

DEFENSE SUPPLY SERVICE
Washington, D.C. 20310

Contract No. MDA903-82-K-0058

September, 1984

OSU

The Ohio State University
Research Foundation

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A148 174
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (*and Subtitle*) AN EXPERIMENTAL STUDY OF AN ULTRA-MOBILE VEHICLE FOR OFF-ROAD TRANSPORTATION	5. TYPE OF REPORT & PERIOD COVERED Semi-Annual Technical Report 10/1/83-3/31/84
	6. PERFORMING ORG. REPORT NUMBER 762945/714250
7. AUTHOR(*s*) Robert B. McGhee and Kenneth J. Waldron	8. CONTRACT OR GRANT NUMBER(*s*) MDA903-82-K-0058
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Ohio State University Research Foundation, 1314 Kinnear Road Columbus, Ohio 43212	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DEPARTMENT OF THE ARMY, Defense Supply Service Room ID-245, The Pentagon Washington, D.C. 20310	12. REPORT DATE September, 1984
	13. NUMBER OF PAGES 260
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*) DARPA 1400 Wilson Bl Arlington VA 22209	15. SECURITY CLASS. (*of this report*) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

[illegible] (A)

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Walking machine
Robotics
Off-road vehicles
Legged locomotion

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

This report summarizes research accomplished during the fifth six-month period of this three-year project. The research is concerned with the design and construction of an experimental off-road vehicle called the "ASV-84." The vehicle concept envisages locomotion without the use of wheels or tracks, but rather employs six active suspension elements, each providing support and propulsion analogously to a biological leg. The report includes test results obtained with a prototype hydraulically-powered vehicle leg making use of an
(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

Block 20 (Abstract) - Continued

innovative hydrostatic system with one variable displacement pump for each actuator. This system is designed to permit locomotion at a considerably lower energy cost than would be possible with a conventional valve-controlled configuration using pressure-compensated pumps. The report also discusses construction of the vehicle frame and cockpit and initial stages of testing of the vehicle prime mover system. The latter is composed of a 90 hp motorcycle engine modified for stationary operation, and an associated high energy density flywheel intended to smooth power demands on the engine imposed by large fluctuations in hydraulic pressure and flow in the leg control pumps.

Electrical system research reported includes validation of the operation of a breadboard version of the vehicle control computer. This computer consists of fifteen intel 86/30 single-board computer systems. A real-time simulation of the ASV-84 mechanical system is implemented on a PDP-11/70 computer to permit testing of the control computer hardware and software. Test results indicate that the control computer possesses adequate power for initial vehicle testing. A preliminary experimental evaluation of an optical radar system to be used to provide the control computer with a local terrain elevation map is also included. A complete listing of a computer program used for validation of algorithms for automatic terrain-following using such data is included in the report.

Semi-Annual Technical Report

for

DARPA Contract MDA903-82-K-0058

AN EXPERIMENTAL STUDY OF AN ULTRA-MOBILE VEHICLE FOR OFF-ROAD TRANSPORTATION

Prepared by

Robert B. McGhee and Kenneth J. Waldron

covering the period

October 1, 1983, through March 31, 1984

College of Engineering
The Ohio State University
Columbus, Ohio 43210

1. INTRODUCTION AND SUMMARY

Research under this contract began on October 1, 1981. As planned, the first two years of work were primarily devoted to design activities supported both by computer simulation studies and by experimental investigations with major subsystems. Several laboratory-scale vehicles were also constructed in this time period to permit validation of algorithms and evaluation of components and concepts. This work was all completed on-time and within budget. Details can be found in our Quarterly R & D Status Reports, and in our previous Semi-Annual Technical Reports. A listing of milestones actually achieved during our first two years of research is attached to our Quarterly R & D Status Report for the period October 1, 1983 through December 31, 1983.

By the beginning of the period covered by this report, conceptual design of the ASV-84 was essentially complete. Per the original project plan, dated May 13, 1981, the first half of FY 84, the period covered by this report, has been devoted to an intensive effort centered on subsystem detailed design and manufacturing, as well as on initial steps toward subsystem integration. The details of our accomplishments in these areas are included in this report and in previously submitted R & D Status Reports for the period. While this work has revealed no significant defects in the overall ASV-84 system design, some delays in manufacturing have been encountered for a variety of reasons. This matter is treated in detail in our letter to DARPA dated February 27, 1984. As a consequence of these factors, and subsequent delays reported in later correspondence, we no longer anticipate that significant testing of the ASV-84 vehicle can be completed under the current contract. Rather, we are concentrating our efforts on completing vehicle construction to the greatest extent possible during the current fiscal year.

Per recent meetings with DARPA, our goal is now to complete final assembly of the ASV-84 vehicle in calendar year 1984. A proposal detailing this activity as well as subsequent phases of vehicle testing, evaluation, and modification has been favorably reviewed by DARPA. As of the time of this writing, a contract covering this work is in the final stages of negotiation.

2. MECHANICAL SYSTEM RESEARCH

2.1 Cockpit Structure

The design and manufacture of the cockpit structure was completed during this reporting period. The structure is shown mated to the body frame in Figure 1. The cockpit frame has subsequently been delivered to the University of Wisconsin to be fitted with displays and controls. The frame dimensions used for the cockpit represent a compromise between the space demanded by an ergonomically ideal design and the necessity to minimize the size of the structure to avoid excessive overall weight and forward displacement of the center of mass of the vehicle relative to the legs. Specifically, the weight of the structure increases with fore-aft length at a rate faster than its second power because increasing the length also increases the maximum bending moment which can be applied to it by a given load. For operator protection purposes, it is necessary to design for a crash in which the vehicle descends from maximum height to a position in which it is supported only on the extreme front of the cockpit structure, and on the rear legs. In the event that the present cockpit dimensions prove to be so small as to cause operator inconvenience which would hinder the vehicle test program, a second cockpit will be designed as a retrofit. This is easily done because the cockpit has been designed as an easily separable module, both structurally and electronically.

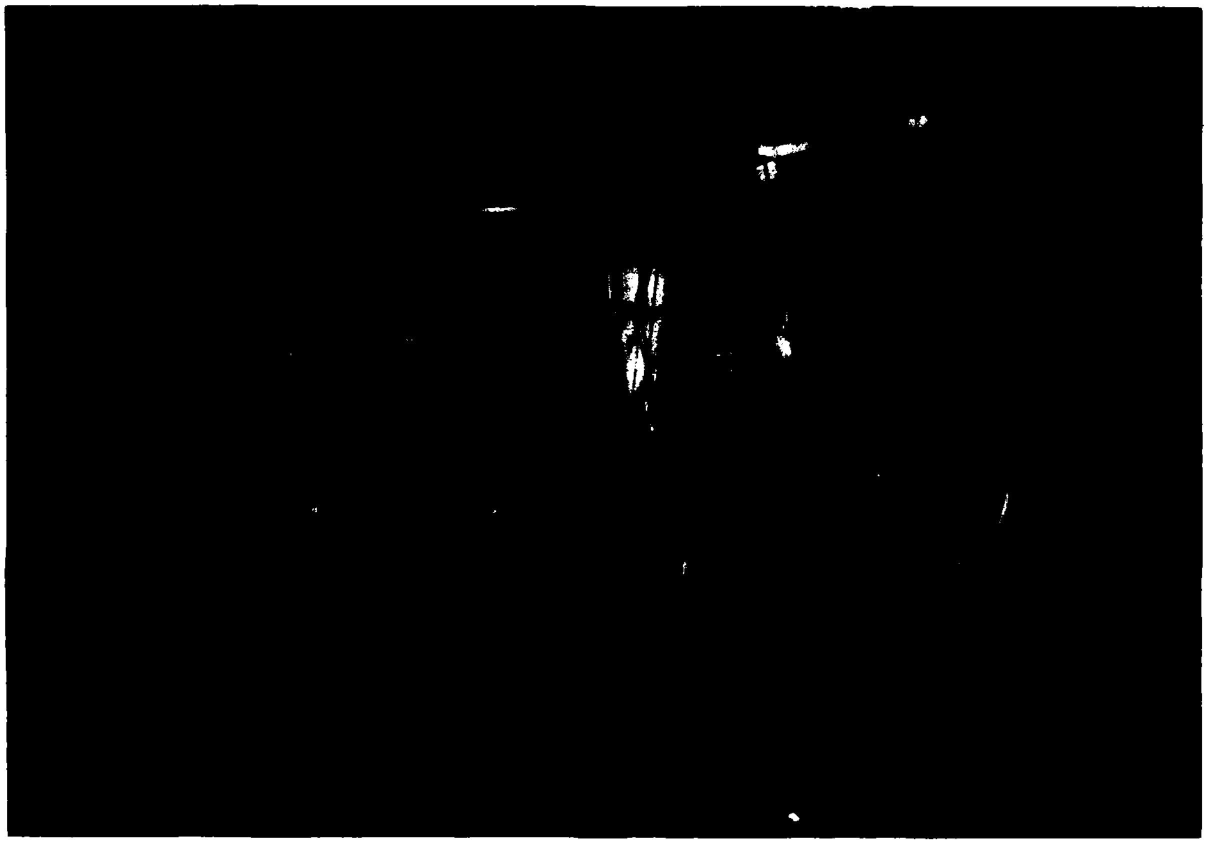


Figure 1. Completed Cab Structure Mated to Vehicle Frame

2.2 Engine Cooling

The engine cooling system has developed to the point that a continuous output of 70 hp is possible without overheating. Achieving this performance represented a substantial problem since, although the engine is rated at 90 hp peak, as a motorcycle engine it is only designed for a fraction of that as continuous output, and, moreover, it is designed to operate in a rapidly moving air stream. To provide for stationary operation, two cooling systems have been designed and tested, and will be installed on the vehicle engine. The first of these is a forced air system consisting of a fan and a cowling to distribute the air over the engine. The second is a water injection system. It was initially felt that sufficient cooling for continuous high-power output would not be attainable by using a forced air system alone. For this reason, the water injection system, based on aircraft engine technology, was designed. The water is injected at a metered rate into the intake manifold through four ports spaced to distribute it evenly among the four cylinders. Dynamometer tests have shown, however, that the fully-developed forced air system is capable of maintaining allowable head temperatures at continuous output levels in excess of 60 hp. Therefore, we expect to be able to operate using the forced air system alone in normal conditions. The water injection system is very compact and has also proved effective in dynamometer tests. We therefore plan to install it as a back-up cooling system which will be automatically activated if engine head temperature exceeds a pre-set value. Figure 2 shows the engine set up on a hydraulic dynamometer. The cowling can be seen. Also evident is an oil cooler which will be part of the installation. The large duct in the photograph was used to deliver air from a fan which will not be used on the vehicle. The fan to be installed will be adjacent to the engine.

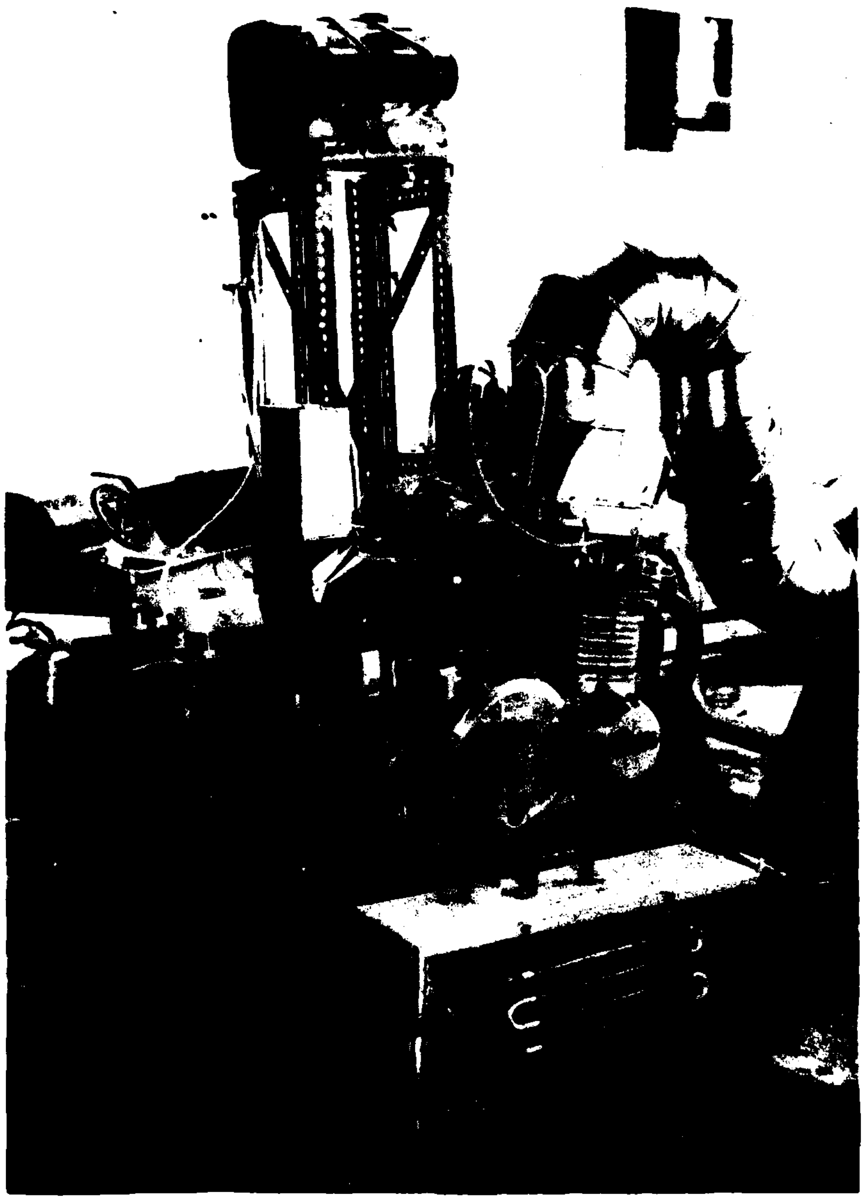


Figure 2. Engine and Cooling System Mounted on Dynamometer Test Stand

2.3 Prototype Leg Tests

The prototype leg was tested at speeds closely approximating those which will be used in vehicle operation. Control software was developed to move the foot along a desired trajectory in the return phase of the cycle, and to maintain a desired contact pressure during the contact phase. A videotape of these tests has previously been supplied to DARPA. The leg has been cycled at frequencies up to 1.4 Hz. Although the leg stroke used was shorter than will be used at comparable speeds on the vehicle, due to the limitations of the test stand, the tests closely simulated vehicle operating conditions in all other respects. In particular, the equivalent ground speed attained at 1.2 Hz. was approximately 5 mph, the designed cruise speed, and that attained at 1.4 Hz. was close to 8 mph, the designed maximum speed. Figure 3 shows a measured foot trajectory and curves of horizontal and vertical velocity versus time for a typical leg test. The curves actually show actuator displacements and rates with inches used as the unit of distance. Thus, the lift actuator displacement and rate must be multiplied by the pantograph factor of 4 to get vertical foot displacement and rate. Correspondingly, the drive data must be multiplied by 5 to get horizontal foot displacement and rate.

During the course of these tests, one of the hydraulic lines of the foot attitude maintenance system was damaged, leading to a progressive degradation in functioning of that system and, ultimately, to complete cessation of function. It was very evident in the tests that the foot attitude system is essential to high speed leg operation. Cyclic operation at frequencies above 0.6 Hz. without the foot attitude system led to large, uncontrolled swings of the foot about the ankle, and to failure of the foot to assume a sole flat position during contact. The problem with the attitude system was minor and will be avoided in the permanent installation.

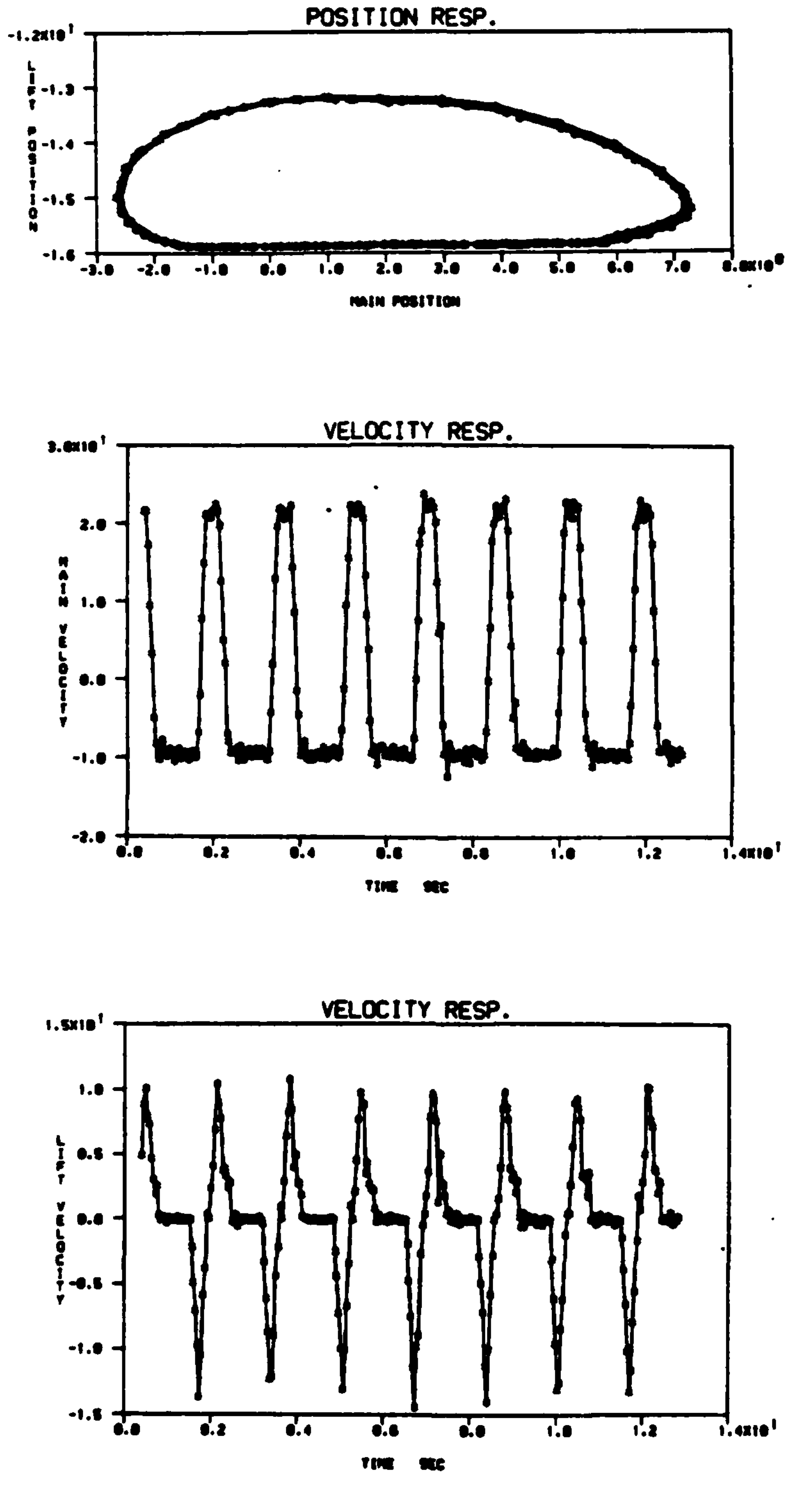


Figure 3. Position and Velocity Response of Prototype Leg Measured at Actuator

2.4 Leg Design and Fabrication

During this period the detailed design of the leg links, and of the joints which connect them was completed. Manufacture has commenced with the aluminum plate from which the links will be fabricated having been cut out. Welding of the leg links is in progress. The leg boxes, which carry the guideways for the linear bearings on which the leg links are mounted, and the mounts for the actuators and the lift and drive pumps, remain to be detailed.

2.5 Flywheel Tests

Personnel from OSU provided technical support to Owens Corning during flywheel spin tests conducted during this period. This was necessitated by a management decision at Owens Corning which withdrew support from these tests. Considerable difficulties were encountered due to the Owens Corning spin pit not being designed to test flywheels in their housings. It is necessary to test the University of Wisconsin flywheel design inside its housing in order to evaluate its evacuation system and to discover any vibrational modes which might arise from the bearing mount configuration. Problems also arose in the testing due to flooding of the case with oil, and due to the inadequate low speed torque of the air turbine used to power the rig. Vibrations, which limited the speed which could be achieved, suggested balancing, or structural resonance problems.

It should be noted that this work does not appear in the work statement of our contract. It was undertaken as work essential to ensure timely completion of our own tasks, since assembly and testing of the vehicle power package is dependent on successful completion of the flywheel test program.

3. ELECTRICAL SYSTEM RESEARCH

3.1 Precision Footing Control

Control of body motion with six degrees of freedom in precision footing mode has been implemented and tested on the breadboard ASV-84 vehicle computer. For these tests, the PDP-11/70 computer was used to simulate the vehicle kinematics including the cranked shank configuration of the prototype leg presented in our previous Semi-Annual Report, dated May, 1984. Body and leg motion were represented in real time during these tests by means of a vector graphics display system using a "wire-frame" vehicle model. A more easily understood shaded color graphics representation was subsequently produced by single-frame motion picture photography. A videotape copy of this motion picture has been provided to DARPA. Figure 4 presents a single frame from this videotape.

While studies of precision footing control of body motion can be accomplished only by simulation prior to completion of the entire ASV-84 vehicle, manual control of individual legs can be investigated using the physical hardware of the prototype leg. This has been done with entirely satisfactory results. Specifically, using a three-axis joystick as an input device, the prototype leg has been operated in Cartesian coordinates analogously to a precision manipulator. An accuracy of better than 1/4 inch in foot placement is easily achieved in this mode of operation. A videotape showing such testing has previously been furnished to DARPA.

3.2 Omnidirectional Vehicle Control

The ASV-84 vehicle mechanical design has been optimized to provide a high-speed forward-directed "dash" mode of locomotion. It therefore is of a relatively slender configuration in comparison to the previous laboratory-

Figure 4. Single Frame from Videotape Showing Precision Footing Control of Simulated ASV-84 Vehicle by Breadboard Computer

scale OSU Hexapod vehicle. This change complicates the problem of omnidirectional or "close maneuvering" control. However, during the present reporting period, an extensive simulation study of omnidirectional control using the ASV-84 vehicle geometry has been completed and documented in the form of a Ph.D. dissertation. This dissertation is attached as an appendix to this report. It is felt that the algorithms presented in this dissertation provide one entirely satisfactory approach to realization of close maneuvering for experimental evaluation of the ASV-84 vehicle. Alternative algorithms are currently under investigation.

The principal features of the algorithms presented in the appendix are as follows:

1. The operator may select gaits in which three, four, or five legs are on the ground at all times, depending upon the desired speed and on the average terrain slope and roughness.
2. If available, proximity sensors can be used to maintain a specified foot clearance above the terrain during the transfer phase of leg motion. Otherwise, a desired maximum foot lift relative to the body is set by the operator to a value judged sufficient for the terrain being negotiated.
3. The body attitude is adjusted so as to permit effective use of the limbs on uneven terrain without subjecting the operator to undue discomfort resulting from excessive body motion. In particular, the operator can elect to have the vehicle only <u>partially</u> adjust its pitch and roll motions to average terrain slope.
4. The algorithm provides for an automatic reduction of body ground-clearance for enhanced stability on steep slopes.

While continuing simulation studies and subsequent experiments with the completed ASV-84 vehicle will undoubtedly produce improvements to the above described algorithm, we feel that we now have established a satisfactory baseline for the conduct of such investigations. The operation of this algorithm has been documented in the form of a videotape of a real-time vector graphics display of simulated vehicle motion with joystick control of forward, lateral, and turning velocities. A copy of this videotape has been provided to DARPA. A photograph of one frame of this videotape is included in the attached Appendix 2.

3.3 Functional Test of Terrain Mapping System

In November, 1983, the terrain scanner developed by ERIM was delivered to Battelle Columbus Laboratories for interfacing to the ASV-84 guidance computer. Subsequently, this system was transferred to OSU for integration with the vehicle control computer. After some initial problems relating to electrical compatibility, the entire system was shown to function as intended except for the following difficulties noted with the scanner:

1. Due to excessive power consumption by the horizontal scanning mirror drive motor, the scanner overheats after approximately fifteen minutes of operation at 70°F ambient temperature. This condition leads to automatic shutdown through the action of an integral thermal protection relay.
2. The main power relay sticks open. It has been replaced, and the problem recurred.
3. The nodding mirror requires more current than anticipated. On advice from ERIM, its fuse has been replaced with one of larger size.
4. The nodding mirror frequently fails to initialize correctly when power is turned on. When this occurs, the subsequent scan cycle is also incorrect.

5. The scanner sometimes produces incorrect range data due to reflections from its protective window occurring near the middle of the azimuth and elevation scan angles.
6. Frame scanning seems to begin prematurely, before the nodding mirror reaches the bottom of its travel.
7. The scanner sends its first line of data in each frame faster than subsequent lines.
8. In a laboratory setting, we have found that the range measured by the scanner is influenced by the type of surface material associated with the objects scanned.

All of the above problems have been communicated to ERIM. It is our understanding that funding for a retrofit to correct these difficulties is included in a proposal which has been submitted by ERIM to DARPA. At the present time, however, it seems unlikely that the deficiences noted can be rectified within the time frame of our current contract. We therefore anticipate that no ASV-84 system or subsystem tests requiring an operational scanner will be undertaken in FY 84.

Appendix 1

List of Publications

9/12/84

RESEARCH PUBLICATIONS SUPPORTED BY CONTRACT MDA903-82-K-0058

College of Engineering
The Ohio State University
Columbus, Ohio 43210

The following theses, dissertations, and papers have been produced with the support of DARPA Contract MDA903-82-K-0058 since its inception on October 1, 1981. Copies of most of these documents are available upon request.

1. Waldron, K.J., Song, S.M., Vohnout, V.J. and Kinzel, G.L., "Computer-Aided Design of a Leg for an Energy-Efficient Walking Machine," Proceedings of the 7th Applied Mechanisms Conference, Kansas City, December 7-9, 1981, pp. VIII-1 to VIII-8.

2. Vohnout, V.J., Mechanical Design of an Energy-Efficient Robotic Leg for Use on a Multi-Legged Walking Vehicle, M.S. thesis, The Ohio State University, June, 1982.

3. Orin, D.E., "Supervisory Control of a Multilegged Robot," International Journal of Robotics Research, Vol. 1, No. 1, Spring 1982, pp. 79-91.

4. Waldron, K.J., Frank, A.A., and Srinivasan, K., "The Use of Mechanical Energy Storage in an Unconventional, Rough-Terrain Vehicle," 17th Intersociety Energy Conversion Engineering Conference, Los Angeles, California, August 8-13, 1982.

5. Brown, F.T., Dynamic Study of a Four-Bar Linkage Walking Machine Leg, M.S. thesis, The Ohio State University, August, 1982.

6. Ju, J.T., Safety Checking System with Voice Response for the OSU Hexapod, M.S. thesis, The Ohio State University, August, 1982.

7. Pugh, D.R., An Autopilot for a Terrain-Adaptive Hexapod Vehicle, M.S. thesis, The Ohio State University, August, 1982.

8. Tsai, C.K., Computer Control Design of an Energy-Efficient Leg, M.S. thesis, The Ohio State University, August, 1982.

9. Srinivasan, K., Waldron, K.J., and Dworak, J.A., "The Design and Evaluation of a Hydraulic Actuation System for a Legged Rough-Terrain Vehicle," presented at ASME Winter Annual Meeting, Phoenix, November, 14-16, 1982, Robotics Research and Advanced Applications, ASME, 1982.

10. Kao, M.L., A Reliable Multi-Microcomputer System for Real Time Control, M.S. thesis, The Ohio State University, December, 1982.

11. Chang, T.W., Motion Planning for Locomotion of a Multilegged Robot over Uneven Terrain, M.S. thesis, The Ohio State University, December, 1982.

12. Cheng, F.T., Computer Simulation of the Dynamics and Control of an Energy-Efficient Robot Leg, M.S. thesis, The Ohio State University, December, 1982.

13. Chung, T.S., Kinematic Simulation of an Adaptive Suspension Vehicle, M.S. thesis, The Ohio State University, December, 1982.

14. Barrientos, C.E., Development of a Multiple Microprocessor-Based Control System for Legged Locomotion Processing Using Interactive Design Tools, M.S. thesis, The Ohio State University, December, 1982.

15. Chuang, J.Y., Simulation of Load-Time History of an Adaptive Walking Vehicle by the Use of Rate Regulation, M.S. thesis, December, 1982.

16. Tsai, S.J., An Experimental Study of a Binocular Vision System for Rough Terrain Locomotion of a Hexapod Walking Robot, Ph.D. dissertation, May, 1983.

17. Gardner, J.F., Modeling and Control of the Hydraulic Drive System for a Walking Machine Leg. M.S. thesis, The Ohio State University, June, 1983.

18. Broerman, K.R., Development of a Proximity Sensor System for Foot Altitude Control of a Terrain-Adaptive Hexapod Robot, M.S. thesis, The Ohio State University, August, 1983.

19. Klein, C.A., Olson, K.W., and Pugh, D.R., "Use of Force and Attitude Sensors for Locomotion of a Legged Vehicle over Irregular Terrain," International Journal of Robotics Research, Vol. 2, No. 2; Summer, 1983, pp. 3-17.

20. Ozguner, F., Tsai, S.J., and McGhee, R.B., "Rough Terrain Locomotion by a Hexapod Robot Using a Binocular Ranging System," Proc. of First International Symposium on Robotics Research, Bretton Woods, N.H., September, 1983.

21. Wang, S.L., The Study of a Hexapod Walking Vehicle's Maneuverability Over Level Ground and Obstacles and Its Computer Simulation, M.S. thesis, The Ohio State University, September, 1983.

22. Song, S.M., Waldron, K.J., and Kinzel, G.L., "Computer-Aided Geometric Design of Legs for a Walking Vehicle," Proceedings of 8th Applied Mechanisms Conference, St. Louis, Missouri, September 19-21, 1983, pp. 70-1 to 70-7.

23. Vohnout, V.J., Alexander, K.S., and Kinzel, G.L., "The Structural Design of the Legs for a Walking Vehicle," Proceedings of 8th Applied Mechanisms Conference, St. Louis, September, 1983, pp. 50-1 to 50-8.

24. Gardner, J.F., Dworak, J.A., Srinivasan, K. and Waldron, K.J., "Design and Testing of a Digitally Controlled Hydraulic Actuation System for a Walking Vehicle Leg Mechanism," Proceedings of 8th Applied Mechanisms Conference, St. Louis, September, 1983, pp. 2-1 to 2-7.

25. Orin, D.E., Tsai, C.K., and Cheng, F.T., "Dynamic Computer Control of a Robot Leg," Proceedings of IECON '83, San Francisco, California, November, 1983.

26. Alexander, K.E.S., The Structural Design and Optimization of a Walking Machine Frame and Leg, M.S. thesis, The Ohio State University, December, 1983.

27. Lee, W.J., A Computer Simulation Study of Omnidirectional Supervisory Control for Rough-Terrain Locomotion by a Multilegged Robot Vehicle, Ph.D. dissertion, March, 1984.

28. Ozguner, F. and Kao, M.L., "A Multimicroprocessor System for Fault Tolerant Control of an Articulated Mechanism," IEEE Trans. on Industrial Electronics, May, 1984.

29. McGhee, R.B., Orin, D.E., Pugh, D.R. and Patterson, M.R., "A Hierarchically-Structured System for Computer Control of a Hexapod Walking Machine," Proc. of Symposium on Theory and Practice of Robots and Manipulators, Udine, Italy, June, 1984.

30. Waldron, K.J., Song, S.M., Wang, S.L. and Vohnout, V.J., "Mechanical and Geometric Design of the Adaptaive Suspension Vehicle," Proc. of Symposium on Theory and Practice of Robots and Manipulators, Udine, Italy, June, 1984.

31. Waldron, K.J., Vohnout, V.J., Pery, A. and McGhee, R.B., "Configuration Design of the Adaptaive Suspension Vehicle," International Journal of Robotics Research, Vol. 3, No. 2, Summer, 1984.

32. Ozguner, F., Tsai, S.J., and McGhee, R.B., "An Approach to the Use of Terrain Preview Information in Rough-Terrain Locomotion by a Hexapod Walking Machine," International Journal of Robotics Research, Vol. 3, No. 2, Summer, 1984.

33. Pearson, K.G., and Franklin, R., "Characteristics of Leg Movements and Patterns of Coordination in Locusts Walking on Rough Terrain," International Journal of Robotics Research, Vol. 3, No. 2, Summer, 1984.

34. Kwak, S.H., A Simulation Study of Free-Gait Algorithms for Omni-Directional Control of Hexapod Walking Machines, M.S. thesis, The Ohio State University, April, 1984.

35. Schiebel, E.N., Design of a Mechanical Proximity Sensor for a Six-Legged Walking Machine, M.S. thesis, The Ohio State University, July, 1984.

36. Song, S.M., Kinematic Optimal Design of a Six-Legged Walking Machine, Ph.D. dissertation, The Ohio State University, July, 1984.

37. Lee, S.P., An Experimental Study of Alternative Schemes for Asynchronous Message Passing in a Real-Time Multicomputer Control System, Ph.D. dissertation, The Ohio State University, August, 1984.

38. Barrietos, C. and Klein, C.A., "Design of a Multimicroprocessor-based Controller Using a Structured Design Approach," to appear in IEEE Trans. on Industrial Electronics, December, 1984.

39. Orin, D.E., Tsai, C.K. and Cheng, F.T., "Dynamic Computer Control of a Robot Leg," to appear in IEEE Trans. on Industrial Electronics, February, 1985.

40. Ozguner, F. and Tsai, S.J., "Design and Implementation of a Binocular Vision System for Locating Footholds for a Multilegged Walking Robot," to appear in IEEE Trans. on Industrial Electronics, February, 1985.

41. McGhee, R.B., "Vehicular Legged Locomotion," Advances in Automation and Robotics, ed. by G.N. Saridis, Jai Press, Inc., Greenwich, Conn., 1985.

Appendix 2

A Computer Simulation Study of Omnidirectional Supervisory Control for Rough-Terrain Locomotion by a Multilegged Robot Vehicle

A COMPUTER SIMULATION STUDY OF OMNIDIRECTIONAL SUPERVISORY CONTROL FOR ROUGH-TERRAIN LOCOMOTION BY A MULTILEGGED ROBOT VEHICLE

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

by

Wha-Joon Lee, B.S.E.E., M.S.E.E.

* * * * * *

The Ohio State University

1984

Reading Committee:

Professor C. A. Klein

Professor R. B. McGhee

Professor D. E. Orin

Approved by

[illegible]

Advisor
Department of Electrical Engineering

to my wife

ACKNOWLEDGMENTS

I would like to acknowledge the encouragement and advice that I received from my advisor, Professor David E. Orin, throughout the course of this work and especially to thank him for spending his time in correcting this manuscript. I would also like to express my appreciation to Professor Robert B. McGhee, my initial advisor, who always supported and encouraged me throughout this research work and my study. I am grateful to Professor Charles A. Klein for his careful review of this work.

Thanks are due to Mr. Sehung Kwak, with whom I spent a great deal of time discussing control algorithms. I would also like to thank Ms. Margaret Starbuck for her assistance in typing and copying this manuscript.

Finally, I wish especially to thank my parents and all members of my family for their constant support and encouragement in my studies. I am most grateful to my lovely wife, Hae-Kyung, for her understanding and patience in spite of her long illness.

This work was supported by the Defense Advanced Research Projects Agency under Contracts MDA903-81-C-0138 and MDA903-82-K-0058.

VITA

March 29, 1949	Born - Sungjin, Korea
February, 1973	B.S.E.E., Electronics, Seoul National University, Seoul, Korea
December, 1979	M.S., Electrical Engineering, The University of Texas, Austin, Austin, Texas
1981 - 1984	Graduate Research Associate, Digital Systems Laboratory, The Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in Computer Engineering: Professors C.A. Klein, R.B. McGhee, K.W. Olson, D.E. Orin

Studies in Control Engineering: Professors R.E. Fenton, H. Hemami, U. Ozguner

Studies in Computer and Information Science: Professor F.C. Crow

Studies in Communication Engineering: Professor R.T. Compton

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

Chapter 1
INTRODUCTION

1.1 General Background

For more than a decade, legged robot vehicles have been designed and constructed in several institutions throughout the world for a wide range of potential applications, generally in environments too hazardous for humans or in those inaccessible to conventional wheeled or tracked vehicles [1-6]. Some of the possible applications for legged vehicles are in the areas of land and under-water exploration, fire fighting, harvesting trees in the logging industry, underground mining, and hazardous tasks in nuclear power plants [5]. The major design objective of legged robot vehicles is to achieve superior mobility/walking capability, utilizing greater adaptability to terrain irregularities due to the fundamentally different nature of their interaction with the supporting terrain, in comparison to wheeled or tracked vehicles [7].

However, the advantages of legged robot vehicles over other types of off-road vehicles result from their large number of controllable degrees-of-freedom which requires highly efficient joint coordination. Controlling such a large number of degrees-of-freedom is a difficult task for a human operator as demonstrated in [1]. This problem has led to the development of the computer-controlled legged robot vehicle, which is made possible by recent advances in computer technology.

Up to now, several investigations directed toward computer control of legged vehicles have been reported in the literature [4-9]. The control algorithms developed to date for locomotion of legged robot vehicles have most often been based on supervisory control in which the human operator provides direction and speed commands for the body, and a computer controls limb cycling in such a way so as to achieve the desired motion while maintaining stability [4,8].

Prior to this dissertation work, researchers at The Ohio State University have also investigated the problem of computer control of legged vehicles. As a result, basic control algorithms have been developed to generate appropriate leg stepping sequences and to coordinate body and leg motions for the OSU Hexapod which was constructed for laboratory experimentation of locomotion [3,8,10]. Much of this work was concerned with level body locomotion over even terrain which is assumed relatively smooth. The major deficiency of the algorithms developed then is their limited ability to execute complex maneuvers required to traverse rough terrain and their lack of body attitude and altitude control to accommodate terrain variations. While basic solutions to the latter problem have been made for the OSU Hexapod using force sensors, a vertical gyroscope, and pendulums [11-13], the algorithms developed still need improvement and modification, which motivates this dissertation work.

Another problem with the previous algorithms developed for the OSU Hexapod is that they do not allow sufficient freedom of body motion. Three modes of control have been defined--Cruise for forward/backward motion, Side-stepping for purely lateral movement, and Turn-in-Place in which the body is rotated about its center of gravity with no lateral or forward/backward movement. The body velocity component values are obtained from a human operator through a three-axis joystick. However, restrictions on these components are made in each of the control modes so that legs will be maintained within their kinematic limits and vehicle stability is ensured. It is desirable to remove the restrictions placed on body velocity and to integrate the three modes of operation to obtain omnidirectional control, and this also motivates this dissertation work.

The objective of this work, then, is to develop control algorithms to combine the three different modes of operation defined for the OSU Hexapod [8] to provide for omnidirectional control and to extend and modify the previous motion planning algorithms to accomplish greater adaptability to uneven terrain. The algorithms are developed through the use of computer simulation on the PDP-11/70 minicomputer. A graphics display device (HP 1350A) is used extensively for motion display and various terrain conditions may be generated interactively. The result is that an approach to computer-aided design of motion planning algorithms has been developed. Since the algorithms developed in this work will be applied to a hexapod robot vehicle, named the Adaptive Suspension Vehicle (ASV), which is currently under construction at The Ohio State University, the vehicle model for the simulation is chosen from it.

The major problems dealt with in this simulation work may be partitioned into the following three areas:

1) In order to investigate locomotion over uneven terrain, a definition of the terrain model over which the vehicle moves is required. The terrain model should be relatively simple to generate but, on the other hand, should present somewhat realistic terrain conditions to vehicle motion.

2) In order to increase maneuverability of the vehicle, the limitations imposed on the body motion trajectory by the previous control algorithms should be removed. In particular, omnidirectional control through a three-axis joystick should be provided. At the same time, stability of the vehicle should be maintained with proper leg stepping sequences.

3) The final problem investigated involves the development of control algorithms for the Close Maneuvering mode of operation for the Adaptive Suspension Vehicle (ASV) [14]. This mode of operation requires proximity and contact sensors for local terrain sensing at each foot. In order to accommodate terrain irregularities, the body attitude and altitude should be appropriately controlled. However, this body regulation tends to reduce stability of the vehicle over sloped terrain. Therefore, approaches to increase stability should be investigated.

1.2 Organization

A brief review of the previous research on legged vehicle locomotion directed to develop control algorithms is given in Chapter 2. This chapter includes previous work on gaits, a discussion of the operational modes for the ASV, supervisory control schemes adopted for the OSU Hexapod, and feedback control strategies presently employed for low-level servo control.

In Chapter 3, terrain and vehicle models used in the simulation are presented in detail. Also, the computer-aided design concept for the development of control algorithms is described.

Chapter 4 details an algorithm to integrate the three joystick control modes previously defined for the OSU Hexapod [8] in order to provide omnidirectional control. Periodic gaits are used and much of the chapter presents a new method for choosing an optimal cycle period. The concept of the constrained working volume is also presented.

Chapter 5 describes algorithms that implement the Close Maneuvering mode of operation for the ASV for uneven terrain. Details of the body and leg motion planning algorithms are presented along with discussion concerning the dynamic changing of the size of the constrained working volume to increase the stability.

Chapter 6 covers the overall software structure of the control program and the graphics routines along with the data base for the terrain and vehicle models. Finally, different evaluations of the control algorithms developed in this work are discussed.

Finally, Chapter 7 summarizes the contributions made in this dissertation work with a discussion of possible further research. An appendix is included which contains a listing of the control programs which were written in PASCAL on the PDP-11/70 minicomputer.

Chapter 2

REVIEW OF EXISTING CONTROL ALGORITHMS

2.1 Introduction

This chapter attempts to give a brief overview of previous research directed to develop the control algorithms for legged vehicle locomotion. A major thrust of much of the early work was to provide a theoretical formalization of gaits of human beings and animals, and this has been well established in the literature [15-17] and will be reviewed here. On the other hand, control algorithms to implement the gaits in legged vehicle locomotion are still under development and the latest work will also be outlined.

Over the past two decades, a number of walking machines have been constructed to implement gaits and to develop control algorithms in the laboratory environment with coordination of multiple legs. Also, some special machines which utilize legged locomotion principles were commercially built. Typical examples of the former are the G.E. Quadruped Transporter [1], 'Phoney Pony' [2], the OSU Hexapod [3], and the Moscow State University Hexapod [4]. For the latter case, there are the Menzi Muck Climbing Hoe [18], the Kaiser Spyder [19] and ODEX 1 [5].

Through experimentation with the above walking machines, several approaches to control of legged locomotion systems have been explored. These include computer control of the joints, supervisory control for the man-machine interface, and feedback control at the servomechanism level to provide energy efficiency, load balancing and active compliance.

All of the above control concepts have been applied to the control of the OSU Hexapod in both software and hardware implementation. It was built in 1976 at The Ohio State University in order to allow laboratory experimentation of legged vehicle locomotion. This vehicle is electrically powered and each leg possesses three degrees of freedom with three revolute joints. An insect-type of leg kinematics is employed for the leg structure. The control algorithms developed for the OSU Hexapod are well documented in [11].

Based on experience with the OSU Hexapod, another six-legged vehicle, called the Adaptive Suspension Vehicle (ASV), is currently under construction at The Ohio State University. The ASV will be roughly the same size as the G.E. Quadruped and hydraulically powered. The leg structure will be based on a mammalian configuration with pantograph coordination of joint motions [20] in order to decouple actuator loads. The present model for the ASV is used in the simulation study in this dissertation work.

In Section 2.2, some theoretical background material for gaits is presented and provides a basis for much of the dissertation work. Several operational modes for the ASV have been proposed by McGhee et al. [14], depending on the terrain conditions, fuel economy, maneuverability of the vehicle or vehicle sensors used, and these are described in Section 2.3. In Section 2.4, two different motion planning control schemes, based on the supervisory control concept, are discussed. Finally in Section 2.5, two of the feedback control schemes developed at the servomechanism level for walking machines are briefly reviewed.

2.2 Gaits

Due to the discrete aspects of legged locomotion, the natural and theoretically possible patterns of leg-lifting and leg-placing events, called gaits, used by human beings and animals have been investigated since the first classification of gaits by Muybridge [15,16]. In the 1960's, Hildebrand [17] formulated the definition of a gait and developed a more quantitative approach to the classification of symmetrical gaits.

Later, McGhee [21] established a general mathematical theory of legged locomotion from the point of view of finite state machine theory by regarding each leg as a two-state (1 and 0) sequential machine. McGhee then introduced a gait matrix, G, which is formally defined as a k-column matrix whose successive rows are binary k-tuples corresponding to the the successive states of a particular gait of k-legged locomotion. A zero entry in column j indicates that leg j is in contact with the ground (support phase) while a 1 entry means that it is not (transfer phase).

McGhee and Jain [22] introduced another mathematical description of a gait, the "event sequence". If the legs of the vehicle are numbered, 1,2,...,k, then the event of placing leg i is denoted event i, while lifting of leg i is arbitrarily denoted event i+k. The event sequence is further employed in this section to describe gaits.

2.2.1 Stability

During legged vehicle locomotion, it is essential that the vehicle should not fall over. This problem led to stability studies on legged vehicle locomotion by many researchers.

The stability of a vehicle may be measured either statically or dynamically during locomotion. The static stability concept is employed in this work, since it provides a simpler basis for the control. The test of static stability involves determining whether the vertical projection of the center of gravity of the vehicle onto the supporting surface is contained within a polygon whose vertices are defined by the feet in contact with the ground. This notion was formally defined in [23] through the following:

Definition 1: The <u>support pattern</u> associated with a given support state [7] is the convex hull (minimum area convex polygon) of the point set in a horizontal plane which contains the vertical projections of the feet of all supporting legs.

There can, however, exist a large number of statically stable gaits due to the combinatorial aspects of the gait selection problem. In order to choose optimal gaits in the sense of static stability, the following measurement of gait stability is defined [23].

Definition 2: The <u>longitudinal stability margin</u> is the shortest distance over an entire cycle of locomotion from the projection of the vehicle center of gravity onto the support plane to the edge of the support pattern as measured in the direction of travel.

The longitudinal stability margin is defined only for straight-line locomotion. When motions of the vehicle are arbitrary, the above definition must be expanded. Thus, the stability margin may be defined for general periodic gaits to include the case of arbitrary locomotion.

Definition 3: The <u>stability margin</u> is the shortest distance over an entire cycle of locomotion from the projection of the vehicle center of gravity onto the horizontal plane to the edge of the support pattern.

For hexapod vehicles, Bessonov and Umnov [24] found a class of optimally stable gaits, called wave gaits, for straight-line locomotion by using the minimax longitudinal stability criterion given in Definition 2. Optimal wave gaits have the property that the stepping events on each side of the vehicle move from the rear to the front of the body for forward motion (forward wave gaits), while they progress from the front to the rear of the body for backward motion (backward wave gaits). The terms "forward" and "backward" are designated with respect to the head of the vehicle (fixed). If there is no distinction between the head and tail of the vehicle, the forward and backward wave gaits represent the same leg sequences with respect to the direction of travel and are both optimal. In other words, the optimal gaits are symmetric.

2.2.2 Periodic Gaits

If every limb of a legged vehicle or animal operates with the same cycle time, then the gait is said to be periodic [25]. The following paragraphs briefly discuss periodic gaits and also define two particular classes of periodic gaits used in artificial legged locomotion systems: optimal wave gaits and Follow-The-Leader gaits [26].

The following definitions for straight-line motion were introduced by McGhee and Frank [23].

Definition 4: The period, τ, is the time required for one complete locomotion cycle of the gait.

Definition 5: The stride length, λ, of a gait is the distance by which the center of gravity of the locomotion system is translated during one complete locomotion cycle.

Definition 6: The duty factor, β, is the fraction of a locomotion cycle that each leg spends in contact with the supporting surface.

Definition 7: The relative phase, ϕ_i, is the fraction of a locomotion cycle by which the contact of leg i with the supporting surface lags behind the contact of leg 1 (left front leg).

Bessonov and Umnov [24] have shown that optimal wave gaits for a hexapod may be described through the following equations:

$$\phi_3 = \beta \qquad \phi_5 = 2\beta - 1 \qquad 1 > \beta > 0.5 . \qquad (2.1)$$

Note that legs on the left side of a hexapod be numbered from front to rear as 1,3,5 successively and those on the right side in the same order as 2,4,6. Since these gaits are symmetric [17,21], the relative phases of the legs of any right-left pair are exactly half a cycle out of phase with each other. The event sequence diagram for optimal wave gaits with $\beta = 5/6$ is shown in Figure 2.1.

In addition to the above definitions, Orin [8] provided the following definition for implementation of periodic gaits.

Definition 8: The kinematic cycle phase, ϕ, is the distance by which the center of gravity of the locomotion system has translated since the last placement of leg 1, normalized to the stride length.

The relation between the kinematic cycle phase, ϕ, and the period, τ, is given in [8]:

$$\phi = \int \frac{1}{\tau} dt. \qquad (2.2)$$

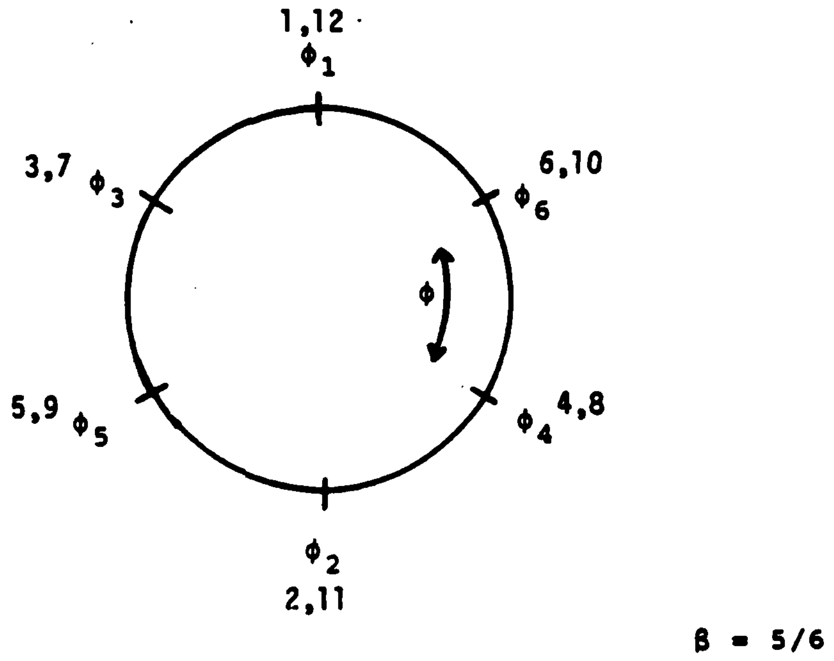


Figure 2.1. Event Sequence Diagram for an Optimal Wave Gait with Duty Factor β=5/6 Showing the Placing (Event i) and the Lifting (Event i+6) of Leg i.

By differentiating both sides of Eq. (2.2), with respect to time, the following is obtained:

$$\frac{d\phi}{dt} = \frac{1}{\tau} \tag{2.3}$$

or

$$\tau = \frac{dt}{d\phi}. \tag{2.4}$$

Eq. (2.4) may be approximated by replacing dt with Δt and $d\phi$ with $\Delta\phi$, respectively, such that

$$\tau \cong \frac{\Delta t}{\Delta\phi}. \tag{2.5}$$

Eq. (2.5) provides an approximate means to compute the period and will be discussed further in Chapter 4. Note that once the period, τ, is defined, the value of ϕ is either increasing or decreasing according to the sign of τ in Eq. (2.3).

The leg placing and lifting times may be determined by the use of the kinematic cycle phase, ϕ, and the relative phase, ϕ_i. In Figure 2.1, ϕ moves clockwise along the circle, when increasing and moves counterclockwise, when decreasing. When ϕ moves clockwise, the leg placing sequence for one locomotion cycle is leg 6, 4, 2, 5, 3, and 1, which implements the forward wave gait. On the other hand, when ϕ moves counterclockwise, the leg placing sequence is leg 1, 3, 5, 2, 4, and 6, which implements the backward wave gait.

From the above discussion, the relationship between ϕ and ϕ_i determines the state or phase for leg i: transfer phase or support phase. Let the leg phase variable of leg i, ϕ_{Li}, be defined as

$$\phi_{Li} = [\phi - \phi_i]\text{mod } 1\cdot \tag{2.6}$$

Then,

$$0 < \phi_{Li} < \beta \qquad \text{support phase} \qquad (2.7)$$

$$\beta < \phi_{Li} < 1 \qquad \text{transfer phase.} \qquad (2.8)$$

During the transfer phase of leg i, ϕ_{Li} changes from β for lift-off to 1 for touchdown for increasing ϕ_{Li} and vice versa for decreasing values. Accordingly, a new phase variable, ϕ_{Ti}, for the transfer leg as normalized to the total transfer time may be computed as follows:

$$\phi_{Ti} = \begin{cases} \dfrac{\phi_{Li} - \beta}{1 - \beta} & \text{for } \tau > 0 \\[2ex] \dfrac{1 - \phi_{Li}}{1 - \beta} & \text{for } \tau < 0. \end{cases} \qquad (2.9)$$

Thus, when the leg lifts off, $\phi_{Ti} = 0$, and when the leg touches down, $\phi_{Ti} = 1$.

In a similar manner, ϕ_{Li} changes from 0 for touchdown to β for lift-off during the support phase when increasing and vice versa when decreasing. Another phase varible, ϕ_{Si}, for the support leg as normalized to the total support time may be obtained as follows:

$$\phi_{Si} = \begin{cases} \dfrac{\phi_{Li}}{\beta} & \text{for } \tau > 0 \\[2ex] \dfrac{\beta - \phi_{Li}}{\beta} & \text{for } \tau < 0 \end{cases} \qquad (2.10)$$

so that ϕ_{Si} always changes from 0 for touchdown to 1 for lift-off.

The graphical representation of the above phase variables is shown in Figure 2.2. While the example used in the above discussion was for

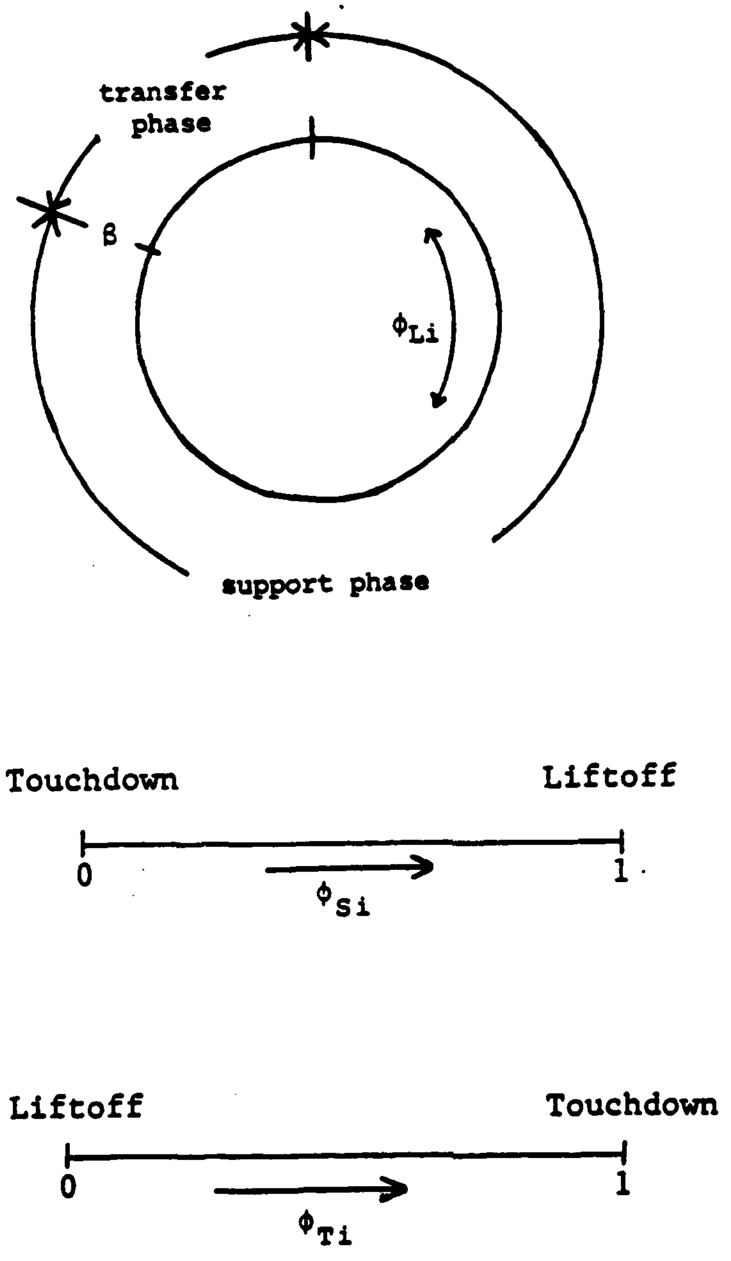


Figure 2.2. Graphical Representation of Phase Variables.

a wave gait, other types of periodic gaits may be easily implemented by altering the relative phase relationships of the legs.

Recently, another class of periodic gaits, called Follow-The-Leader (FTL) gaits, have been investigated [26]. In FTL gaits, the middle and rear legs use the same footholds as are used by the front leg on the same side of the body. For example, the placing sequence of one locomotion cycle, 1-3-5-2-4-6, possesses the FTL property. This sequence is the same as that for the backward wave gait as mentioned before. However, the stability of a FTL gait is not optimal, because the vehicle always moves forward in the FTL mode. In this sense, FTL gaits and optimal wave gaits are two different families of gaits whose intersection is empty [26].

2.2.3 Free Gaits

As discussed in the previous section, periodic gaits are totally governed by the fixed relative phase relationships of the legs. In other words, once the relative phases are defined, the leg lifting and placing events are fixed for one cycle of locomotion regardless of the terrain conditions. This results in control algorithms which are relatively simple to implement. Also, for locomotion over level terrain, periodic gaits may be efficient and optimal.

However, as the degree of terrain irregularities increases, periodic gaits may lose such efficiency and optimality. Since free gaits have no fixed relative phase relationships among the legs, there exists an infinite number of possible gaits. This allows the free gait control algorithms to choose an optimal gait for locomotion over rough terrain with great flexibility. Implementation of the control algorithms, however, may be more complicated than that for periodic gaits.

The complete theoretical formalization for the free gaits has not yet been established, and the algorithms developed are rather heuristic as compared to those for periodic gaits. Kugushev and Jaroshevskij [27] have suggested that the mathematical approach to periodic gaits for straight-line locomotion may be extended to include the nonperiodic gaits, i.e., free gaits. In [27], the problem for free gaits is partially formalized in such a way that a trajectory is specified in advance for the motion of the center of gravity of a legged system over a given terrain containing certain regions which are unsuitable for support.

Later, McGhee and Iswandhi [7] extended the work of Kugushev and Jaroshevskij to complete the formalization of this problem and developed a heuristic algorithm. Specifically, in [7], the terrain was divided into discrete cells, each about the size of a footprint. Each cell was designated as <u>permitted</u> (suitable for weight bearing), or <u>forbidden</u> to simulate the terrain irregularities. The control algorithm develpoed in this work was based on the <u>kinematic margin</u> of a foothold which relates to how long a foot can be on a given cell before the leg reaches its kinematic limit [7]. If the vehicle remains stable, legs are lifted so as to maximize the minimum value of the kinematic margin over all supporting legs. Otherwise, the algorithm corrects the unstable situation by placing a leg in the air when this is possible. Whenever a leg is placed, among those legs which make the vehicle stable, the one with the largest kinematic margin is utilized. Thus, the basic criteria for leg sequencing in this algorithm are the kinematic margin of each leg and the stability margin in the support pattern.

The above algorithm was further refined by Kwak [28], who employed the same criteria except that the kinematic margin is measured in time rather than in distance. In his work, improvements to the basic algorithms suggested by McGhee and Iswandhi [7] were implemented in software on the PDP-11/70 minicomputer using the HP 1350A graphics device for display of the vehicle and the terrain cells.

Recently, Patterson [29] proposed the control algorithm for free gaits in coorporation with the guidance algorithm using the terrain scanner which provides a terrain preview of average terrain slope using ultrasonic or laser signal. In this algorithm, the desired accelerations for the body motion of the vehicle are generated based on the operator's velocity commands and the current vehicle velocity. If the desired accelerations cause that the motion of the vehicle is unstable or the desired foothold of the leg is out of its kinematic limit, then the <u>deceleration plan</u> [29] is invoked to slow down the vehicle speed or to allow the vehicle to be brought to a halt while maintaining stability.

2.3 Operational Modes of the ASV

Control of vehicle locomotion strongly depends on both the terrain conditions for the desired motion of the vehicle and the sensors used. The sensors used for vehicle locomotion are force sensors, proximity sensors, the terrain scanner, etc. Especially, the use of the terrain scanner affects the operational modes over rough terrain. Presently, five operational modes [14] are considered to be implemented in control of the ASV--Cruise Mode, Terrain Following Mode, Dash Mode, Close Maneuvering Mode, and Precision Footing Mode.

The Cruise Mode is for locomotion over relatively smooth terrain with minimization of fuel consumption. In this mode, the body altitude and attitude of the vehicle could be maintained at the desired values by the computer using terrain and inertial sensors. The body crab angle, which is defined as an angle between the heading of the vehicle and the instantaneous body velocity vector, could be limited to a relatively small value and minimum turning radius could be of the order of several body lengths. That reduces manueverability somewhat, but the optimum speed of the vehicle can be achieved in the sense of fuel economy.

In order to increase maneuverability for the locomotion over rough terrain, the limitations imposed on the Cruise Mode may be removed. The Terrain Following Mode is essentially for forward locomotion over rough terrain with a slower speed than that in the cruise mode and moderate fuel consumption. In this mode, all the vehicle sensors can be used such as the terrain scanner, proximity sensors, force sensors, etc. The terrain scanner can provide a terrain preview to predict foothold locations and terrain slope information for body attitude and altitude regulation. The proximity sensors are used for local control of foot positions, while the force sensors can be used for force feedback control to achieve active compliance [30]. Hence, automatic body regulation can be accomplished by using sensor information.

The Dash Mode is desired when speed of the vehicle is more important than maneuverability or fuel efficiency. In this mode, limitations are imposed on both crab angles and the body turning radius to obtain maximum foot velocities.

For very difficult terrain where human intelligence can be more appropriate than the sensors to choose proper foothold positions without concern for speed, the individual legs may be controlled by the operator by means of joystick or keyboard commands. Such a control mode is termed as "precision footing". In this mode, feedback of foot position relative to terrain is done by operator vision. The proximity sensors are used to control the foot altitude.

Since the terrain scanner is only useful for the forward locomotion, it can not be used for backward, sideway, or turning motion. The Close Maneuvering Mode of operation which requires an arbitrary motion: a combination of forward, backward, sideway, and turning motions, is for locomotion over reasonably rough terrain without terrain preview. Thus, speed of vehicle is relatively slow compared with that in the Terrain Following Mode. This is a three-axis control mode in which the body turning radius and crab angles are not restricted to achieve high degree of maneuverability. Body attitude and altitude are automatically regulated to accomplish the terrain following locomotion through the proximity or force sensors.

2.4 Supervisory Control of the OSU Hexapod

The advantages of legged vehicles over wheeled or tracked vehicles result from their greater number of controllable degrees of freedom. However, the manual control of such a large number of degrees of freedom is an exhausting task for the human operator to endure as revealed in the test of the G. E. Quadruped Vehicle.

In order to relieve a human operator of such a complex task as the manual coordination of limb motions, an appropriate control strategy for the man-machine interface was developed [8], which is called supervisory control. The basic hierarchical structure of the supervisory control scheme employed in the control of the present OSU Hexapod is shown in Figure 2.3 [8].

2.4.1 Three-axis Control

In this supervisory control scheme, the operator provides longitudinal velocity, lateral velocity, and yaw turning rate with respect to the body to the motion planning block. These velocities and rates can be generated by a three-axis joystick or a similar device.

For the OSU Hexapod control, the three-axis control mode has been implemented for locomotion over flat level terrain using a three-axis joystick [8]. In this implementation, the vehicle operation was divided into three modes called Cruise mode, Turn-in-place mode and Side-stepping mode according to the body velocity [8]. This was done in order to avoid complexity in leg-trajectory planning for unrestricted three-axis control. In all three modes, the body of the vehicle is assumed to remain at a constant height and parallel to the plane of support.

In Cruise mode, the minimum turning radius of the motion trajectory of the center of gravity of the vehicle and its crab angle is limited so that this mode is appropriate for cruising forward or backward with small components of side-to-side velocity or turning rate. Turn-in-place mode is for the turning motion of the vehicle in place about its center of gravity with the other two components of vehicle velocity forced to zero. Side-stepping mode is for the lateral motion of the vehicle without changing its heading, while the other two components of vehicle velocity are forced to zero.

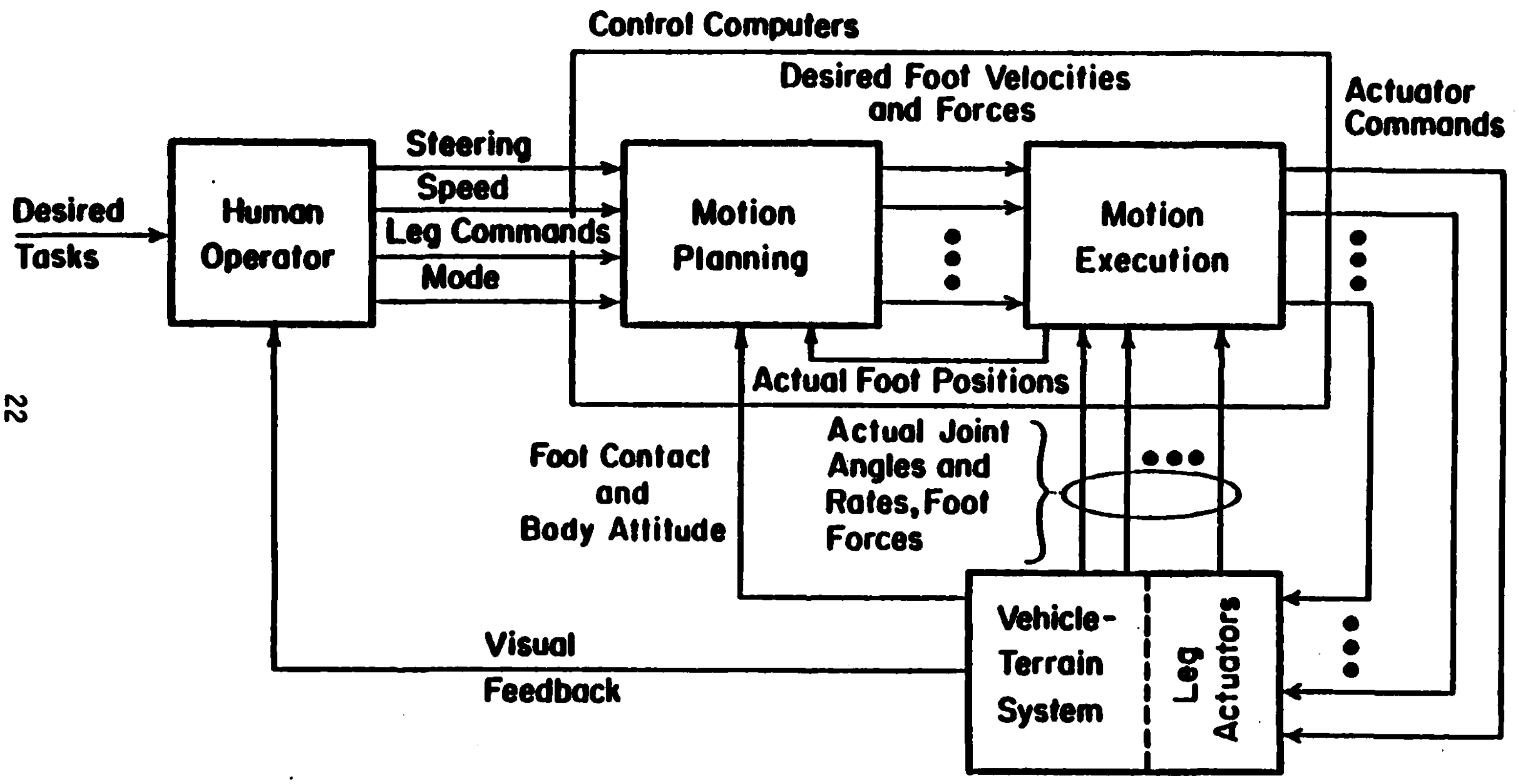


Figure 2.3. Supervisory Control Scheme Used with the OSU Hexapod [8].

For the locomotion over even terrain, the other three components of the body velocity: up/down velocity, roll rate, and pitch rate, are set to zero. Hence, the body height remains constant above the terrain and there is no attitude and altitude control during locomotion.

Based on the above control scheme, an algorithm was developed by Chang [12] for locomotion of the OSU Hexapod over uneven terrain. In this algorithm, the body roll and pitch rates are generated in the motion planning block for the automatic body attitude control so that the body may be parallel to the estimated support plane obtained from the present support points through linear regression. Also, the up/down velocity of the body is computed in the motion planning block in order to keep the body height constant above the estimated support plane. The estimated support plane is updated whenever one of the transfer legs touches ground.

2.4.2 Follow-The-Leader Control

For the full automation of the walking machine, it is necessary to raise the level of the man-machine communication scheme from simple steering of the body to that of optical designation of footholds. Toward this end, the Follow-The-Leader (FTL) control has been employed for locomotion of the OSU Hexapod vehicle over rough terrain [26]. In this mode of operation, the ranging system, which provides a terrain information only for front-leg footholds, permits the full utilization of human intelligence in selecting the footholds. The human operator specifies the desired footholds for the front pair of legs and the successive legs of each side of the body step on the same points as used by the front legs.

First, the candidate footholds are selected by the human operator through the optical sensing device. Then the computer of the vehicle checks whether or not they are inside the kinematic limit of the leg. If they are inside the kinematic limit, they are accepted by the vehicle. Otherwise, they are rejected. When the candidate footholds are accepted, then the motion planning block initiates the body motion planning by determining the desired vehicle velocity vector and the turning rate so that the vehicle moves the desired direction, which is, in contrast, provided by the human operator in the three-axis control mode. When the candidate footholds are rejected, the computer requests another selection from the human operator until an acceptable foothold is found.

One of the important problems relating to the FTL control algorithm is how to determine the vehicle center of gravity for the next cycle of body motion. Tsai [31] developed a heuristic algorithm to find such an optimal point for the vehicle center of gravity by maximizing the stability margin. The motion planning algorithm for the FTL control is described in detail in [31], which was implemented and tested for the OSU Hexapod over rough terrain in a laboratory environment.

2.5 Feedback Control

As shown in Figure 2.3, the body motions are finally executed by the motion execution block in the servomechanism level. Since this level involves the full dynamic complexity of the vehicle/terrain system, it is necessary to develop an appropriate control strategy to obtain the desired joint motions. Various control concepts have been developed for walking machines. The following paragraphs describe two of them which are presently implemented for control of the existing legs at the Ohio State University.

2.5.1 Jacobian Control

Jacobian control is based on resolved rate at the foot so that the control is made in Cartesian coordinates. Also, this control scheme incorporates both force and position feedback. The Jacobian control law is given by the following equation as

$$\underline{\dot{\theta}}_C = J^{-1}[\underline{\dot{x}}_D + K_P(\underline{x}_D - \underline{x}_A) + K_F(\underline{F}_D - \underline{F}_A)] \qquad (2.11)$$

where $\underline{x}_D$ is the desired foot position derived by the desired rate, $\underline{x}_A$ is the actual foot position, $\underline{F}_D$ is the desired foot force, and $\underline{F}_A$ is the actual foot force. The values of the elements of the gain matrices K_P and K_F determine the accomodation [32] of force errors by position errors and, for diagonal matrices, lead to the concept of active compliance [30]. The Jacobian control structure with compliance servo for the OSU Hexapod is shown in Figure 2.4 [11]. In this figure, $\underline{V}$ denotes the vector of the joint actuator input voltages and $\underline{F}_f$ denotes the foot force in the foot coordinate system.

Recently, based on the Jacobian control structure, force feedback control for active complince using force sensors and attitude control using inertial sensors have been implemented and experimentally evaluated for the OSU Hexapod [11]. In this work, each of the feet has been equipped with two semiconductor strain gauges to measure lateral forces and a piezoelectronic load cell to measure vertical forces. A vertical gyroscope and pendulums as the attitude sensors have also been used. The desired foot force $\underline{F}_D$ was computed from the conditions such that the vehicle body maintains static equilibrium. However, the

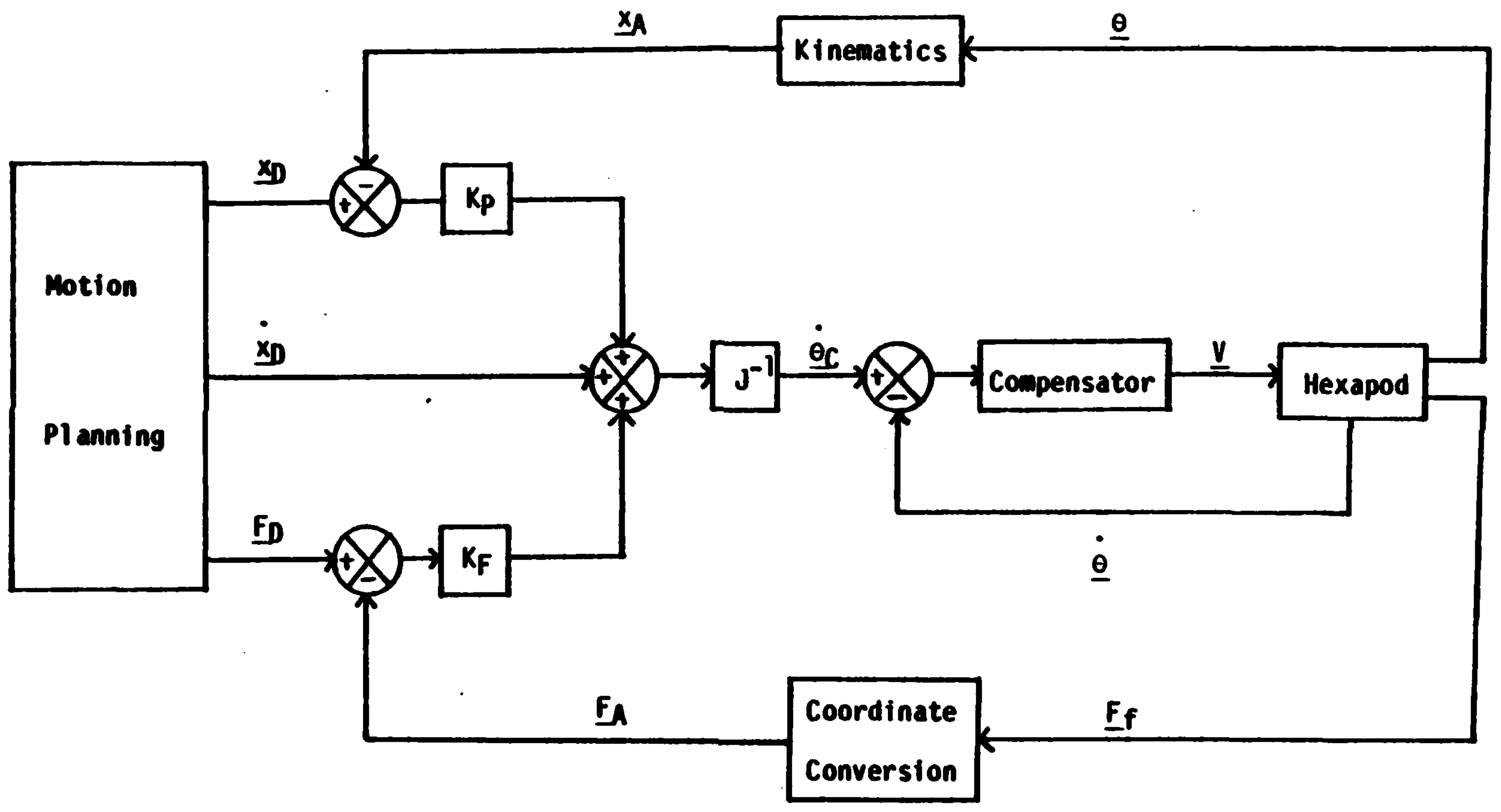


Figure 2.4. Jacobian Control Structure with Force Feedback [11].

conditions resulted in the underspecified set of linear equations. The optimal solution for $\underline{F}_D$ was obtained as a pseudo-inverse solution to these equations using the least squares of force criterion [11].

The force feedback control law for vertical active compliance under ideal rate servo assumption is given by [13]

$$\dot{z}_A = \dot{z}_D + k_P(z_D - z_A) + k_F(F_D^z - F_A^z) \qquad (2.12)$$

where $\dot{z}_A$ and $\dot{z}_D$ are the actual and desired vertical foot velocities; z_A and z_D are the actual and desired vertical positions; F_D^z and F_A^z are actual and desired vertical forces; k_P and k_F are gains. Eq. (2.10) results in the spring-damper system when the spring stiffness, k_s, and the damper coefficient, α, are chosen as [30]

$$k_s = k_P/k_F \qquad (2.13)$$

and

$$\alpha = 1/k_F. \qquad (2.14)$$

To accommodate large terrain height variations "soft" spring constants were chosen. This, however, causes the vehicle body to pitch and roll excessively. To avoid this problem, an attitude control scheme was developed [13]. From small angle approximation, the body pitch and roll errors are converted to vertical foot displacement in a linear equation:

$$\Delta z_i = -X_i \gamma_A + Y_i \delta_A \qquad (2.15)$$

where X_i and Y_i are the x and y components of the foot position of leg i with respect to the center of gravity of the body; γ_A and δ_A are the body pitch and roll angles measured by the attitude sensors.

During support phase the desired velocity for attitude control, $\dot{z}_D^A$, can be computed as

$$\dot{z}_D^A = k_A \, (X_i \, \gamma_A \; - \; Y_i \, \delta_A) \tag{2.16}$$

where k_A is the desired attitude-control gain, and z_D^A is obtained by integration of $\dot{z}_D^A$. Then the desired vertical foot position and velocity are given by

$$z_D = z_D^P + z_D^A \tag{2.17}$$

and

$$\dot{z}_D = \dot{z}_D^P + \dot{z}_D^A, \tag{2.18}$$

respectively, where z_D^P and $\dot{z}_D^P$ are the desired vertical foot position and velocity from the motion planning block. The attitude control system associated with the Jacobian control structure in Figure 2.4 is given in Figure 2.5 [11].

2.5.2 State Feedback Control

In State Feedback Control, the actual state of each of the joints (position and velocity) is compared with the desired state and is used to provide control inputs to each of the actuators, usually through a linear feedback control law. Since the steady-state error and the the feedback gains are interrelated for the overall system stability, a feedforward approach is employed to eliminate steady-state error without increasing the gains too much. For this purpose, the inverse dynamic model of the leg system is used, called inverse plant.

The inverse plant and state feedback control scheme [33] was applied to control the Monopod system [34]. The Monopod system was built at the Ohio State University to investigate the energy efficient leg of the

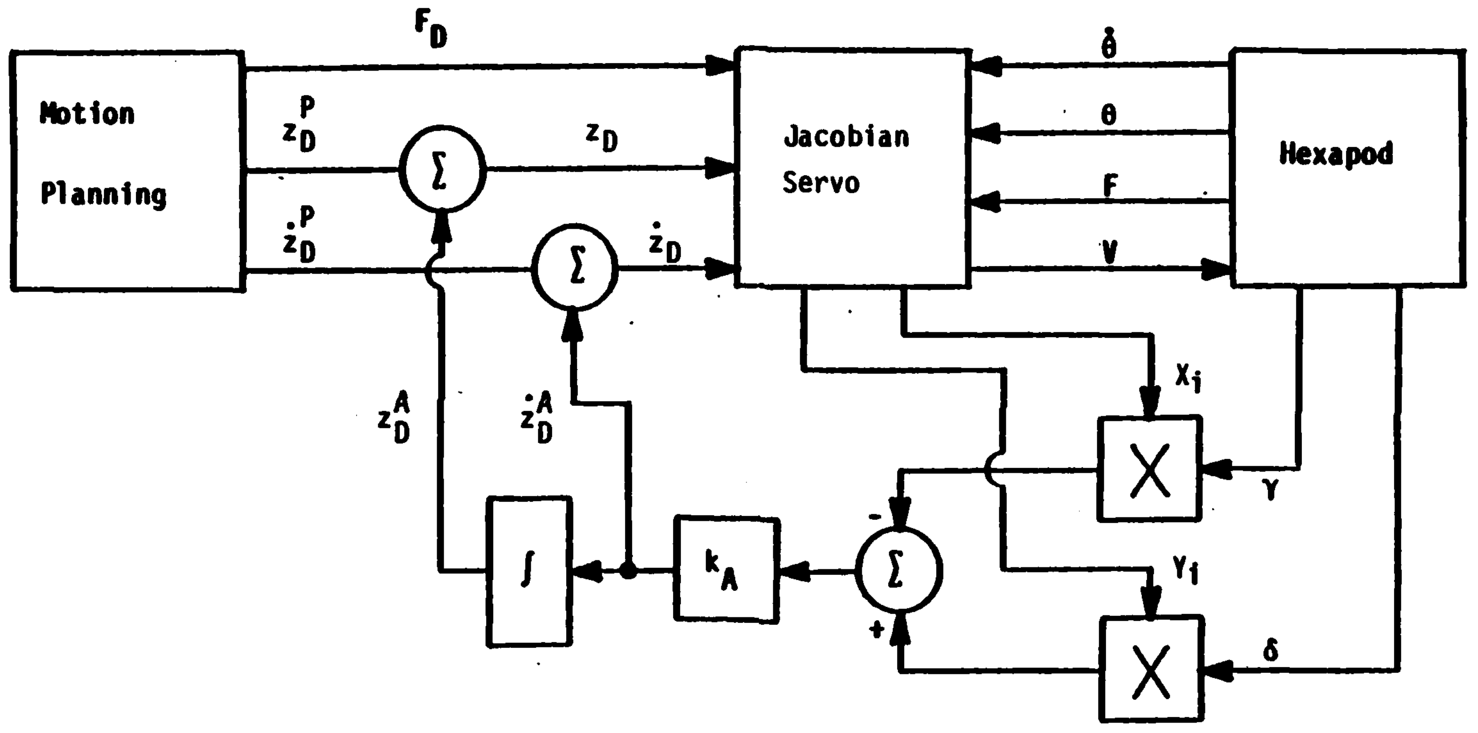


Figure 2.5. Attitude Control Associated with Jacobian Control Structure in Figure 2.4 [11].

walking machine in the laboratory and consists of a three-degree-of-freedom leg with three-wheeled cart. The basic equation describing the state feedback control law is

$$\underline{V} = K(\underline{x}_D - \underline{x}) + \underline{V}_D \qquad (2.16)$$

where $\underline{V}$ is actual actuator voltage, $\underline{V}_D$ is the nominal voltage of the actuator, $\underline{x}$ is the state variables (joint angles and rates), $\underline{x}_D$ is the desired set points for the state variables and K is the feedback gain matrix. The overall inverse plant and state feedback control structure is shown in Figure 2.6 [33].

2.6 Summary

This chapter has presented brief review of existing theory and control algorithms of legged vehicle locomotion related to this dissertation. The results of the mathematical theory that has been developed for gait selection and implementation are employed in this work.

The supervisory control scheme provides a basic structure in software implementation of control algorithms. This dissertation mainly deals with the motion planning block of the supervisory control scheme, since the low-level servo control is not necessary by the graphic simulation of the vehicle. However, the Jacobian control described in section 2.5 was implemented in the motion execution block for control of the OSU Hexapod and will, also, be implemented for control of the ASV. Based on the theory presented in this chapter, the control algorithms for locomotion of a six-legged vehicle over uneven terrain are discussed in the following chapters.

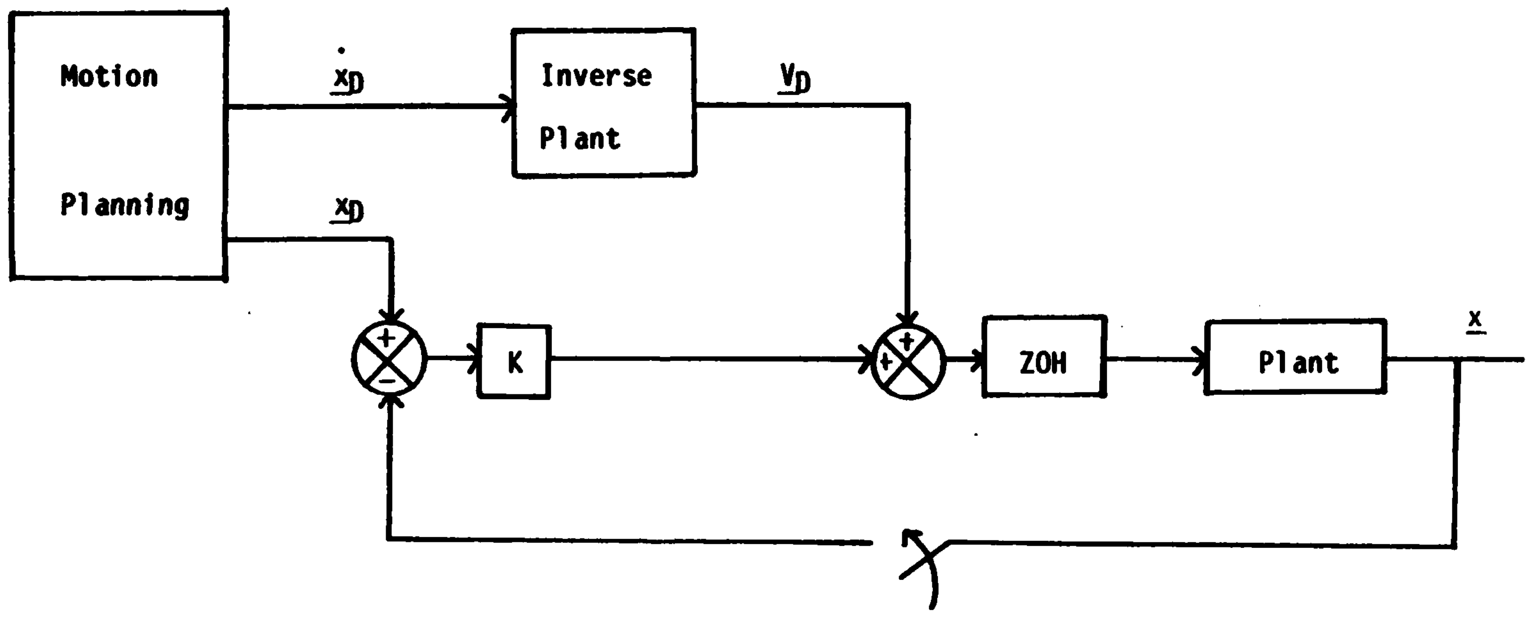


Figure 2.6. State Feedback Control Structure [33].

Chapter 3

TERRAIN AND VEHICLE SIMULATION

3.1 Introduction

The principal advantage of legged vehicles over tracked or wheeled vehicles may be superior adaptability to rough terrain in off-the-road applications. The major thrust of this dissertation is to investigate the locomotion of legged vehicles in such conditions through computer simulation. As an important basis for this work, it is necessary to define a proper vehicle/terrain model. In this chapter, the vehicle and terrain models are given.

Terrain models have often been developed for two-dimensional (2-D) analysis [9]. When three-dimensional (3-D) terrain models were used, the terrain data base has not generally been provided interactively [35]. However, in this dissertation work, various 3-D terrain models may be generated interactively so that locomotion over different kinds of terrain may be rather simply studied. In Section 3.2, a simple but realistic 3-D terrain model, called a prismatic model [35], is introduced. First, the terrain profile, which is composed of a piecewise linear curve, is interactively generated using a three-axis joystick. Then the desired perspective view of the prismatic terrain model is displayed on the screen by changing the viewing angle with the joystick.

The vehicle model used in this work is for the ASV (Adaptive Suspension Vehicle) [20] . The vehicle is a hexapod with three degrees of freedom in each of its six legs. The kinematic equations relating foot positions and rates to joint angles and rates are relatively simple since the leg is based on a pantograph mechanism. These equations are derived and given in Section 3.3.

Also, due to the mechanical limits of the joints, the reachable space of the foot is confined in a 3-D volume, called the working volume. The working volume is especially important in this work since algorithms for leg placement and liftoff as well as those for foothold selection have been based upon it. The general equations for the working volume are the same for each leg and are also developed in Section 3.3. The locomotion algorithms that use the working volume concept are presented in Chapter 4.

The ASV is a very complex system consisting of a total of 18 degrees of freedom and more than 13 mechanical links and further it will encounter a myriad of terrain conditions. Because of this, it is necessary to provide the control designer with convenient tools so that he may interactively experiment with various parameters of the control algorithms and terrain. This approach is just that of Computer-Aided Design (CAD) and has been developed to some extent in this work and is presented in Section 3.4. As stated before, the prismatic terrain model may be interactively generated. Also, locomotion of the ASV over the terrain may be viewed in any perspective. In addition to this, the instantaneous stability margin is displayed as well as the position of each of the feet within their constrained working volumes. Finally, this makes it possible to alter such parameters as the leg duty factor.

3.2 Terrain Model

The general type of terrain simulated may be defined as a 3-D surface in which the heights of the surface points are not constant so that slopes, ditches, holes, obstacles, etc., may be configured. In order to reduce the complexity of the simulation while maintaining a reasonable amount of generality, a terrain model is developed in such a way that the cross section of the terrain surface along one of the horizontal axes is constant. Furthermore, the cross section along this axis consists of a piecewise linear curve. This model may be termed a prismatic terrain model [35].

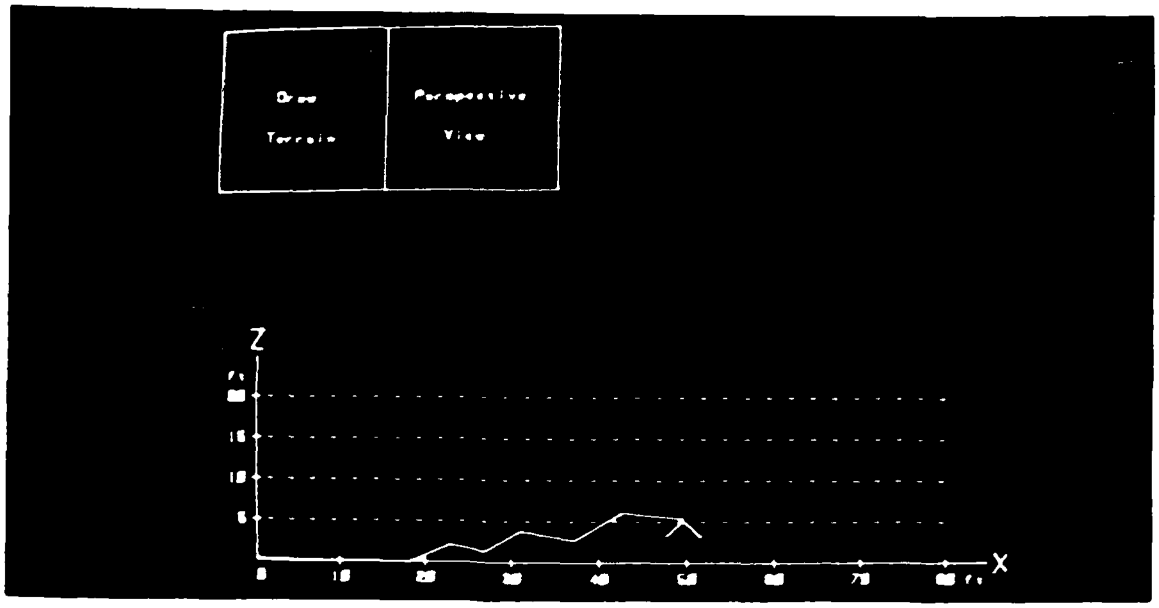
Since the body of the vehicle moves on the terrain with a full six-degree-of-freedom, the terrain conditions imposed on the vehicle at some instant during locomotion should be somewhat arbitrary with respect to the vehicle posture. The prismatic terrain model may be interactively defined to impose many different terrain conditions on the vehicle and this may provide a rather realistic walking environment.

The terrain model is defined in the earth-fixed coordinate system $(\hat{x}_E, \hat{y}_E, \hat{z}_E)$ where the $\hat{z}_E$ axis is in the opposite direction of gravitational acceleration (positive upward), and the $\hat{x}_E$ and $\hat{y}_E$ axes are appropriately defined such that $\hat{x}_E$, $\hat{y}_E$, and $\hat{z}_E$ are mutually orthogonal unit vectors. Once the earth coordinate system is fixed, then the terrain model is set up in such a way that the height (Z_E-component) of the terrain changes with changing distance along the $\hat{x}_E$ axis. The terrain width along the $\hat{y}_E$ direction is fixed to some constant value.

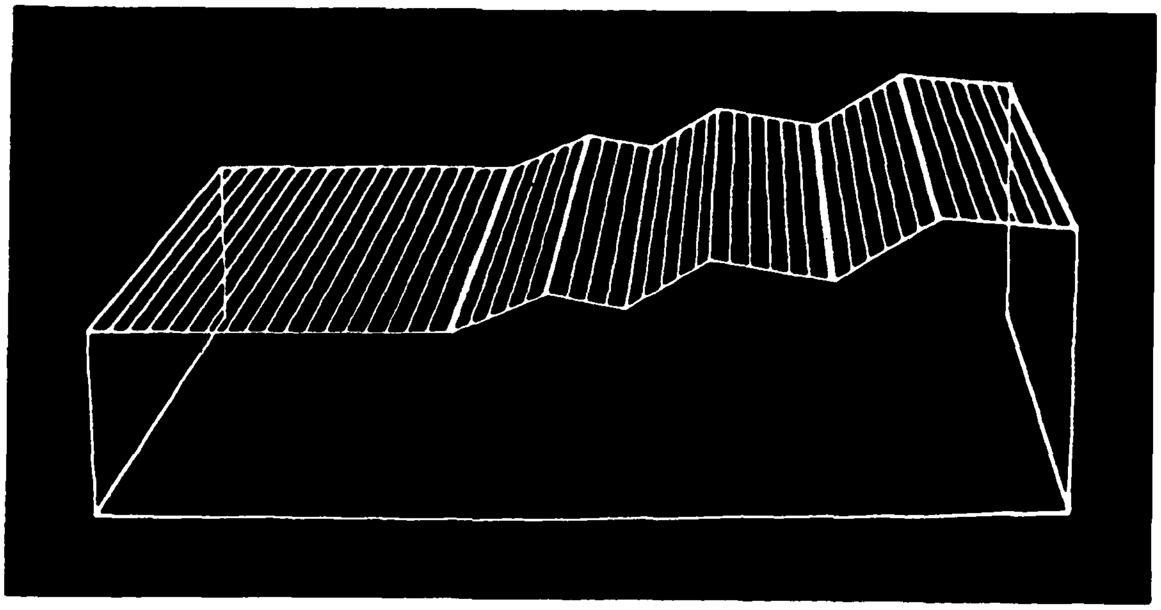
In terrain generation , a discrete set of points are specified in the Z_E-X_E plane. Then the terrain profile is generated by connecting two adjacent points with a straight line. In this manner, the profile of a smooth or undulating terrain may be modelled through the use of a piecewise linear curve. Finally, a 3-D prismatic terrain model may be obtained by including the Y_E-component to give a constant cross section in this direction. Every two adjacent points of the profile, along with the Y_E-component, compose a rectangular plane which is represented by a plane equation in 3-D space. After generating the terrain model, the terrain information is stored as an array of records in the computer main memory. Each record contains the coefficients of each rectangular plane equation and the X_E-components of two adjacent points which define the left and right boundaries of the plane in the $\hat{x}_E$ direction.

Figure 3.1a shows a typical terrain profile which is interactively generated on the screen of the HP 1350A vector graphics display using a three-axis joystick. To define the next discrete terrain surface point, the cursor (Λ) is positioned through two axes of the joystick to the desired position. A twist of the joystick sets that as the next point and connects a straight line to it. The number of points for the terrain profile is limited to 50 in this simulation.

Figure 3.1b gives a perspective view of the terrain. After the profile has been defined, then two axes of the joystick may be used to define the viewing angle of the terrain. Straight lines are also added in the $\hat{y}_E$ direction, spaced one foot apart along the $\hat{x}_E$ axis. These lines enhance the visual image of the terrain by indicating the slope of the surface. For example, the surface lines for a steep slope appear to be more densely drawn in the perspective view than those for a level surface.



(a) Terrain Profile



(b) Perspective View of the Prismatic Terrain of (a)

Figure 3.1. Computer Graphics Display of the Prismatic Terrain.

3.3 Vehicle Model

The hexapod vehicle model used in most part in this work is for the ASV which is presently under construction at The Ohio State University. The ASV is to be the first fully self-contained, computer-controlled terrain-adaptive walking vehicle. It is supported by six three-degree-of-freedom legs, each with a special kinematic configuration (pantograph) providing a simple and energy-efficient structure.

3.3.1 Body Model

The body is modelled as a hexahedral box with the top plane wider than the bottom plane, which allows the abduction/adduction motion of the legs. The six legs are attached to the body at the outside edges of the top plane.

Figure 3.2 shows a perspective of the simulation model of the ASV and its related coordinate systems. The earth-fixed coordinate system ($\hat{x}_E$, $\hat{y}_E$, $\hat{z}_E$) has been described in the previous section; it is used when specifying the absolute position and velocity of the body.

In order to describe the motion of the vehicle walking over terrain, it is also useful to define a body-fixed coordinate system which is a reference coordinate system for the operator's commands and for generating foot positions and velocities. The body coordinate system ($\hat{x}_B$, $\hat{y}_B$, $\hat{z}_B$) is fixed at the top plane of the body and its origin is located at the center of the plane. The $\hat{x}_B$ axis is along the body longitudinal axis and directed forward while the $\hat{z}_B$ axis is in the direction of the normal vector to the body plane (positive upward).

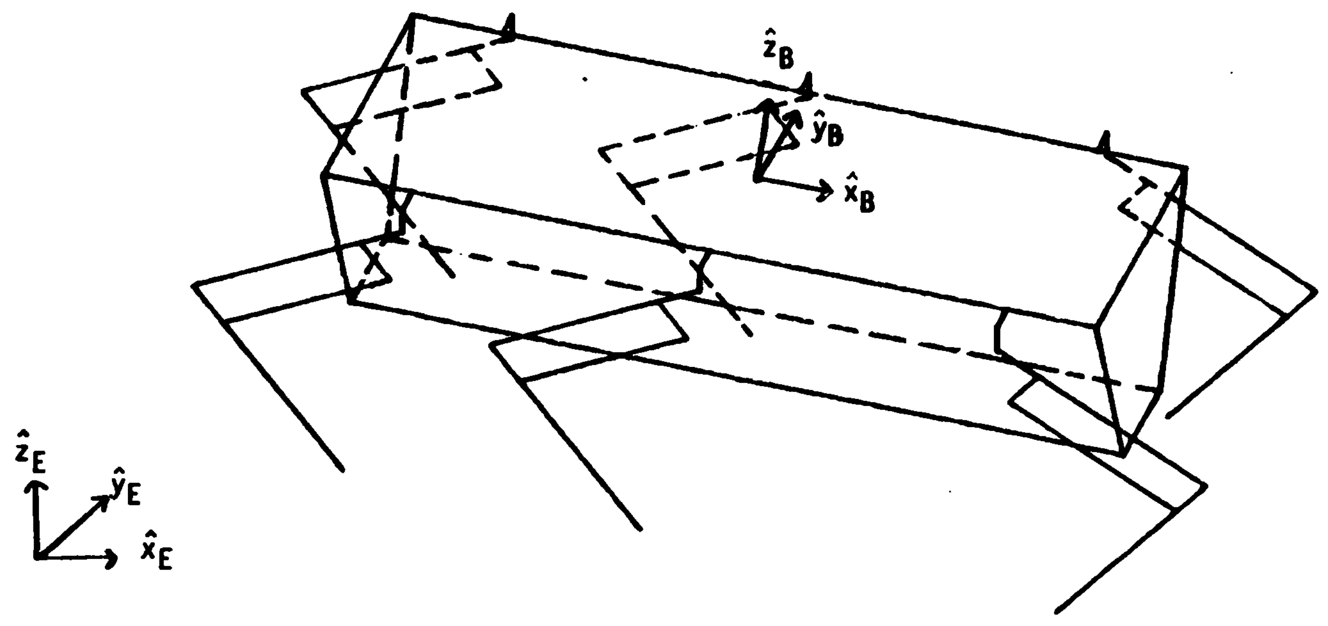


Figure 3.2. Simulation Model for the ASV.

The $\hat{y}_B$ axis is defined by the vector cross-product

$$\hat{y}_B = \hat{z}_B \times \hat{x}_B. \tag{3.1}$$

The transformation from the body coordinate system to the earth-fixed coordinate system has been defined through the use of a 4X4 homogeneous transformation matrix, H, such that [36]

$$\begin{bmatrix} X_E \\ Y_E \\ Z_E \\ 1 \end{bmatrix} = H \begin{bmatrix} X_B \\ Y_B \\ Z_B \\ 1 \end{bmatrix} \tag{3.2}$$

where $[X_E\ Y_E\ Z_E]^T$ and $[X_B\ Y_B\ Z_B]^T$ are position vectors to a common point as referenced to the appropriate coordinate systems. The matrix H may be partitioned into submatrices:

$$H = \left[\begin{array}{ccc|c} & H_r & & {}^E\underline{p}_H \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.3}$$

where H_r is a 3X3 orientation matrix and ${}^E\underline{p}_H$ is a 3X1 position vector. Any velocity vector defined in the body coordinate system (${}^B\underline{v}$) may be transformed to the earth-fixed coordinate system (${}^E\underline{v}$) by premultiplying by the orientation matrix H_r:

$${}^E\underline{v} = H_r\, {}^B\underline{v}. \tag{3.4}$$

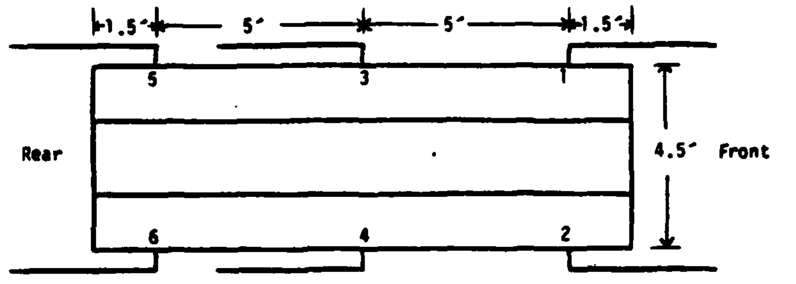
The vector ${}^E\underline{p}_H$ represents the position of the center of the body in the earth-fixed coordinate system and the orientation matrix H_r expresses the rotational property between the earth and body coordinate systems. Thus, the homogeneous transformation matrix H can be easily obtained by the relative transformation method [36] and continuously updated according to the movement of the vehicle. This will be discussed in detail in a later chapter.

3.3.2 Leg Model

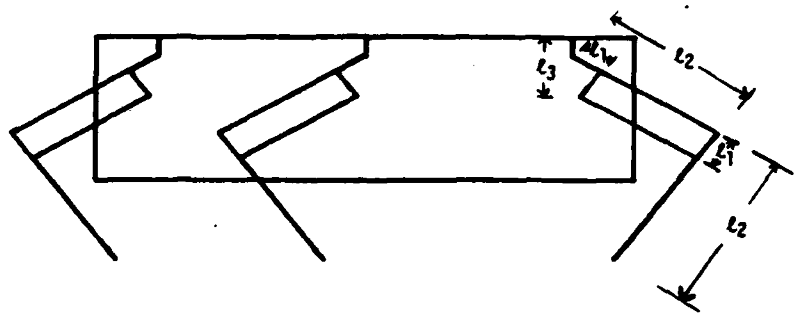
The six legs of the ASV have been designed by employing a special kinematic configuration, called a pantograph. Each leg has two one-degree-of-freedom sliding joints which are located on a base plate. One sliding joint provides up/down motion and the other gives forward/backward motion of the foot with respect to the base plate. Due to the kinematic characteristics of the pantograph, the motion of the foot tip appears to be proportional to the motion of each sliding joint through a fixed ratio. Each leg is placed on a base plate which is attached to the body through hinges to allow abduction/adduction motion of the leg. Thus, the abduction/adduction motion at the hip gives a third kinematic degree-of-freedom for each leg. The dimensions of the vehicle are shown in Figure 3.3 along with the leg model for the simulation.

3.3.2.1 Kinematics of the Leg

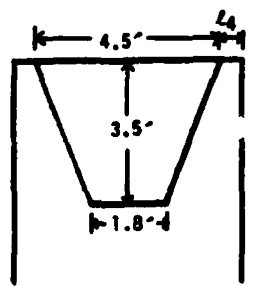
Because of the relatively simple geometry of the pantograph leg, the kinematic equations may be easily obtained by using simple geometrical principles. Let the foot position of leg 1 be (X_F,Y_F,Z_F) in the body coordinate system. As shown in Figure 3.4, the foot position of leg 1



(a) Top View



(b) Side View



ℓ_1=0.8´

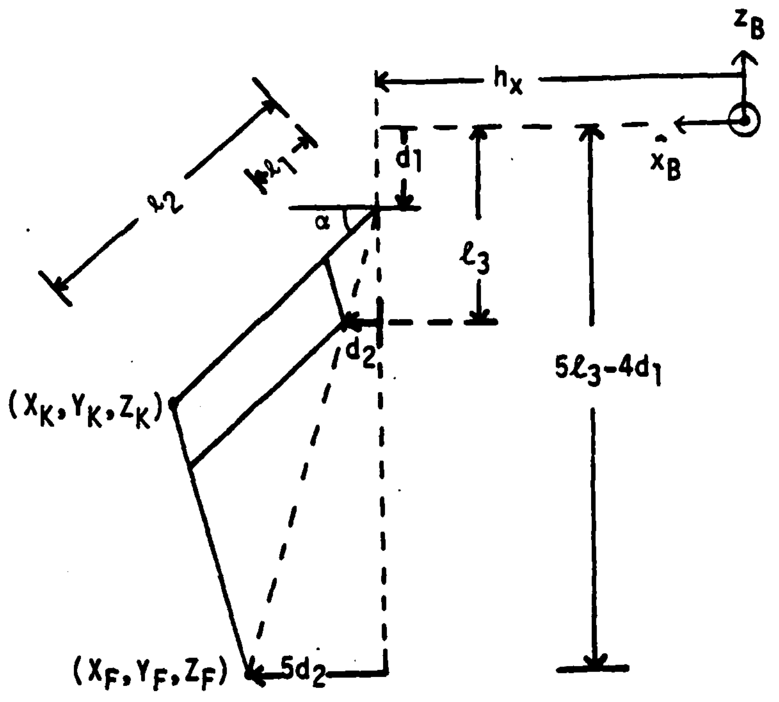
ℓ_2=4.0´

ℓ_3=1.45´

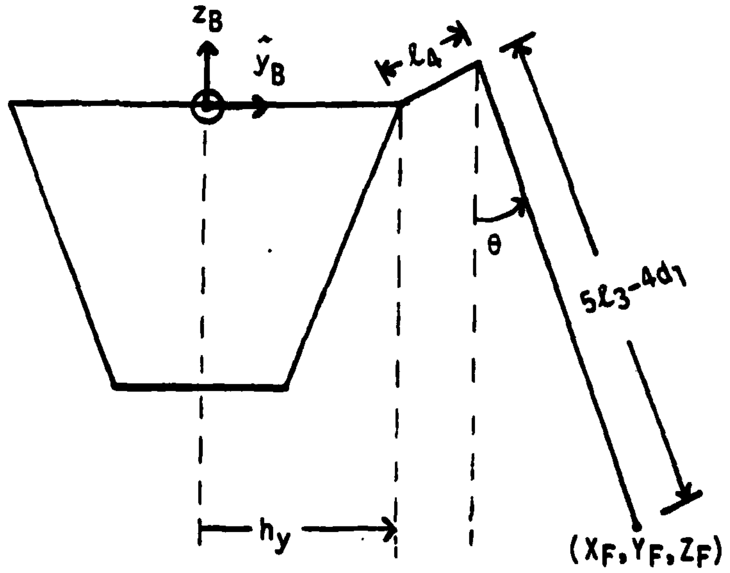
ℓ_4=0.5´

(c) Front View

Figure 3.3. Geometry of the ASV.



(a) Side View



(b) Front View

Figure 3.4. Geometrical Computation of the Foot Position.

(front left leg) is given by

$$\begin{bmatrix} X_F \\ Y_F \\ Z_F \end{bmatrix} = \begin{bmatrix} 5d_2 + h_x \\ (5\ell_3 - 4d_1)\sin\theta + \ell_4\cos\theta + h_y \\ -(5\ell_3 - 4d_1)\cos\theta + \ell_4\sin\theta \end{bmatrix} \tag{3.5}$$

where θ, d_1, and d_2 are the joint variables. Since all of the six legs have the same basic configuration, the kinematic equations for each may be obtained from Eq. (3.5) with appropriate sign changes.

The knee position (X_K, Y_K, Z_K) of leg 1 may be obtained by also applying simple geometrical principles and is given:

$$\begin{bmatrix} X_K \\ Y_K \\ Z_K \end{bmatrix} = \begin{bmatrix} \ell_2\cos\alpha + h_x \\ (\ell_2\sin\alpha + d_1)\sin\theta + \ell_4\cos\theta + h_y \\ -(\ell_2\sin\alpha + d_1)\cos\theta + \ell_4\sin\theta \end{bmatrix} \tag{3.6}$$

where

$$\alpha = \frac{\pi}{2} - \tan^{-1}\frac{d_2}{\ell_3 - d_1} - \cos^{-1}\frac{\sqrt{(\ell_3 - d_1)^2 + d_2^2}}{2\ell_1}\ . \tag{3.7}$$

Again, since all of the six legs have the same basic configuration, the equations for the positions of the knees of the other legs may be obtained from Eq. (3.6) with appropriate sign changes.

The inverse kinematic equations may be derived from Eq. (3.5) in the following manner. From the equation of the first row, d_2 is

$$d_2 = \frac{1}{5}(X_F - h_x). \tag{3.8}$$

Squaring the both sides of the equations of the second and third rows, respectively, yields

$$a^2\sin^2\theta + 2a\ell_4\sin\theta\cos\theta + \ell_4^2\cos^2\theta = (Y_F - h_y)^2 \tag{3.9}$$

$$a^2\cos^2\theta - 2a\ell_4\sin\theta\cos\theta + \ell_4^2\sin^2\theta = Z_F^2, \tag{3.10}$$

where $a = 5\ell_3 - 4d_1$. Adding Eqs. (3.9) and (3.10) and then rearranging the result, it is obtained

$$d_1 = \frac{1}{4}\left(5\ell_3 - \sqrt{(Y_F - h_y)^2 + Z_F^2 - \ell_4^2}\right). \tag{3.11}$$

From the equation of the third row of Eq. (3.5), θ can be solved by the following trigonometric substitutions

$$-\ell_4 = r\cos\phi \tag{3.12}$$

$$-a = r\sin\phi \tag{3.13}$$

where

$$a = 5\ell_3 - 4d_1 \tag{3.14}$$

$$r = \sqrt{\ell_4^2 + a^2} \tag{3.15}$$

$$\phi = \tan^{-1}\frac{-\ell_4}{-a}. \tag{3.16}$$

Then θ is given by

$$\theta = \phi - \tan^{-1}\frac{Z_F}{-\sqrt{r^2 - Z_F^2}} . \tag{3.17}$$

For the computer simulation of the ASV, only the inverse kinematic equations are used. For velocity control of the vehicle, however, the inverse of the Jacobian, which converts foot velocity commands to joint rate commands for the leg actuators, is required. Hence, the 3X3

Jacobian matrix, J, is defined as follows:

$$\begin{bmatrix} \dot{X}_F \\ \dot{Y}_F \\ \dot{Z}_F \end{bmatrix} = J \begin{bmatrix} \dot{\theta} \\ \dot{d}_1 \\ \dot{d}_2 \end{bmatrix} \tag{3.18}$$

where

$$J = \begin{bmatrix} \frac{\partial X_F}{\partial \theta} & \frac{\partial X_F}{\partial d_1} & \frac{\partial X_F}{\partial d_2} \\ \frac{\partial Y_F}{\partial \theta} & \frac{\partial Y_F}{\partial d_1} & \frac{\partial Y_F}{\partial d_2} \\ \frac{\partial Z_F}{\partial \theta} & \frac{\partial Z_F}{\partial d_1} & \frac{\partial Z_F}{\partial d_2} \end{bmatrix} . \tag{3.19}$$

The elements of J for leg 1 may be obtained by partial differentiation of the right hand side of Eq. (3.5), which gives

$$J = \begin{bmatrix} 0 & 0 & 5 \\ a\cos\theta - \ell_4 \sin\theta & -4\sin\theta & 0 \\ a\sin\theta + \ell_4\cos\theta & 4\cos\theta & 0 \end{bmatrix} \tag{3.20}$$

where $a = 5\ell_3 - 4d_1$. Since the Jacobian matrix is so simple, its inverse may be easily obtained. The inverse Jacobian, J^{-1}, is given by

$$J^{-1} = \begin{bmatrix} 0 & \frac{1}{a}\cos\theta & \frac{1}{a}\sin\theta \\ 0 & -\frac{1}{4a}(a\sin\theta + \ell_4\cos\theta) & \frac{1}{4a}(a\cos\theta - \ell_4\sin\theta) \\ \frac{1}{5} & 0 & 0 \end{bmatrix} . \tag{3.21}$$

3.3.2.2 Working Volume for a Leg

The mechanical limits of the joints restrict the motion of the leg and this is a major factor to consider when developing control algorithms for the legged vehicle. Therefore, it is necessary to define the volume of points in space that the foot of a leg can reach, called the working volume, in order to appropriately specify the foothold positions that insure that all the joint positions are within their mechanical limits.

Since each leg has the same geometrical configuration and joint limits, the working volumes of each leg are identical. The limits of the joint variables for leg 1 are as follows:

$$\begin{aligned} -10^\circ &< \theta < 21^\circ \\ 0' &< d_1 < 1' \\ -0.6' &< d_2 < 0.6'. \end{aligned} \qquad (3.22)$$

Another condition which may be considered is the case where the leg stretches out in such a way that all the joints of the pantograph are located on a straight line. This may be expressed in the following equation:

$$(\ell_3 - d_1)^2 + d_2^2 = (2\ell_1)^2. \qquad (3.23)$$

However, the joint limits given in Eq. (3.22) guarantee that this case will never occur.

The analytic expressions for the working volume may be derived from Eq. (3.5). Without loss of generality, let $h_x = h_y = 0$. Then, Eq. (3.5) reduces to

$$X_F = 5d_2$$
$$Y_F = a \sin\theta + \ell_4 \cos\theta \qquad (3.24)$$
$$Z_F = -a \cos\theta + \ell_4 \sin\theta$$

where $a = 5\ell_3 - 4d_1$. By eliminating θ from Eq. (3.24), the following is obtained:

$$Y_F^2 + Z_F^2 = a^2 + \ell_4^2. \qquad (3.25)$$

From Eqs. (3.22) and (3.24), the limits of (a) are given by

$$5\ell_3 - 4 < a < 5\ell_3. \qquad (3.26)$$

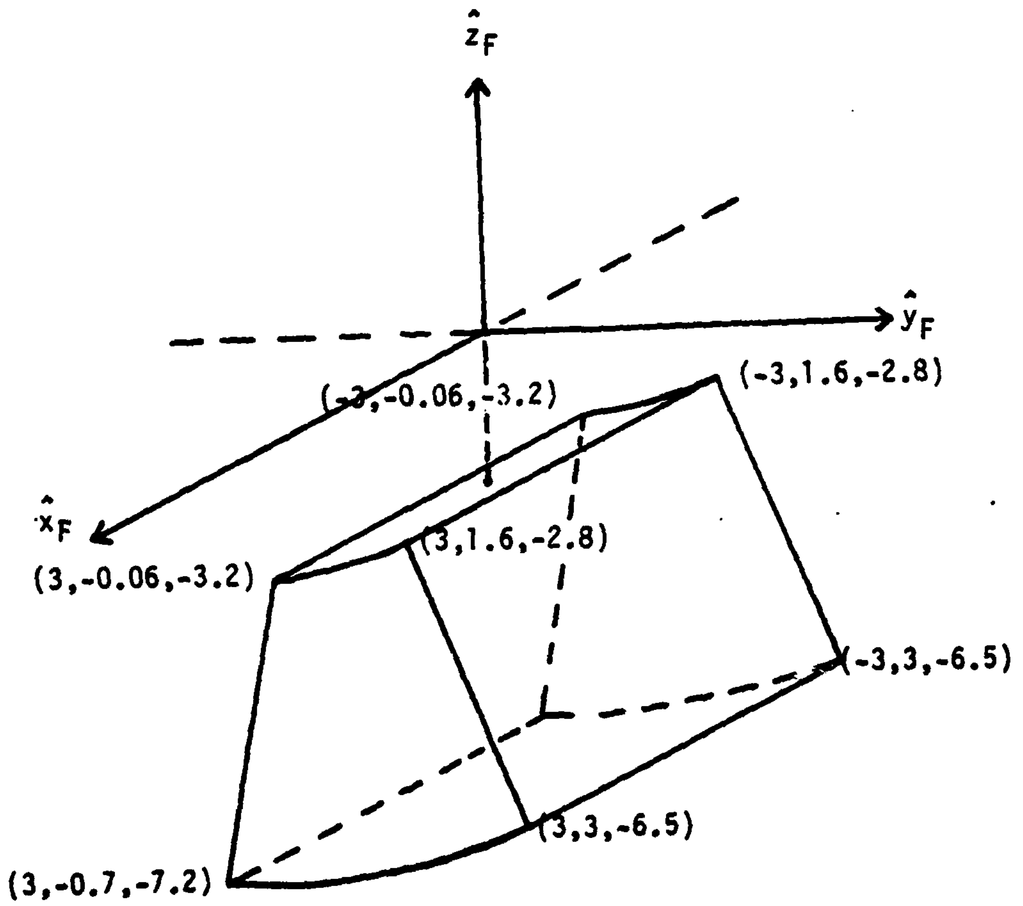
Also, eliminating (a) from Eq. (3.24) gives

$$Z_F = -Y_F \cot\theta + \ell_4/\sin\theta. \qquad (3.27)$$

With consideration of the joint limits given in Eq. (3.22) and (a) in Eq. (3.26), the Eqs. (3.25) and (3.27) define the working volume as a sector of an annulus in the Y_F-Z_F plane using the leg dimensions given in Figure 3.3. The 3-D working volume may be obtained by extending the cross section in the Y_F-Z_F plane to the $\hat{x}_F$ direction as shown in Figure 3.5. Note that the origin of the coordinate system $(\hat{x}_F, \hat{y}_F, \hat{z}_F)$ is located at the hip socket and each axis is parallel to the corresponding axis of the body coordinate system.

3.4 Computer-Aided Design

Computer-Aided Design (CAD) systems are now widely used in many engineering fields, which has largely resulted from the development of sophisticated 3-D interative graphics hardware and software. The purpose of CAD systems is to reduce costly prototyping by substituting



units = ft

Figure 3.5. Dimensions and Shape of the Working Volume for the Legs of the ASV-84 Vehicle.

3-D computer models for physical models. The computer simulation in this work has been accomplished using the PDP-11/70 minicomputer along with the HP 1350A vector graphics device which displays the perspective view of the vehicle and terrain, the top view of the vehicle, and finally its support pattern. The overall display is shown in Figure 3.6 and represents a CAD approach to the development of the controls for the ASV.

The perspective view of the vehicle and terrain allows one to examine the overall performance of the control algorithms for 3-D locomotion of the ASV. As has been discussed in Section 3.2, there is considerable flexibility in designing the terrain cross-section and in specifying the viewing angle.

The top view is generated by projecting the vehicle onto the X_B-Y_B plane of the body coordinate system. The top view of the constrained working volumes for each of the legs is also included. The constrained working volume for a leg is just the rectangular solid volume, inside the actual working volume, in which the leg control algorithms attempt to maintain the foot motion (see Chapter 4). In addition to the constrained working volume, the position of the foot within it is clearly shown. Furthermore, the support foot which is "nearest" the edge of its constrained working volume (according to the algorithms developed in Chapter 4), is indicated by a line from the foot to the <u>critical edge</u> (see Chapter 4). Finally, any problems with leg collisions may be determined by taking note of the top view.

The support pattern [23] is displayed in order to investigate the degree of static stability during the motion of the vehicle. The support pattern is a polygon projected onto the X_E-Y_E plane of the earth-fixed coordinate system, and the projection of the center of gravity is

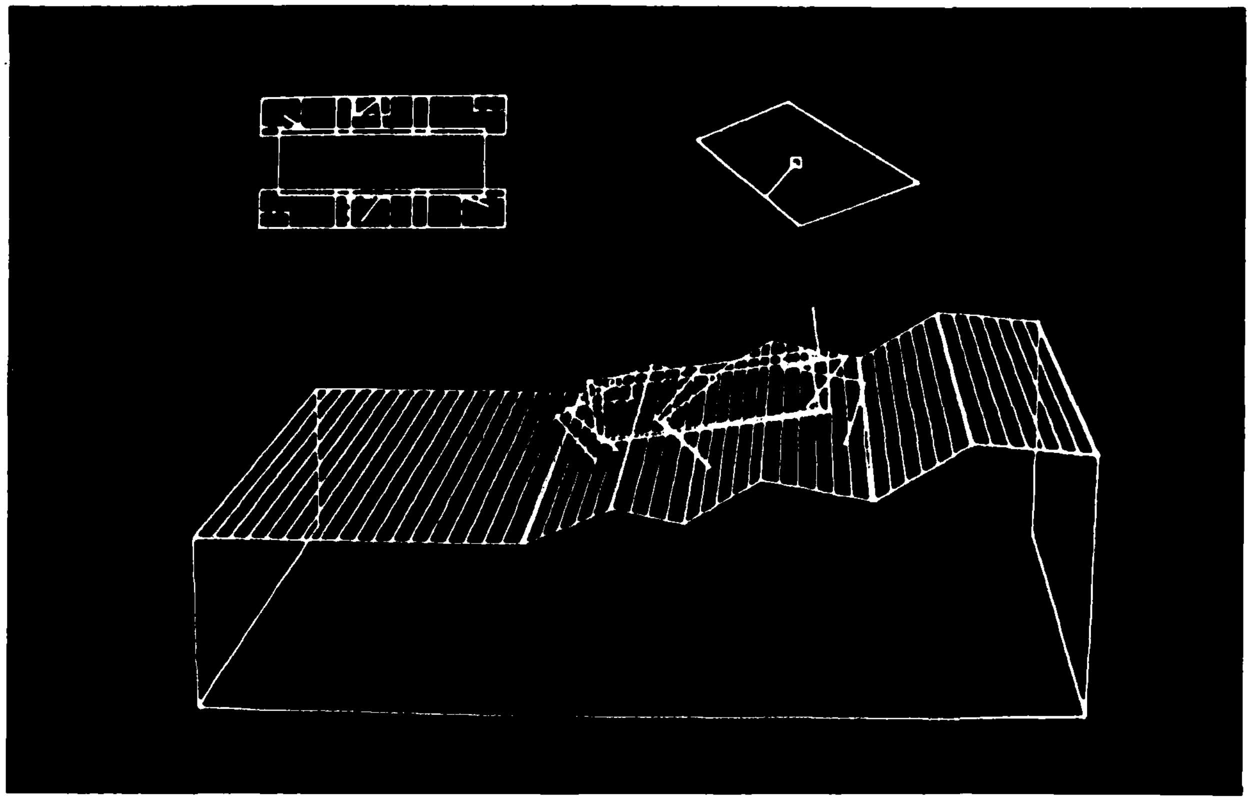


Figure 3.6. Graphics Display of the Vehicle/Terrain, the Top View, and the Support Pattern for the ASV.

also shown. A line from the center of gravity to the nearest edge of the support polygon gives an indication of the instantaneous stability margin, and this is also computed and displayed.

The stability margin is computed in the following manner. Let two adjacent vertices of the support polygon be (X_i, Y_i) and (X_{i+1}, Y_{i+1}) in the earth-fixed coordinate system. Then the equation of the line which passes through two vertices may be expressed as

$$AX + BY + C = 0 \tag{3.28}$$

where

$$A = Y_{i+1} - Y_i \tag{3.29}$$

$$B = X_i - X_{i+1} \tag{3.30}$$

$$C = Y_i X_{i+1} - X_i Y_{i+1}. \tag{3.31}$$

Assume that the center of gravity of the vehicle, (X_C, Y_C), is given. The perpendicular distance from the center of gravity to the line, D_i, is obtained as follows:

$$D_i = \frac{|AX_C + BY_C + C|}{\sqrt{A^2 + B^2}} \; . \tag{3.32}$$

Hence, the minimum of the D_i's gives the stability margin, and the corresponding edge of the support pattern is specified.

The center of gravity of the vehicle is constantly changing, but may be computed since the mass of the body and the legs as well as their positions are given. However, for simplicity, the center of gravity used in this work for the ASV is assumed to be the geometric center of the body.

The computation of the stability margin as just described, where it is measured with respect to the vertical projection of the center of gravity, does not include the dynamic effects of the inertial acceleration of the body and legs. As such, even though the vertical projection of the center of gravity is contained within the support pattern, the vehicle may be unstable since the projection of the center of gravity along the total acceleration vector (both gravitational and inertial components) may fall outside the support pattern. In this regard in the general case, the stability margin should be measured while including the inertial effects of the massive legs and the body.

The inertial and gravitational forces on the body and legs are balanced by the external forces acting on the vehicle at its feet. The net effect of these ground reaction forces may be replaced by a single force (which is the sum of all ground reaction forces) acting at the zero moment point [3] or center of pressure. The zero moment point is defined as the point in the support plane about which the moment due to the vertical ground reaction forces is equal to zero [3]. The requirement for stability is that the zero moment point lies within the support pattern [23]. When the inertial accelerations are zero, then the zero moment point is just the vertical projection of the center of gravity onto the support plane.

Since the complete specifications for the mass of the body and legs of the ASV have not yet been made, the stability margin for the ASV has not been based on the zero moment point concept. However, in previous simulation work for the OSU Hexapod walking over flat level terrain, the zero moment point was computed in the following way.

Assume that all the leg mass is concentrated at the knee joint. (A large motor at the knee is the most of the leg mass.) Also, assume that the rotational accelerations about the pitch and roll axes are negligible since the motion is for flat level terrain. Then the following equations, with components expressed in the body coordinate system, are satisfied:

$$\begin{aligned}\sum M_x &= Y_{ZM}F_z - Z_{ZM}F_y \\ &= \sum_{i=0}^{6}[Y_i m_i(-g + \ddot{Z}_i) - Z_i m_i \ddot{Y}_i] \quad ,\end{aligned} \tag{3.33}$$

and

$$\begin{aligned}\sum M_y &= -X_{ZM}F_z + Z_{ZM}F_x \\ &= \sum_{i=0}^{6}[-X_i m_i(-g + \ddot{Z}_i) + Z_i m_i \ddot{X}_i] \quad ,\end{aligned} \tag{3.34}$$

where $\sum M_x$ and $\sum M_y$ are total moments about the $\hat{x}_B$ and $\hat{y}_B$ axes, respectively; X_{ZM}, Y_{ZM}, and Z_{ZM} are the components of the position vector for the zero moment point; X_i, Y_i, and Z_i are the components of the position vector for the center of mass of the body (i=0) or leg i; $\ddot{X}_i$, $\ddot{Y}_i$, and $\ddot{Z}_i$ are the translational accelerations of the center of mass i; m_i is mass of the body (i=0) or leg i; g is the gravitational acceleration; F_x, F_y, and F_z are external ground reaction forces onto the feet. The external forces are given by

$$F_x = \sum_{i=0}^{6} m_i \ddot{X}_i \tag{3.35}$$

$$F_y = \sum_{i=0}^{6} m_i \ddot{Y}_i \tag{3.36}$$

$$F_z = \sum_{i=0}^{6} m_i(-g + \ddot{Z}_i). \tag{3.37}$$

Note that the gravitational vector is assumed to be in the $-\hat{z}_B$ direction in Eq. (3.37); this is the case when the body is parallel to a level terrain. Also, since the vehicle is always contacting the ground, the zero moment point is always located on the ground plane. Thus, Z_{ZM} is the body height which is known. From Eqs. (3.33) and (3.34), X_{ZM} and Y_{ZM} may be obtained:

$$X_{ZM} = \frac{-\sum M_y + Z_{ZM} F_x}{F_z} \tag{3.38}$$

$$Y_{ZM} = \frac{\sum M_x + Z_{ZM} F_y}{F_z} \quad . \tag{3.39}$$

The translational acceleration of the body and the angular accelerations of each joint are calculated by first order numerical differentiation of the body velocity and the joint rates. Using those values, the translational acceleration at the center of mass of the leg is computed through use of the kinematic equations given in [37].

Figure 3.7 shows the perspective view, the top view, and the support pattern of the OSU Hexapod during locomotion over flat level terrain. It is noted in this figure that the zero moment point is not coincident with the vertical projection of the center of gravity when the OSU Hexapod suddenly halts.

By displaying three different features of the motion of either the ASV or the OSU Hexapod vehicle on the graphics display, evaluation of the control algorithms developed is enhanced. Also, it is possible to interactively change various parameters for the vehicle and terrain and to immediately view the effect on the control. The initial effectiveness of interactively using the graphics display will hopefully motivate

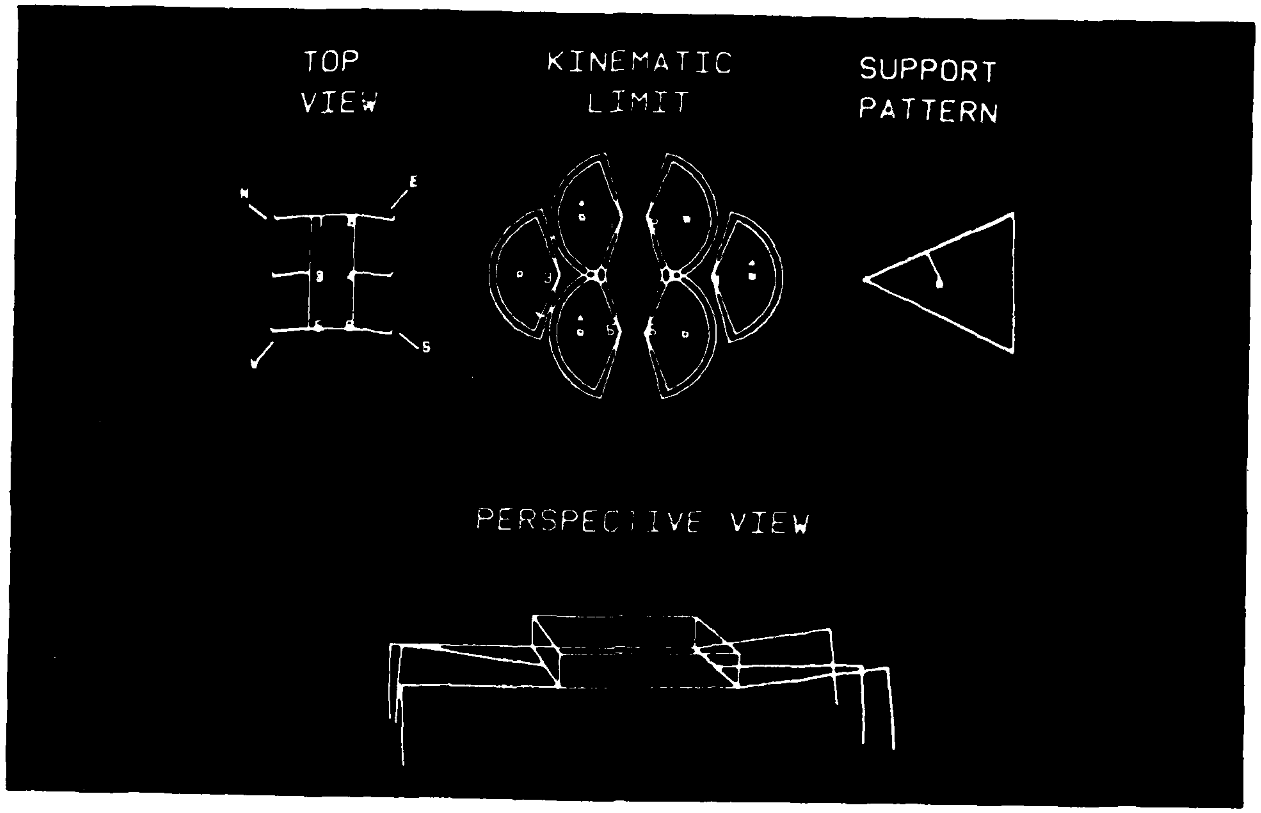


Figure 3.7. Graphics Display of the Vehicle, the Top View, and the Support Pattern for the OSU Hexapod.

the extended use of Computer-Aided Design in the development of the control algorithms for legged vehicles.

3.5 Summary

In this chapter, terrain and vehicle models were presented. Furthermore, the Computer-Aided Design concept was introduced for development of the control algorithms using the interactive graphics device.

The terrain model developed is simple but realistic enough to investigate the legged vehicle locomotion problem over rough terrain. Based on the 2-D terrain profile, the 3-D prismatic terrain model may include various kinds of terrain irregularities such as holes, ditches or slopes. Furthermore, the terrain can be interactively generated on the graphics display using a three-axis joystick.

The vehicle model of the ASV has been defined and its kinematic equations have been derived. From the mechanical joint limits, the working volume of the leg has been computed in 3-D space, which will be an important factor in developing control algorithms for locomotion over uneven terrain as presented in the next chapters.

Finally, by displaying three different features of the motion of the vehicle on a graphics display: the perspective view of the vehicle on the terrain, the top view, and the support pattern, control algorithms developed may be more simply evaluated by observation. The ease of evaluation along with interactive change of various parameters of control algorithms may motivate the extended use of Computer-Aided Design in the development of the control algorithms for legged vehicle locomotion.

Chapter 4
OMNIDIRECTIONAL CONTROL

4.1 Introduction

As discussed in Chapter 2, the operation of the OSU Hexapod, using a three-axis joystick for supervisory control, was divided into three different control modes--Cruise mode, Turn-in-place mode and Side-stepping mode. This approach was taken in order to avoid the complexity of leg trajectory planning for unrestricted three-axis joystick control [8]. The complexity mainly stems from the difficulty of changing the locomotion period from mode to mode *without* stopping the vehicle. However, experience with the previous control algorithm now makes it possible to develop a new one which integrates the above three joystick control modes into one so that omnidirectional control is provided, and this is the subject of the present chapter.

Previously, in [8], the period was defined separately for each mode as the ratio of a particular stride length, associated with the basic motion in that mode, and a component of body translational or rotational rate. This definition of the period resulted in the need to limit the motion of the body in each mode. For instance, the turning radius and the body crab angle [8] are limited in the Cruise mode, the lateral and longitudinal velocity components are forced to zero in the Turn-in-place mode, and the turning rate and the longitudinal velocity component are forced to zero in the Side-stepping mode. Experience with the algorithm indicated that the legs would attempt to go out of their kinematic limits unless the motion was sufficiently restricted.

In order to combine the three different modes, the limitations imposed on the motion of the vehicle have been eliminated so that omnidirectional control is provided. The resulting control scheme has been applied to the ASV and provides the basis for the Close Maneuvering mode of operation [14].

The basic approach of the new algorithm is a "bottom-up" control strategy [25] in which the period is selected on the basis of reachability of each leg. In other words, the period is defined such that the foot positions of each leg are specified inside or, at most, on the boundary of the working volume. However, the original working volume cannot be used directly for this purpose because it contains some reachable points which may cause instability during the vehicle motion. To ensure static stability, the concept of a constrained working volume is used and introduced in Section 4.2.

In Section 4.3, an algorithm for computation of an optimal period based on the constrained working volume is discussed. The optimal period for each locomotion cycle is determined based on how close each leg is to its own kinematic limits in time and phase. This optimal period ensures that at least one of the six legs fully utilizes its constrained working volume, while the other legs are inside their own kinematic limits. In this sense, the period obtained is optimal.

In periodic gaits as used in this work, the locomotion period determines the leg swing time. Mechanically, the leg liftoff, transfer forward, and placement takes at least a finite amount of time, called the transfer time threshold. If the optimal period is too short to swing the leg, it is necessary to slow down the vehicle to achieve an appropriate leg sequencing determined by the transfer time threshold. In this regard, the deceleration plan [29] is presented in Section 4.4.

4.2 Constrained Working Volume

As discussed in Chapter 3, the working volume of the foot is given by the mechanical limits of the joints. It is noted in the simulation that the center of gravity of the vehicle goes outside the support pattern, which indicates static instability, at some instants during locomotion, even if the support points of the feet are constrained to be inside their working volumes. The problem is due in part to the fact that the initial criterion for foot positioning used was not based on the stability of the vehicle, but simply on the kinematic limits of each leg. In order to correct this, consideration was given to further constraining the movement of the feet inside their working volumes so as to ensure static stability.

The minimum number of support legs to maintain static stability of the vehicle is three; that is, the triangular support pattern is the minimal set of support points. In the wave gaits used in this work, the middle leg of one side is in the support phase along with the front and rear legs of the other side for the tripod gait ($\beta = 1/2$). If the stability of the vehicle is ensured for the tripod gait by appropriately constraining the support points of the three contacting feet within a subset of the working volume, then stability for other wave gaits which have more than three legs in support ($\beta > 1/2$) is also guaranteed for the same constraints on the support points. Figure 4.1 illustrates the case to be corrected where the triangular support pattern of the tripod gait shows instability for the ASV.

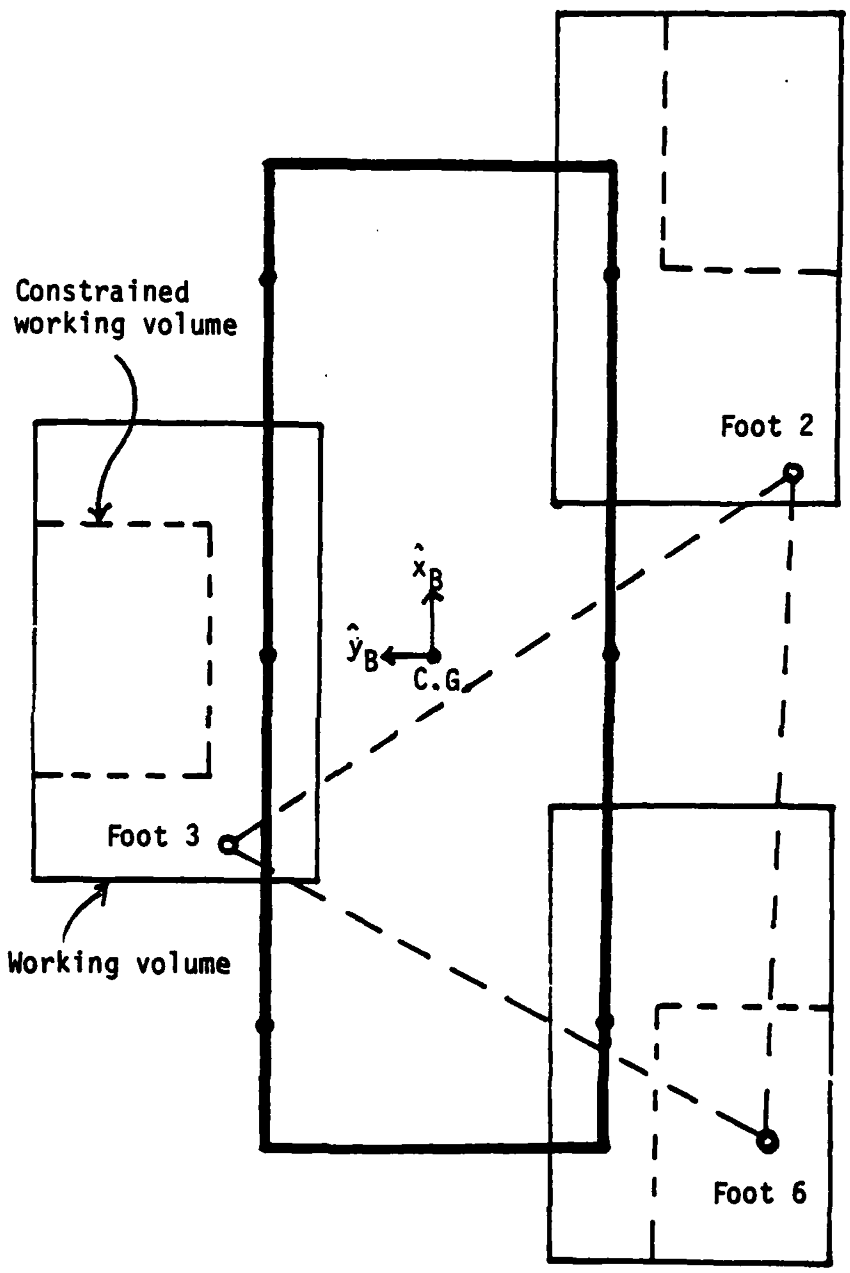


Figure 4.1. Unstable Support Pattern for the ASV with a Tripod Gait Using the Original Working Volume for Each Leg (Top View).

The foregoing discussion leads to the use of the constrained working volume in the control algorithms. It is defined as a subset of the original working volume for each leg that ensures static stability when there is a tripod or more of support. The constrained working volume sets soft limits for each leg so as to exclude points from the working volume that may lead to instability.

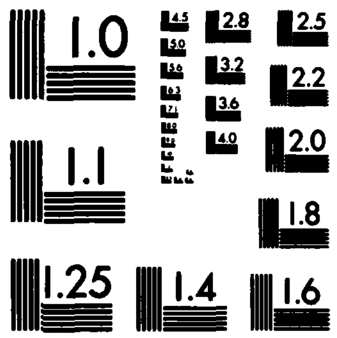
To simplify the control algorithms, the constrained working volume for each leg is specified as a rectangular solid volume. Its size and location is set by six parameters: $[X_{Ci}, Y_{Ci}, Z_{Ci}]$ and $[d_{xi}, d_{yi}, d_{zi}]$. $[X_{Ci}, Y_{Ci}, Z_{Ci}]$ give the center of the constrained working volume for leg i in body coordinates while $[d_{xi}, d_{yi}, d_{zi}]$ are its dimensions along the body axes. Figure 4.2 shows the top and side views of the original working volumes and constrained working volumes for each leg of the ASV. The solid lines represent the original working volume and the dashed lines represent the soft limits imposed by the constrained working volumes. Note that both the top and bottom sides of the original working volume consist of curved surfaces. Thus, the solid lines merely represent the maximum size of the volumes in 2-D space.

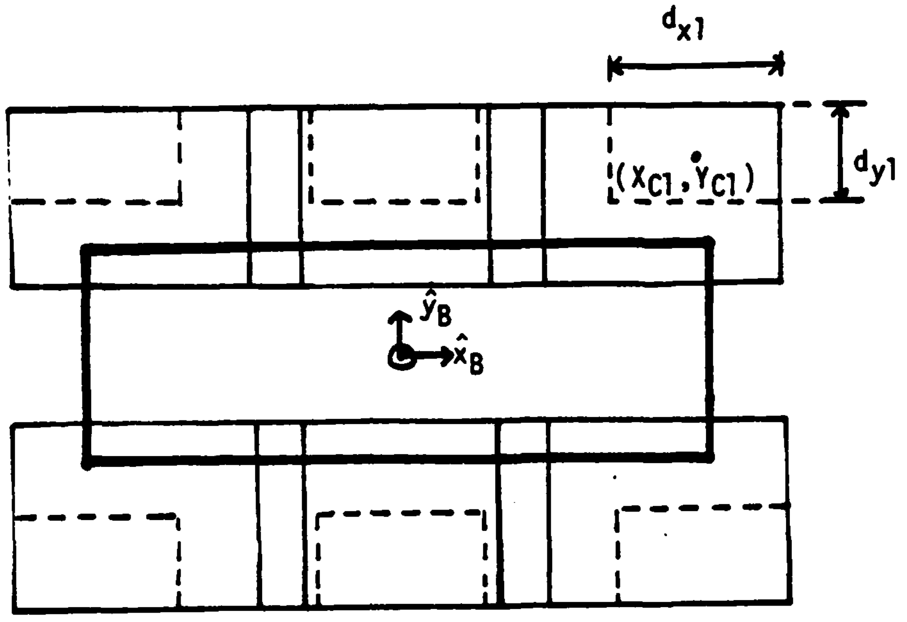
Stability of the vehicle motion may be examined by changing the location of the soft limits of each leg imposed by the constrained working volume. It is noted from the simulation results that the further the locations of the soft limits are away from the center of gravity of the vehicle, the better the stability margin of the vehicle that is obtained (see Figure 4.2a).

It must also be stated that the previous discussion considers locomotion over flat level terrain only. Locomotion over uneven and sloping terrain is considered in the next chapter and further adjustments to maintain static stability are made.

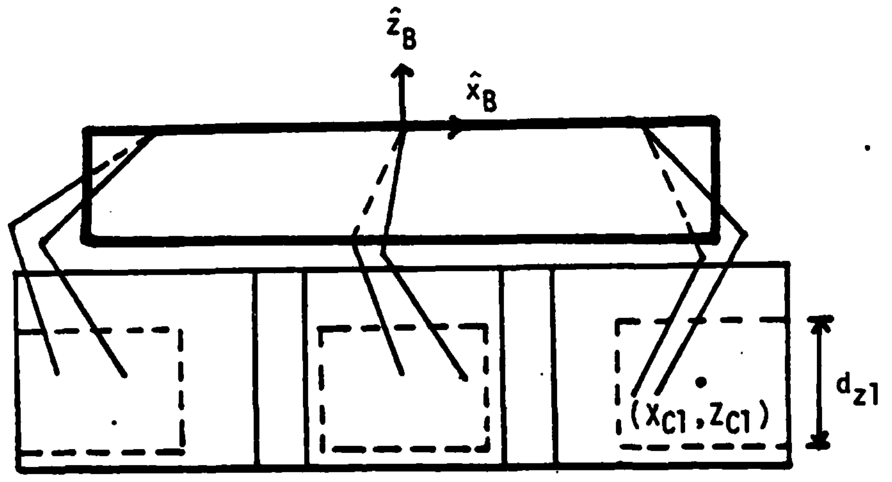
AD-A148 174 AN EXPERIMENTAL STUDY OF AN ULTRA-MOBILE VEHICLE FOR OFF-ROAD TRANSPORTATION(U) OHIO STATE UNIV RESEARCH FOUNDATION COLUMBUS R B MCGHEE ET AL. SEP 84 2/3

UNCLASSIFIED MDA903-82-K-0058 F/G 6/4 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6



(a) Top View



(b) Side View

Figure 4.2. Working Volumes (Solid Lines) and Constrained Working Volumes (Dashed Lines) for Each Leg.

4.3 Optimal Period

In the previous algorithm [8], the locomotion period for periodic gaits was determined by the ratio of an appropriate stride length to a particular body velocity component. Since this approach does not explicitly consider the kinematic limits of a leg, there was no way to ensure that the foot position of each leg during locomotion would be within its working volume. In fact, there were times when a leg of the OSU Hexapod reached its limit.

However, if the body motion were constrained to be, in most part, in one direction, then the stride length could be set as needed for that motion so as to keep the legs within limits. This led to three different modes of operation for the OSU Hexapod for forward/backward movement (Cruise), lateral movement (Side-stepping), and for turning (Turn-in-place). The purpose of the present work is to consider the kinematic limits of the legs more directly so that the body motion need not be artificially constrained. The result is that the three modes will be integrated into one which provides the basis for omnidirectional control of the ASV.

In order to find the optimal period based on the constrained working volume, the following definitions are made.

Definition 9: The <u>temporal kinematic margin</u> of leg i, t_{Si}, is a predicted time-to-go until the foot-tip of leg i in the support phase reaches the surface of the constrained working volume with the present foot velocity.

By the above definition,

$$t_{Si} = \frac{d}{|\underline{v}_i|} \tag{4.1}$$

where $\underline{v}_i$ is the present foot velocity of leg i and d is the distance from the support point to the surface of the constrained working volume in the direction of $\underline{v}_i$.

Definition 10: The maximum instantaneous support period for leg i, τ_{Si}, is a predicted maximum time for leg i to be within its constrained working volume while in the support phase.

It is noted from Eq. (2.5) that the period is proportional to the ratio of incremental time change to incremental phase change. For each leg during the support phase, the value of the support phase variable, ϕ_{Si}, which is defined in Eq. (2.10), changes from 0 for touchdown to 1 for lift-off. Then the maximum instantaneous support period for each leg may be computed through the use of the temporal kinematic margin, t_{Si}, and the support phase variable, ϕ_{Si}, as follows:

$$\tau_{Si} = \frac{t_{Si}}{1 - \phi_{Si}} . \tag{4.2}$$

A leg which has the minimum value of the maximum instantaneous support period is called a "critical leg", and the boundary surface of the constrained working volume which the critical leg will reach at the end of the support phase is called a "critical edge". To keep each leg inside its constrained working volume, the optimal support period, τ_S, should be taken from the critical leg. That is,

$$\tau_S = \min_i (\tau_{Si}) \qquad \text{for } i = 1,..,6. \tag{4.3}$$

Then the magnitude of the optimal period, τ, is given by

$$|\tau| = \frac{\tau_S}{\beta} \quad . \tag{4.4}$$

As discussed in Chapter 2, the sign of the period, τ, determines whether a forward or backward wave gait is implemented. Both gaits implement the same sequence of leg liftings and placings, however, in opposite order. While optimal wave gaits are the only ones used in this work, there is still the possibility of using a forward or backward sequence.

The forward sequence (which gives a forward wave action of the legs as referenced to the front of the vehicle) has been shown to be optimally stable for straight-line forward locomotion and will be used when the longitudinal component of velocity, v_x, is positive (forward movement). The backward sequence (actually forward when referenced to the rear of the vehicle) will be used when v_x is negative. While there is no guarantee that either of these sequences is optimal for arbitrary body motion, use of these should generally enhance stability.

There is one final consideration in changing from one sequence to the other. The gait change from the forward wave gait to the backward wave gait, or vice versa, should occur only when the magnitude of the body translational velocity is small. Otherwise, the sudden sign change of the period during high speed locomotion may lead to a leg going out of its constrained working volume. This can be more clearly understood through an example. If a leg is about to liftoff, the sign change of the period causes that leg to appear as if it has just touched down. This means that the time that this leg spends on the ground increases. For high body velocity, the leg will quickly reach the limits of its constrained working volume. The problem is due to the fact that the leg motion depends on both the locomotion period and the body velocity, while the body motion depends only on the body velocity.

The above considerations lead to the graphical representation of the gait change in the v_x-v_y velocity plane as shown in Figure 4.3, where v_y is the lateral velocity component. The outside of the circle represents the high speed region where a gait change should not occur. There are two regions inside the circle. The band region where no gait change occurs provides hysteresis in order to avoid quick gait changes for nominally zero forward velocity. The values of a and b are chosen experimentally as a=0.05 ft/sec and b=0.2 ft/sec in this work.

4.4 Deceleration Plan

The speed of leg return in the transfer phase is limited by the capabilities of the joint servo motors. Hence, there exists a minimally required transfer time, called the transfer time threshold, for the leg to move from lift-off to touchdown. For legged vehicle locomotion, the commanded transfer time of each leg must be greater than the transfer time threshold.

When the optimal period is obtained as described in the previous section, it is necessary to check whether it satisfies the condition imposed by the transfer time threshold. If not, the vehicle speed must be reduced.

The foot velocity in the support phase is linearly related to the body velocity components (note Eq. (5.51) in Chapter 5). From Eqs. (4.1) through (4.3),

$$\frac{1}{|\tau|} = \sum_{i=1}^{6} C_i |\underline{v}_i| \tag{4.5}$$

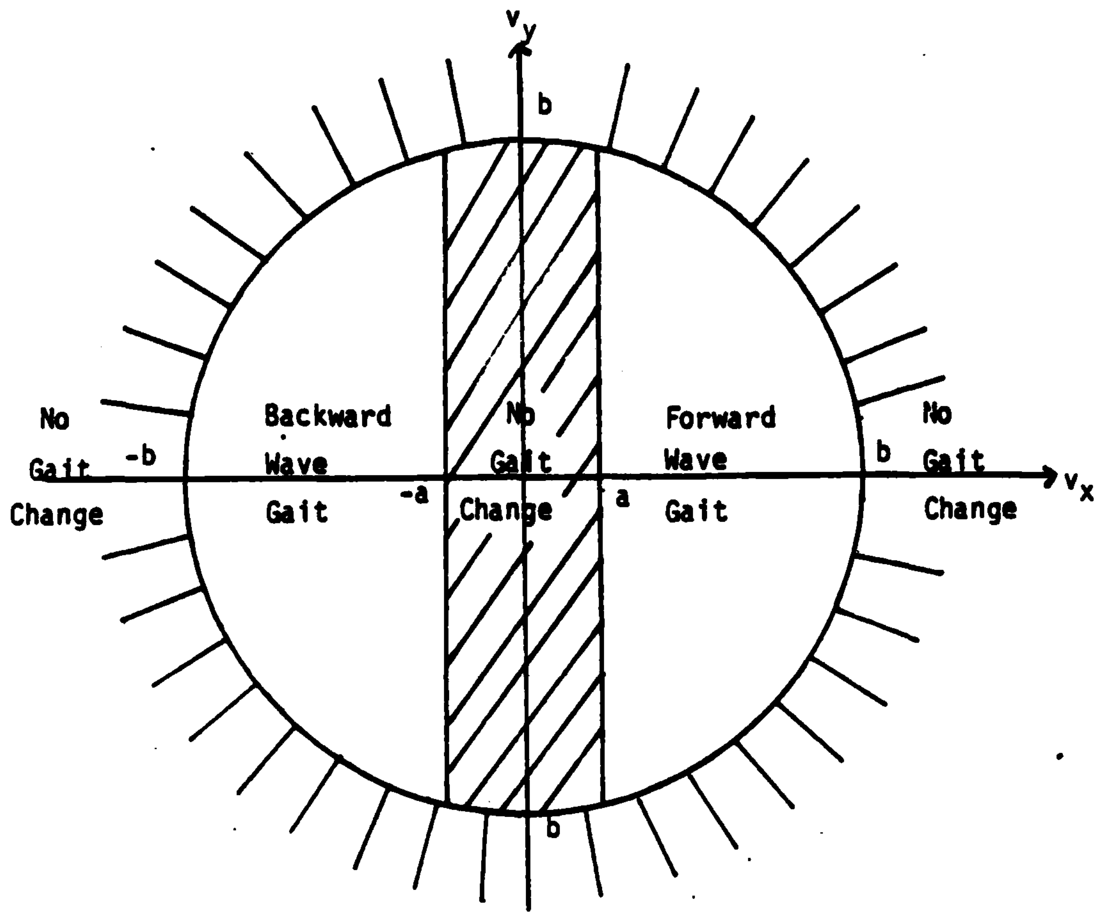


Figure 4.3. Diagram for Gait Change as a Function of Body Translational Velocity.

where C_i is a proportionality factor for leg i which is constant for a specific foot position, leg phase, and duty factor. Also, the transfer time, τ_R, is related to τ such as

$$|\tau_R| = (1 - \beta)\ |\tau|. \tag{4.6}$$

Let the transfer time threshold be τ_{Rth}. If τ_R is less than τ_{Rth}, then it is necessary to slow down the body velocity components to limit the transfer time to be τ_{Rth}. Let the ratio, μ, of the absolute values of τ_R and τ_{Rth} be

$$\mu = \frac{|\tau_R|}{|\tau_{Rth}|}\ . \tag{4.7}$$

Using the relation of τ_R and τ given in Eq. (4.6), the period after slowdown, τ', is given by

$$|\tau'| = \frac{|\tau_{Rth}|}{1 - \beta} \tag{4.8}$$

From Eqs. (4.6) through (4.8), the following is obtained:

$$\frac{1}{|\tau'|} = \frac{\mu}{|\tau|}\ . \tag{4.9}$$

Substituting Eq. (4.5) into Eq. (4.9) yields

$$\frac{1}{|\tau'|} = \sum_{i=1}^{6} C_i\,|\mu \underline{v}_i|\ . \tag{4.10}$$

Again, from the linear relationship between the foot velocity in the support phase and the body velocity, μ may be a slowdown factor for the body velocity.

Thus, whenever the transfer time of the optimal period is less than the transfer time threshold, the slowdown factor is computed and the six body velocity components are reduced accordingly. The value of the transfer threshold time is chosen to be 0.5 sec in this simulation work.

4.5 Summary

In this chapter, the algorithm for integration of the three different joystick control modes, which were previously defined for the OSU Hexapod, has been described. This integration makes it possible to implement the Close Maneuvering Mode of operation for the ASV which provides for omnidirectional control.

The integration is based on the concept of the constrained working volume which provides soft limits for leg motion in order to regulate leg stepping so as to maintain static stability for the vehicle. Since the dimensions of the constrained working volume may be easily changed in the simulation, it is possible to examine changes in stability for different dimensions of the constrained working volume, and this is discussed in Chapter 5.

The optimal period was defined in such a way that at least one of the six legs fully utilizes its constrained working volume, while the other legs are maintained inside these limits. Thus, the optimal period ensures that each leg moves within its constrained working volume during locomotion over uneven terrain. Also, in this chapter a simple algorithm has been developed to choose between forward and backward wave gaits, and hopefully it will enhance stability for arbitrary body motion.

Finally, a deceleration plan has been presented to slow down the vehicle speed in order to satisfy the minimum transfer time capability of the legs. The slowdown factor was defined based on the notion that the inverse of the optimal period is linearly related to the magnitude of the body velocity. Thus, when the transfer time obtained from the optimal period is shorter than the transfer time threshold, each of six velocity components is reduced by the slowdown factor.

The flow chart for the algorithm developed in this chapter is given in Figure 4.4. The algorithm provides an essential means to implement the Close Maneuvering Mode of operation for the ASV which provides for omnidirectional control over uneven terrain. Actual control algorithms for locomotion over uneven terrain will be discussed in Chapter 5.

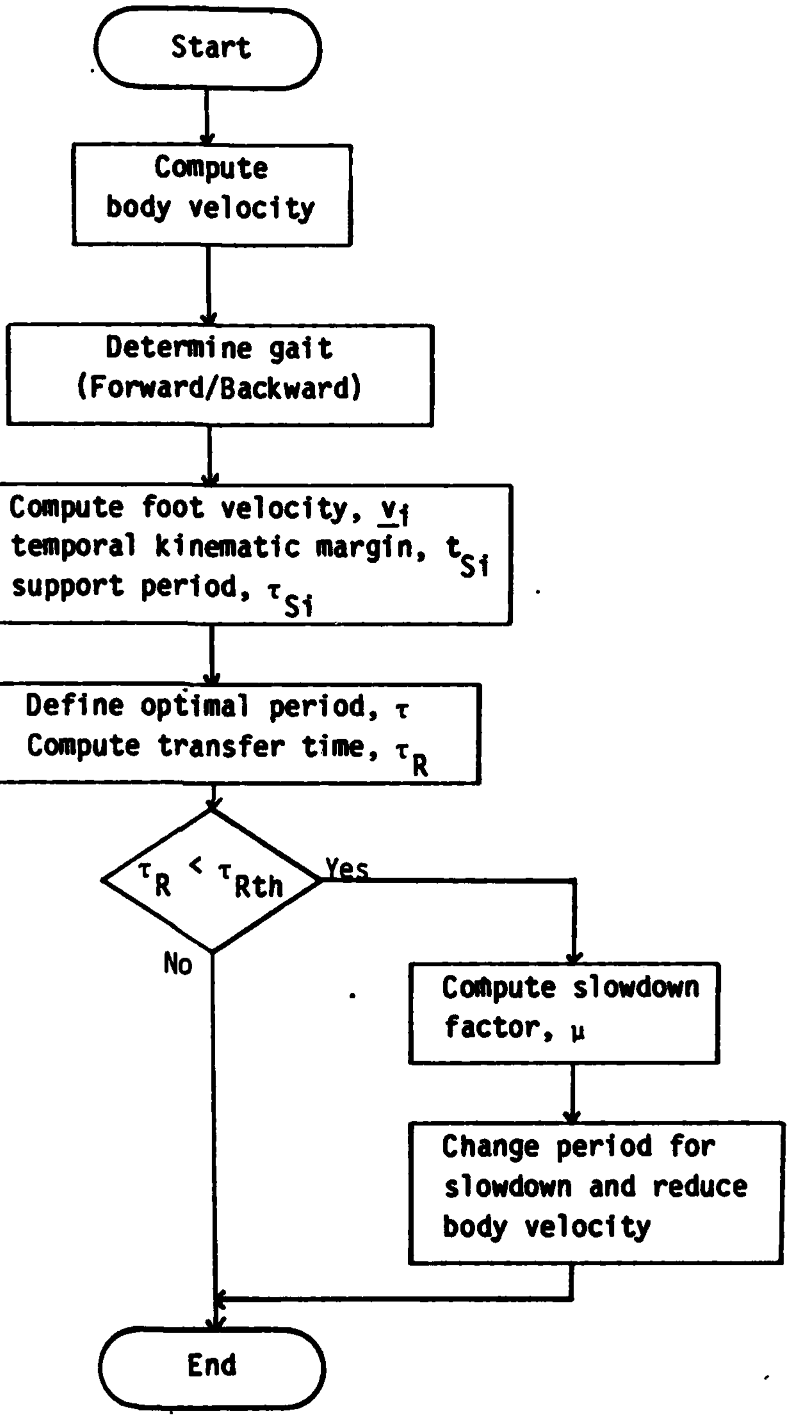

Start
Compute body velocity
Determine gait (Forward/Backward)
Compute foot velocity, $\underline{v}_i$ temporal kinematic margin, t_{Si} support period, τ_{Si}
Define optimal period, τ Compute transfer time, τ_R
$\tau_R < \tau_{Rth}$
Yes
No
Compute slowdown factor, μ
Change period for slowdown and reduce body velocity
End

Figure 4.4. Flow Chart for the Algorithm to Provide Omnidirectional Control.

Chapter 5

MOTION PLANNING ALGORITHMS FOR UNEVEN TERRAIN

5.1 Introduction

In this chapter, control algorithms for the Close Maneuvering Mode of operation, which has been described in Section 2.3, over uneven terrain are presented by employing the theoretical results for the gaits, the vehicle/terrain models, and the algorithms for omnidirectional control developed in the previous chapters. The control algorithms developed in the previous research have mainly focused on locomotion over flat level terrain [8], or some efforts have been made in developing control algorithms for uneven-terrain locomotion [3,9,11,12]. Based on the previous work, the complete control algorithms for locomotion over uneven terrain are developed and discussed in this chapter.

The terrain models used in this work are assumed that they are free of holes or large obstacles, because the Close Maneuvering Mode of operation is suitable for reasonably rough terrain. These terrain models are referred to as "Type-0" terrain by Hirose et al. [6]. They may be simulated by the prismatic terrain models described in Section 3.2.

The control algorithms developed are based on the supervisory control scheme described in Section 2.4. According to the basic hierarchical structure of the supervisory control scheme as shown in

Figure 2.3, there are two functional blocks: the motion planning and execution blocks. Since the motion execution block is replaced by the vehicle/terrain simulation block in this work (see Chapter 6), only the motion planning algorithms are discussed in this chapter.

The motion planning algorithms control the desired body and leg trajectories with feedback of contact and proximity information from the leg subsystems. In the computer simulation of this work, proximity sensors are simulated in software which are assumed to measure the distance from the foot-tip to the support surface in the direction of the gravity vector. The software simulation of a proximity sensor is discussed in detail in Section 5.2. Also, contact sensors are simulated in the same way as proximity sensors.

Body motion planning algorithms, which are described in Section 5.3, generate the body position and velocity set-points using the operator's velocity commands and information about the local terrain surface. The local terrain surface is estimated by using six points of estimation based on contact and proximity information from the six legs through linear regression [3]. Based on the estimated plane of support, algorithms for body attitude and altitude control are developed in this section.

Leg motion planning algorithms are designed to generate the foot trajectory for the support and transfer phases and to choose the desired foothold for the transfer phase based on the body velocity and the constrained working volumes for the legs. Furthermore, proximity sensors are used for the transfer trajectory in order to avoid foot contact with terrain surface. Discussions about leg trajectory generation are made in Section 5.4.

An algorithm to increase the stability of the vehicle over sloped terrain by dynamically changing the positions of the constrained working volumes for the middle legs according to the slope angle is presented in Section 5.5. Since the size of the constrained working volume of each leg also affects the stability margin, discussions about the change of the size of the constrained working volume according to the vehicle speed are made in this section.

5.2 Proximity Sensors

Recently, the use of sensing technology to endow a robot with a greater degree of "intelligence" in dealing with its environment is receiving increased attention. Moreover, for the multilegged robot vehicle, the information about the walking environment provided by the sensor systems is essential for adaptive locomotion over uneven terrain.

For omnidirectional movement in the Close Maneuvering Mode of operation for the ASV, full terrain preview is not provided since the optical radar system used is limited to scanning in the front of the vehicle. On the other hand, proximity sensors will be used to good advantage to sense the local terrain conditions under each foot. Hopefully, these will allow each leg to adapt to the terrain. In the simulation work of this dissertation, it is assumed that proximity sensors are available and these will be integrated into the control algorithms in order to facilitate leg placement onto the terrain surface and to avoid foot contact with terrain surface during the transfer phase.

Broerman recently surveyed the proximity sensing techniques and implemented a proximity sensor system based on ultrasonic ranging for the OSU Hexapod [38]. The proximity sensors were attached to the lower limb segment of each leg in order to measure the foot altitude and to approximately control the position and velocity of each leg while in its transfer phase of motion, especially during foot descent.

In the computer simulation of this work, each leg of the ASV is simulated in software to be equipped with a proximity sensor, which is assumed to measure the distance from the foot-tip to the support surface in the direction of the gravity vector. While it may be true that most proximity sensors would not necessarily provide this kind of data, it will be used in this work since the exact form of the data will vary significantly with the physical sensor implemented.

From the terrain database obtained through interactive generation of the prismatic terrain model as described in Section 3.2, it is known that the plane equation of each rectangular surface is

$$AX + BY + CZ + D = 0. \tag{5.1}$$

Let the foot position of leg i in the earth-fixed coordinate system be

$${}^{E}\underline{p}_{Fi} = [X_{Fi} \quad Y_{Fi} \quad Z_{Fi}]^{T}, \tag{5.2}$$

and let a point which is a vertical projection of the foot-tip of leg i onto the rectangular plane along the $-\hat{z}_E$ direction be

$${}^{E}\underline{p}_{Pi} = [X_{Pi} \quad Y_{Pi} \quad Z_{Pi}]^{T}. \tag{5.3}$$

Then, by the assumption made on the direction of proximity sensing,

$$X_{Pi} = X_{Fi}, \tag{5.4}$$

$$Y_{Pi} = Y_{Fi}, \tag{5.5}$$

and, from Eq. (5.1),

$$Z_{Pi} = -(AX_{Pi} + BY_{Pi} + D)/C. \tag{5.6}$$

The output of each proximity sensor is the altitude of the foot of leg i above the support surface, F_{Ai}, and is given by

$$F_{Ai} = Z_{Fi} - Z_{Pi}. \tag{5.7}$$

Figure 5.1 illustrates the method for simulating each proximity sensor.

5.3 Body Motion Planning

Body motion planning algorithms specify the position and orientation of the vehicle body which varies according to the operator's velocity commands and the terrain conditions. Many of the algorithms developed in past work have been applicable for the case of flat level terrain [8] and, 2-D planar techniques were applied.

In this section, development of the algorithms is broken into three parts. First, the technique used to estimate the local terrain surface is presented. Second, the coordinate systems used to specify the motion of the body in three dimensions are defined. Finally, the body regulation plan to automatically control the altitude and attitude is described.

5.3.1 Estimation of the Support Plane

In [12], the number of points of estimation is limited to the number of legs which are presently in contact with the ground. For the hexapod vehicle, thus, the number of legs in the support phase is between three and six during locomotion, which often gives only a limited set of present terrain information. However, through the use of the proximity sensors, the number of points of estimation becomes equal to the total number of legs of the vehicle.

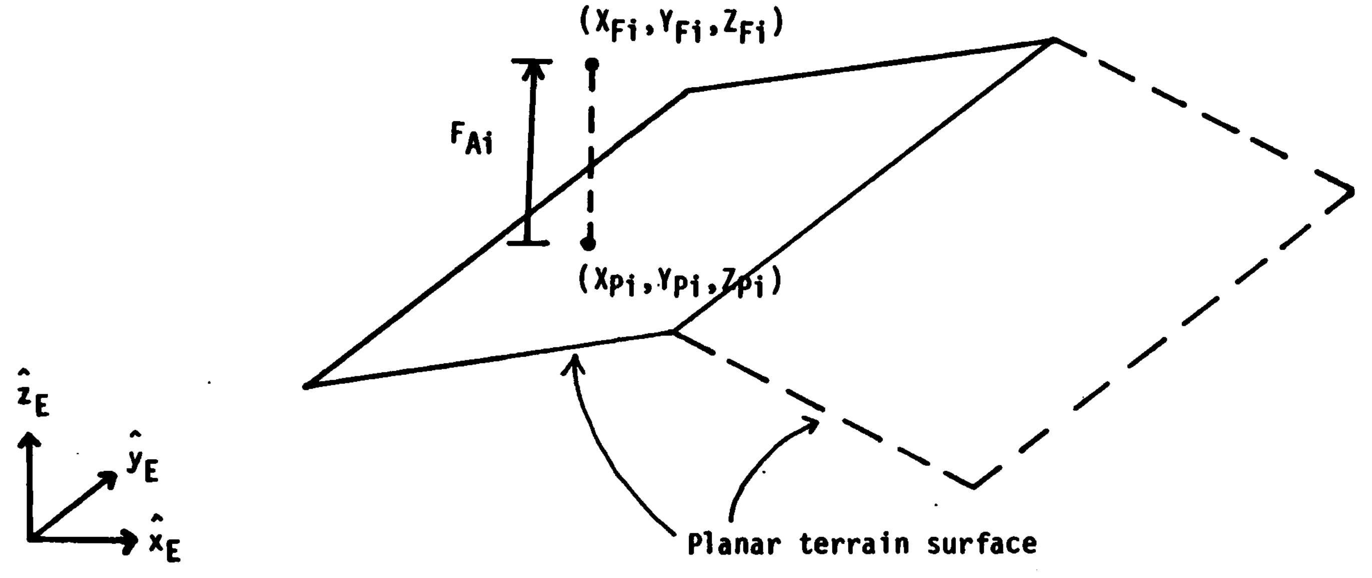
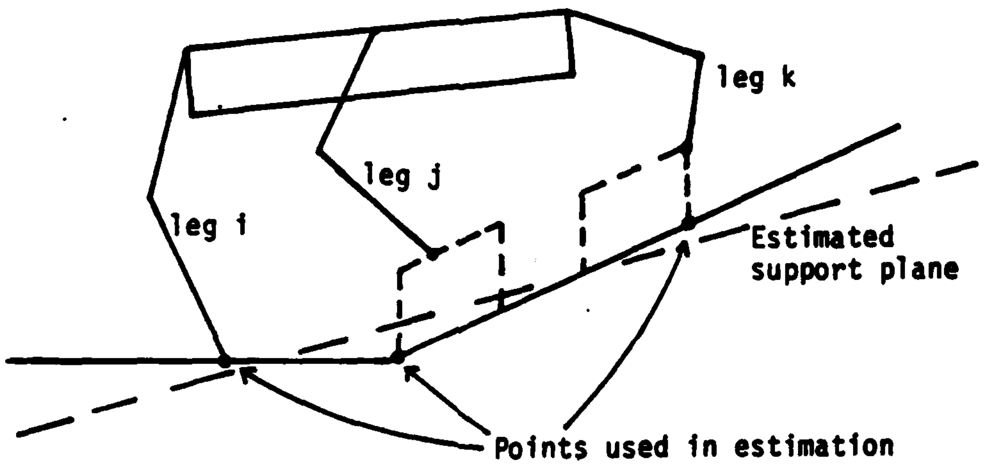


Figure 5.1. Simulation of the Proximity Sensor for the ASV.

There are two cases to be considered to obtain the six points of estimation for a hexapod vehicle. The first case is for the legs in the support phase. In this case the present support points of each leg may be directly used in estimation. The second case is for the legs in the transfer phase. The transfer trajectory is divided into three subphases: lift-off, transfer forward, and placement, which will be discussed in detail in Section 5.4. When a leg is in the placement subphase, the leg proximity sensor detects its approximate touchdown point on the ground. This touchdown point is used in the estimation. It may be noted that this scheme may increase terrain adaptability by controlling the body altitude and attitude before the leg steps onto the ground. When the transfer leg is in other than the placement subphase, its last support point is used for estimation. Figure 5.2 illustrates the above scheme.

The use of six points for estimation is not completely arbitrary and some further comments may serve to explain this choice. As proposed in [12], the number of points of estimation may be more than six by including the support points of previous locomotion cycles. However, the adaptability problem is confined to the local terrain conditions in which the vehicle is presently traversing. In other words, after the vehicle has already passed over a certain point of the terrain, the terrain information around that point is only marginally useful for the present vehicle motion. Instead, the terrain information ahead of the vehicle is more useful. That is why the vision system [26] which provides terrain preview is essential for operational modes other than Close Maneuvering. Since terrain preview is not provided in the Close Maneuvering Mode, terrain information may be obtained from the support



leg i in support phase
leg j in transfer phase (other than placement subphase)
leg k in placement phase

Figure 5.2. Selection of Points for Each Leg Used in Estimating the Support Plane.

points and the estimated touchdown points as measured by the proximity sensors.

The estimation technique employed is that of linear regression as described in [3]. The only change made in the equations given in [3] is that the number of points of estimation is fixed to six as previously described.

The foot positions are usually expressed in the body coordinate system and may be transformed to earth coordinates through the homogeneous transformation given in Eq. (3.2). Using the appropriate support points or measured touchdown points, the estimated support plane may be obtained in the earth-fixed coordinate system through linear regression as

$$AX + BY + CZ + D = 0 \tag{5.8}$$

where $[A\ B\ C]^T$ is a unit vector which points outward in the normal direction from the plane. The estimated plane is situated at a distance of -D along the normal vector from the origin of the earth-fixed coordinate system.

While the legs are in the support phase, the support points do not change with respect to the earth-fixed coordinate system. The support plane, thus, needs to be estimated only when at least one of the six legs begins the placement subphase at which time the proximity sensor can measure the touchdown point (next estimated support point).

5.3.2 Coordinate Systems

The body trajectory is determined not only by the operator's velocity commands but also by the terrain conditions imposed on the motion of the body. In order to describe the body motion over uneven terrain, it is useful to define three coordinate systems: the earth-fixed coordinate system, the body-fixed coordinate system, and the terrain coordinate system. The earth and body coordinate systems have been defined in Chapter 3. In this subsection, the terrain coordinate system and the homogeneous transformation matrices between the coordinate systems are defined.

Since only the local terrain conditions affect the vehicle motion and are changing while the vehicle is moving, it is convenient to define the terrain coordinate system, which characterizes the local terrain conditions, as a moving frame with respect to the earth-fixed coordinate system. Furthermore, the terrain coordinate system is defined with respect to the estimated support plane discussed in the previous section.

The origin of the terrain coordinate system ($\hat{x}_T, \hat{y}_T, \hat{z}_T$) is located at the projection of the body center of gravity, along the gravity vector ($-\hat{z}_E$ direction), and the estimated support plane. The unit vector, $\hat{z}_T$, is directed normal to the estimated support plane and its components in the earth-fixed coordinate system are given by Eq. (5.8):

$$^{E}\hat{z}_T = [A \;\; B \;\; C]^T. \tag{5.9}$$

The unit vector, $\hat{x}_T$, is defined to be in the direction of the rotation axis from $\hat{z}_E$ to $\hat{z}_T$; thus,

$$^{E}\hat{x}_T = \frac{\hat{z}_E \times \hat{z}_T}{|\hat{z}_E \times \hat{z}_T|} = [-\frac{B}{m} \;\; \frac{A}{m} \;\; 0]^T, \tag{5.10}$$

where $m = \sqrt{A^2 + B^2}$. Finally, the unit vector, $\hat{y}_T$, is defined to give a right-hand set of unit vectors. With this definition of the terrain coordinate system, $\hat{y}_T$ is always directed up the slope. Furthermore, the slope angle is given as:

$$\eta = \cos^{-1}(\hat{z}_E \cdot \hat{z}_T). \tag{5.11}$$

If $\eta = 0$, then $\hat{x}_T$, $\hat{y}_T$, and $\hat{z}_T$ are forced to be parallel to $\hat{x}_E$, $\hat{y}_E$, and $\hat{z}_E$, respectively. The coordinate systems assigned are shown in Figure 5.3. Note that the terrain coordinate system changes as the body moves along its trajectory.

The homogeneous transformation matrix H from body coordinates to earth coordinates has been defined in Eqs. (3.2) and (3.3). Let the homogeneous transformation matrix from the terrain coordinates to earth coordinates be T. It has the following form:

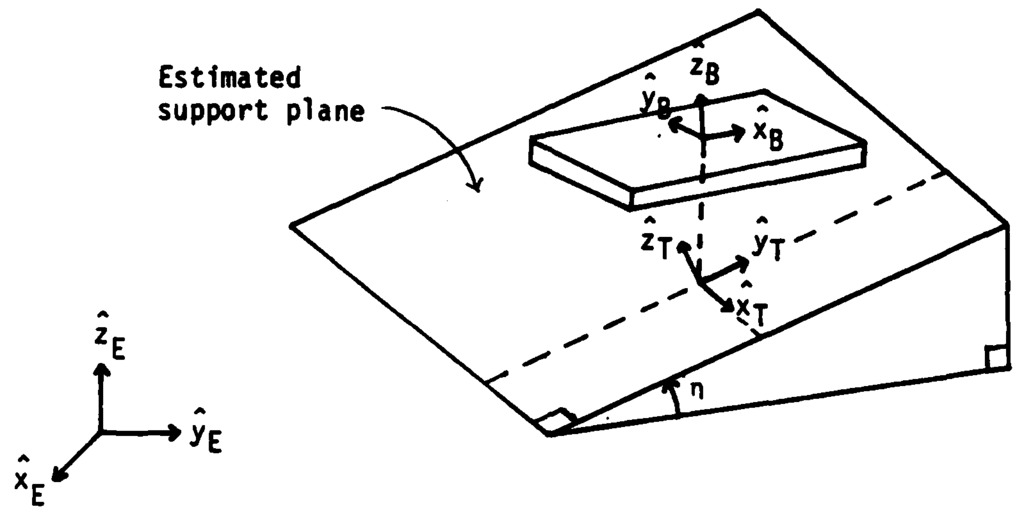
$$T = \left[\begin{array}{ccc|c} & T_r & & {}^E\underline{p}_T \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{5.12}$$

where T_r is a 3X3 orientation matrix and ${}^E\underline{p}_T$ is a 3X1 position vector given as:

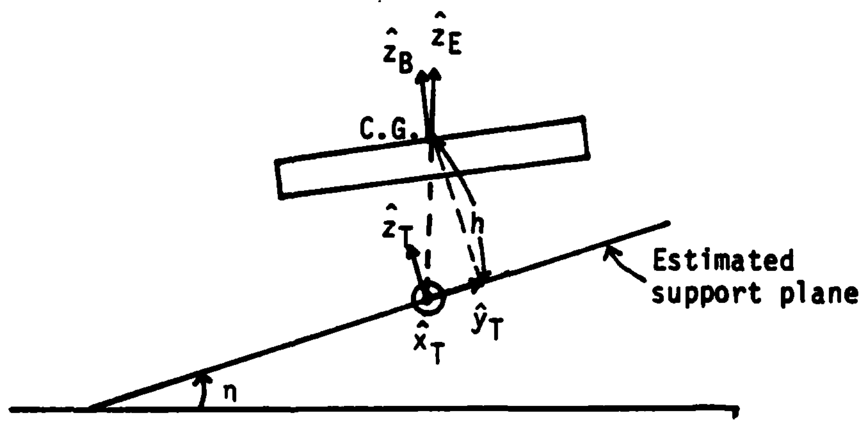
$${}^E\underline{p}_T = [P_x \;\; P_y \;\; P_z]^T. \tag{5.13}$$

From the foregoing discussion concerning the definition of the terrain coordinate system, it may be determined that P_x and P_y are equal to the x and y components of the position vector ${}^E\underline{p}_H$ in H, respectively. By substituting P_x and Py into Eq. (5.8), P_z is obtained as

$$P_z = -(AP_x + BP_y + D) / C. \tag{5.14}$$



(a) Coordinate Systems in 3-D Space



(b) Coordinate Systems in the Y_T-Z_T Plane

Figure 5.3. Definition of the Terrain Coordinate System.

The components of the orientation matrix T_r are determined by the rotational relationship between the earth and terrain coordinate systems as shown in Figure 5.4. The angle, α, between $\hat{x}_E$ and $\hat{x}_T$ is derived from Eq. (5.10) such that

$$\alpha = \tan^{-1}\left(\frac{A}{-B} \right). \tag{5.15}$$

Then, T_r is obtained by successive rotations about the z and x axes [36]:

$$T_r = \mathrm{Rot}(z,\alpha)\mathrm{Rot}(x,\eta) \tag{5.16}$$

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha\cos\eta & \sin\alpha\sin\eta \\ \sin\alpha & \cos\alpha\cos\eta & -\cos\alpha\sin\eta \\ 0 & \sin\eta & \cos\eta \end{bmatrix} . \tag{5.17}$$

Let the homogeneous transformation matrix from the body coordinates to terrain coordinates be G. Then G is simply given by

$$G = T^{-1} H. \tag{5.18}$$

The three coordinate systems and the three transformation matrices defined in this section will be utilized to define the body and leg trajectories in 3-D space as discussed in the following sections.

5.3.3 Body Regulation Plan

In previous work for the OSU Hexapod, automatic body regulation algorithms have been developed to control the body attitude and altitude to maintain the body level using force and inertial sensors [11] or to keep the body parallel to and at a constant height above the estimated support plane using contact information from the force sensors [12]. An algorithm to be presented in this section deals with a more general body regulation plan in which the body is not always parallel to the

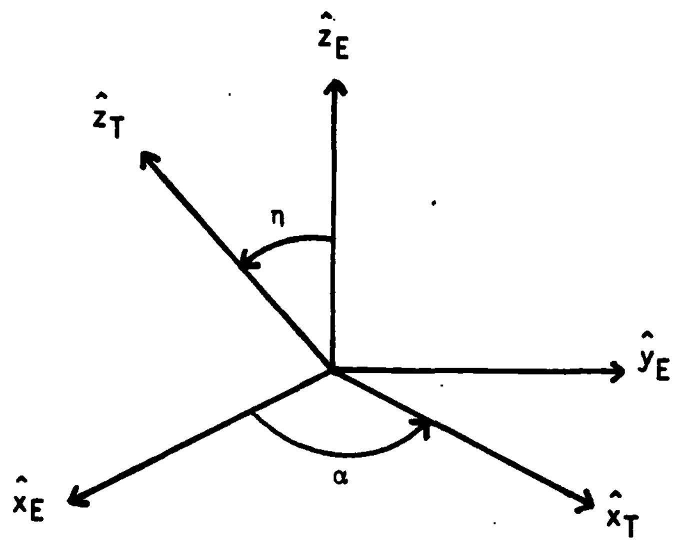


Figure 5.4. Rotational Relationship between the Earth and Terrain Coordinate Systems.

estimated plane and the body height is variable. In this way, this algorithm may allow more flexibility in body attitude and altitude control so that more adaptability to terrain irregularities can be achieved.

The body velocity for full six-dgree-of-freedom motion consists of the translational velocity, $\underline{v}$, and the rotational velocity, $\underline{\omega}$, (with components expressed in the body coordinate system):

$$\underline{v} = [v_x \; v_y \; v_z]^T \tag{5.19}$$

and

$$\underline{\omega} = [\omega_x \; \omega_y \; \omega_z]^T, \tag{5.20}$$

where v_x is the longitudinal speed; v_y is the lateral speed; v_z is the vertical speed; ω_x is the roll rate; ω_y is pitch rate; ω_z is the yaw rate. In the supervisory control scheme, v_x, v_y, and ω_z are provided by the human operator and v_z, ω_x, and ω_y are the velocity components to be actively controlled by the control software to achieve the desired body attitude and altitude according to the body regulation plan.

The coefficients in the equation for the estimated support plane (Eq. (5.8)) may be expressed as a row vector:

$$E\underline{\varphi} = [A \; B \; C \; D]. \tag{5.21}$$

Note that the coefficients are written with respect to the earth-fixed coordinate system. Then, $E\underline{\varphi}$ may be simply transformed into body coordinates [36] by postmultiplying by the appropriate homogeneous transform:

$$B\underline{\varphi} = E\underline{\varphi} \cdot H = [A' \; B' \; C' \; D']. \tag{5.22}$$

The components of the unit normal vector of the estimated support plane in the body coordinate system are

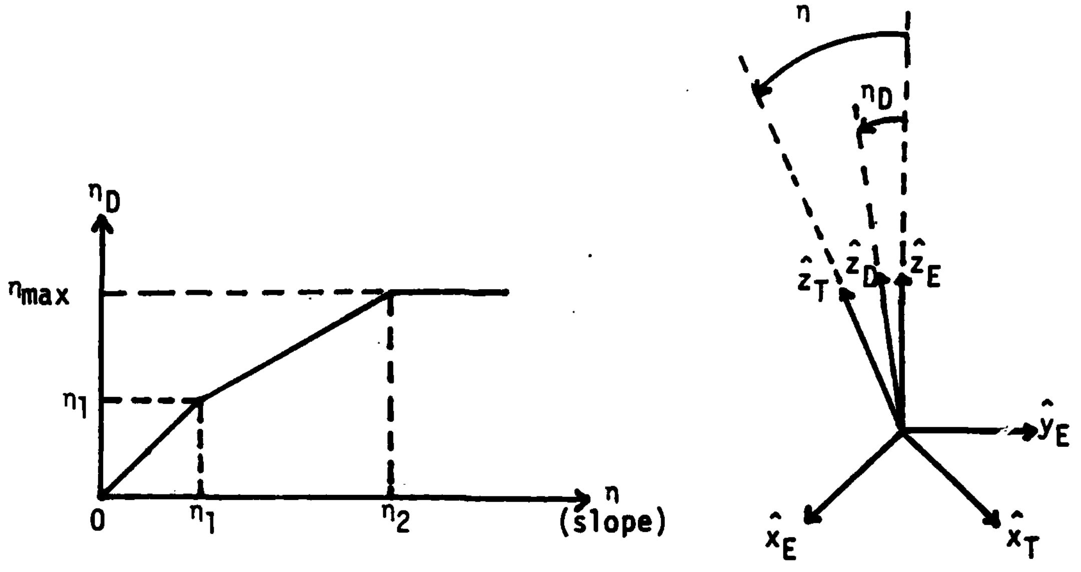
$$^{B}\hat{z}_T = [A' \; B' \; C']. \tag{5.23}$$

Note also that -D' is the distance from the center of gravity of the body along $^{B}\hat{z}_T$ to the estimated support plane.

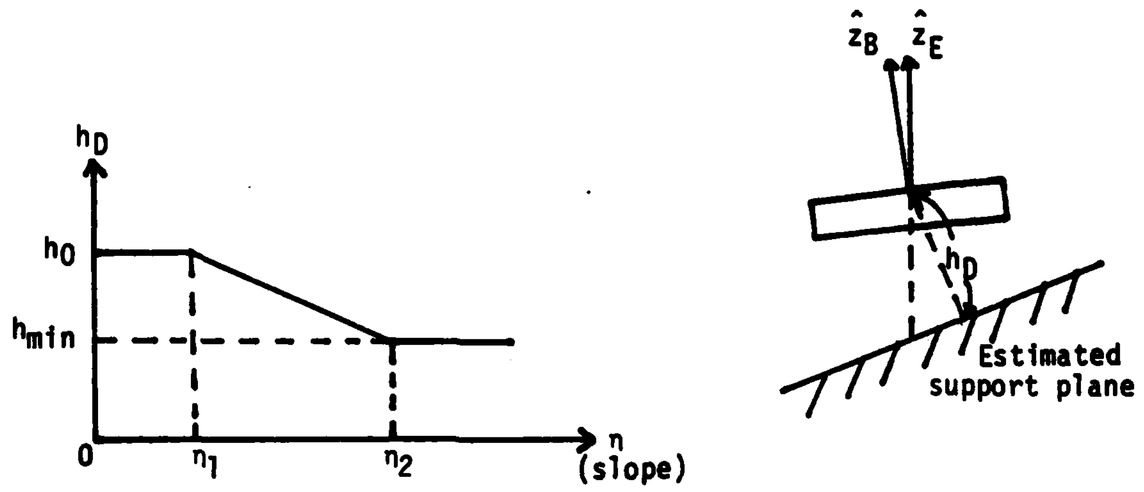
For body attitude control, it is necessary to define the desired body angle, η_D, which is measured from the $\hat{z}_E$ axis on the plane formed by the two unit vectors $\hat{z}_E$ and $\hat{z}_T$. The angle η_D is positive for counterclockwise rotation about the $\hat{x}_T$ axis. Body altitude control is accomplished by changing the distance from the center of gravity to the estimated support plane in the $\hat{z}_T$ direction.

Figure 5.5 shows the body attitude and altitude regulation plan adopted in this simulation work. The body attitude and altitude are related to the slope angle η through piece-wise linear curves. The quantities η_1 and η_2 are the threshold angles of the slope and h_0 is the nominal body height during locomotion over flat level terrain. The quantities η_{max} and h_{min} are the body angle and height, respectively, when the slope angle is larger than η_2. All these values may be set by the operator before the vehicle begins to move. The piece-wise linear curves are expressed as:

$$\eta_D = \begin{cases} \eta & \text{for } 0 < \eta < \eta_1 \\ \dfrac{\eta_{max} - \eta_1}{\eta_2 - \eta_1}(\eta - \eta_1) + \eta_1 & \text{for } \eta_1 < \eta < \eta_2 \\ \eta_{max} & \text{for } \eta_2 < \eta \end{cases} \tag{5.24}$$



(a) Attitude Regulation



(b) Altitude Regulation

Figure 5.5. Body Regulation Plan for Sloped Terrain.

and

$$h_D = \begin{cases} h_0 & \text{for } 0 < \eta < \eta_1 \\ -\dfrac{h_0 - h_{min}}{\eta_2 - \eta_1}(\eta - \eta_1) + h_0 & \text{for } \eta_1 < \eta < \eta_2 \\ h_{min} & \text{for } \eta_2 < \eta . \end{cases} \qquad (5.25)$$

This plan is designed to keep the "belly" of the vehicle down to the ground to increase stability over sloped terrain and to incline the body toward the slope to avoid the excessive tilting of the body as the slope becomes stiffer. There are only two threshold angles, η_1 and η_2, in this plan. However, if desirable in future work, the number of the threshold angles may be rather simply increased to more than two.

From the above regulation plan, the desired unit normal vector of the body plane, $\hat{z}_D$, is defined in Figure 5.5. Since $\hat{z}_D$ lies on the Z_E-Z_T plane, $\hat{z}_D$ may be obtained by rotating $\hat{z}_E$ about $\hat{x}_T$ by the angle η_D. By using the formula of a rotation about an arbitrary axis given in [36] along with Eq. (5.10), the components of $\hat{z}_D$ in the earth coordinates are obtained as:

$$^E\hat{z}_D = [\ \frac{A}{m}\sin\eta_D \quad \frac{B}{m}\sin\eta_D \quad \cos\eta_D\ ]^T. \qquad (5.26)$$

For attitude control, it is necessary to transform $^E\hat{z}_D$ into body coordinates and these are given as follows:

$$^B\hat{z}_D = H_r^{-1}\ ^E\hat{z}_D = [a\ b\ c]^T. \qquad (5.27)$$

Then, the rotation axis from the unit normal vector of the present body plane, $\hat{z}_B$, to $\hat{z}_D$, as shown in Figure 5.6, is given by

$$^B\hat{k} = \frac{\hat{z}_B \times \hat{z}_D}{|\hat{z}_B \times \hat{z}_D|} \tag{5.28}$$

$$= [\ -\frac{b}{\ell} \quad \frac{a}{\ell} \quad 0\ ]^T \tag{5.29}$$

$$= [\ k_x \quad k_y \quad k_z\ ]^T, \tag{5.30}$$

where $\ell = \sqrt{a^2 + b^2}$. The rotation angle, γ, is

$$\gamma = \cos^{-1}(\hat{z}_B \cdot \hat{z}_D) = \cos^{-1}(c). \tag{5.31}$$

Therefore, attitude control may be done by rotating $\hat{z}_B$ about the rotation axis $\hat{k}$ by γ in an appropriate manner. For a physical system, it is not possible to move instantaneously by the angle γ. In order to achieve a smooth transition, the following control law, which gives an exponential response, is defined such that

$$\dot{\gamma} = -K_A\ \gamma, \tag{5.32}$$

where K_A is the inverse of the time constant for the motion . From Eqs. (5.30) and (5.32), the rotational vector $\dot{\gamma}\ \hat{k}$ may be decomposed into two components in the body coordinate system:

$$\dot{\gamma}\ \hat{k} = -K_A\ k_x\ \gamma\ \hat{x}_B\ -\ K_A\ k_y\ \gamma\ \hat{y}_B. \tag{5.33}$$

The x and y components may be related to the roll rate, ω_x, and the pitch rate, ω_y, respectively, with the following result:

$$\omega_x = -K_A\ k_x\ \gamma \tag{5.34}$$

$$\omega_y = -K_A\ k_y\ \gamma. \tag{5.35}$$

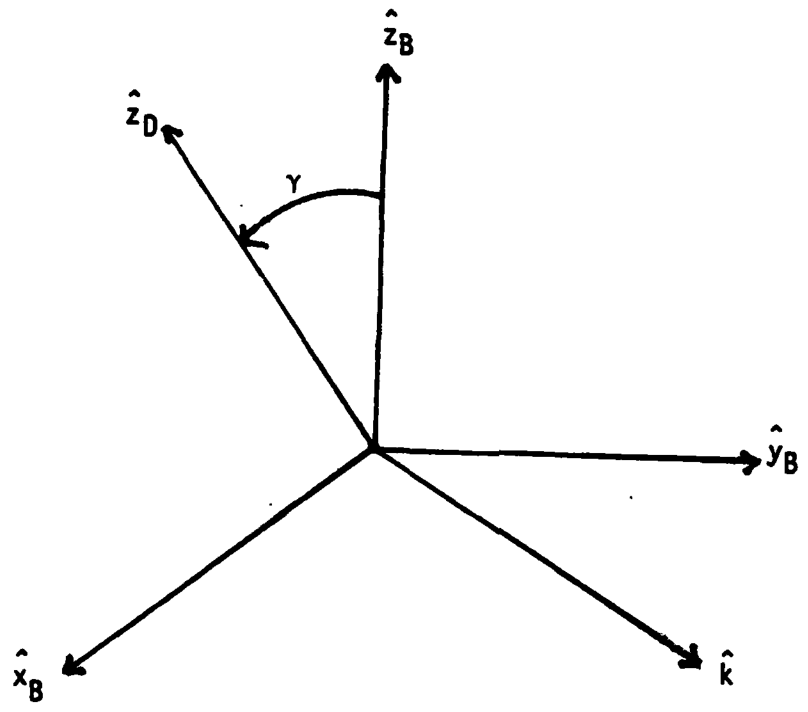


Figure 5.6. Angular Displacement between the Desired and Present Body Unit Normal Vectors.

For altitude control, the present body height, h, is obtained from Eq. (5.22):

$$h = -D'. \tag{5.36}$$

Also, the desired body height, h_D, is given in Eq. (5.25). In the same manner as above, the control law for altitude is defined as

$$v_z = -K_A(h_D - h). \tag{5.37}$$

Note that the above altitude control law is only valid under the assumption that the angle between the body and the estimated support plane is small.

The above discussion about the automatic body regulation algorithm may be summarized in a flow chart. That is shown in Figure 5.7.

In real-time operation, the control program is executed in an incremental time, Δt, which is called the computation cycle time. The six body velocity components computed as above cause a differential change in H by Δt. The differential change in H, dH, is defined as follows [36]:

$$dH = H\, H_\Delta , \tag{5.38}$$

where H_Δ, which is called the differential translation and rotation transform, is given as:

$$H_\Delta = \begin{bmatrix} 0 & -\delta_z & \delta_y & dx \\ \delta_z & 0 & -\delta_x & dy \\ -\delta_y & \delta_x & 0 & dz \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{5.39}$$

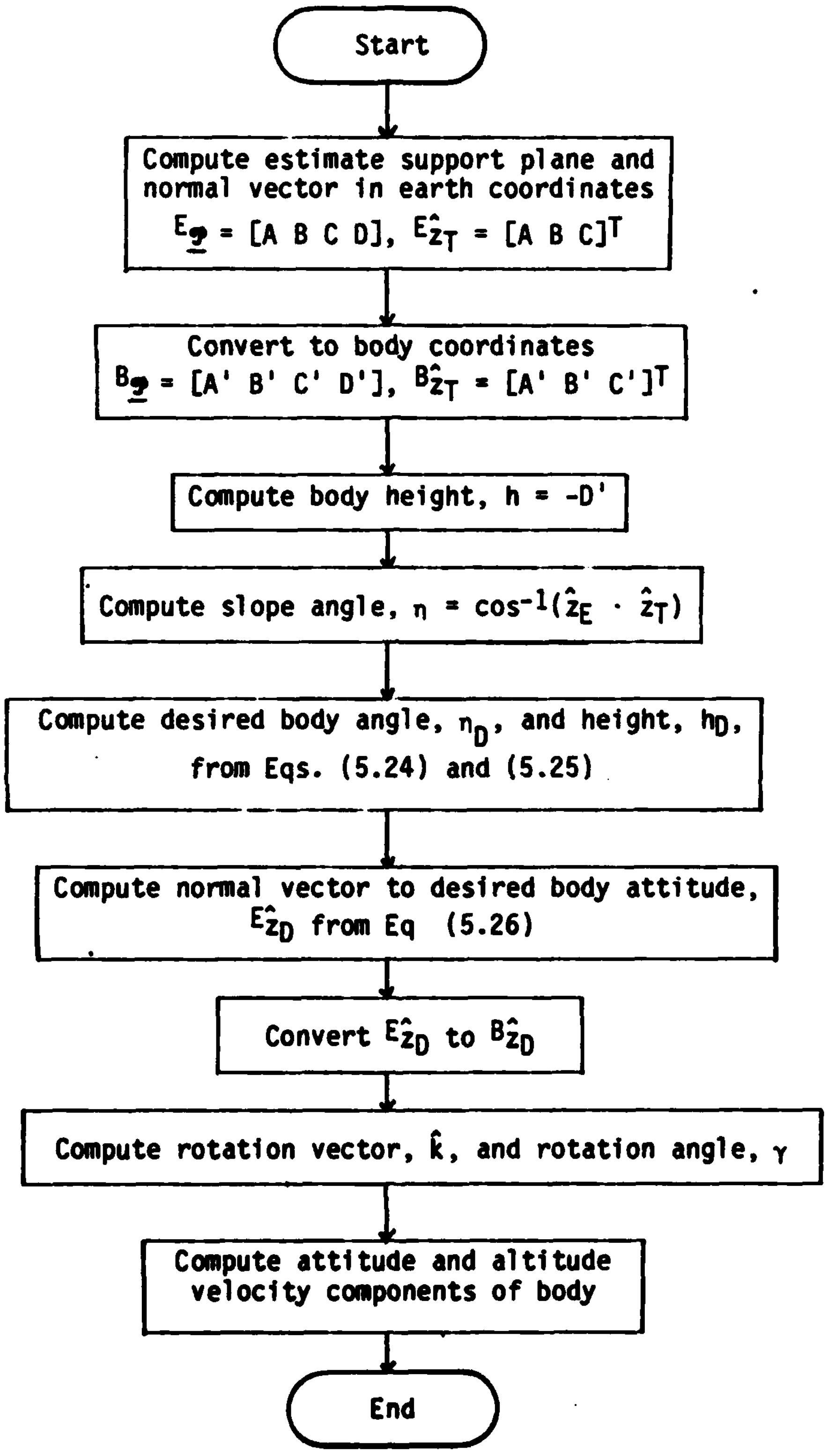

Start
Compute estimate support plane and normal vector in earth coordinates
${}^{E}\underline{\varphi} = [A\ B\ C\ D]$, ${}^{E}\hat{z}_T = [A\ B\ C]^T$
Convert to body coordinates
${}^{B}\underline{\varphi} = [A'\ B'\ C'\ D']$, ${}^{B}\hat{z}_T = [A'\ B'\ C']^T$
Compute body height, $h = -D'$
Compute slope angle, $\eta = \cos^{-1}(\hat{z}_E \cdot \hat{z}_T)$
Compute desired body angle, η_D, and height, h_D, from Eqs. (5.24) and (5.25)
Compute normal vector to desired body attitude, ${}^{E}\hat{z}_D$ from Eq (5.26)
Convert ${}^{E}\hat{z}_D$ to ${}^{B}\hat{z}_D$
Compute rotation vector, $\hat{k}$, and rotation angle, γ
Compute attitude and altitude velocity components of body
End

Figure 5.7. Flow Chart for the Body Regulation Algorithm.

and

$$\begin{aligned} dx &\cong v_x \cdot \Delta t \\ dy &\cong v_y \cdot \Delta t \\ dz &\cong v_z \cdot \Delta t \\ \delta_x &\cong \omega_x \cdot \Delta t \\ \delta_y &\cong \omega_y \cdot \Delta t \\ \delta_z &\cong \omega_z \cdot \Delta t. \end{aligned} \tag{5.40}$$

Thus, H is updated each computation cycle as

$$H \leftarrow H + dH. \tag{5.41}$$

However, Eq. (5.41) may not guarantee orthogonality of the orientation part (H_r) of the updated H matrix due to the differential change. In order to orthogonalize the updated H matrix, the Gram-Schmit orthogonalization process [39] was employed.

5.4 Leg Motion Planning

Leg motion planning algorithms are designed to generate the foot trajectory according to the change of the kinematic cycle phase and to choose the desired foothold for the transfer leg. The change of kinematic cycle phase and the foothold selection mainly depend on the body velocity and the interaction of the support plane and constrained working volumes for the legs.

From Eq. (2.2), the kinematic cycle phase, ϕ, may be updated every computation cycle time, Δt, as follows:

$$\phi \leftarrow [\phi + \frac{\Delta t}{\tau}]_{\text{mod } 1}. \tag{5.42}$$

Then the leg phase variable of leg i, ϕ_{Li}, is updated through use of Eq. (2.6). The updated kinematic cycle phase and leg phase variables determine the the state of leg i, support phase or transfer phase, and synchronize the leg stepping events in order to implement wave gaits.

In this simulation work, the proximity sensor is used to specify the touchdown point of the transfer leg which is exactly the contact point on the terrain surface. On real terrain, however, the touchdown point measured by the proximity sensor may not actually be on the ground due to measurement errors. To ensure foot contact, contact sensors (often implemented by force sensors) should be used at the end of transfer phase.

The following subsections discuss the problems related to foot trajectory generation over uneven terrain in 3-D space. The reference position which is useful in selecting the foothold, is first defined to be on the center line of the constrained working volume. The support trajectory is then computed based on the foothold and body velocity. Finally, the transfer trajectory is defined, again based on the foothold and also the slope of the terrain.

5.4.1 Reference Position

In order to plan the trajectory of a supporting foot, a nominal foot position, near the center of the constrained working volume through which the foot trajectory is predicted to pass, is defined. This point is called the reference position. The reference position is also used to determine the desired foothold and this will be discussed in Section 5.4.3.1.

For locomotion over flat level terrain with the body parallel to the support plane, the reference position is chosen to be located at the center of the constrained working volume, a point which is on the support plane as shown in Figure 5.8a. In this case, the nominal body height, h_0, is equal to the z-component of the center of the constrained working volume, Z_{Ci}, in body coordinates. By planning the support trajectory to pass through the center of the constrained working volume, each leg may more fully utilize its constrained working volume while in the support phase.

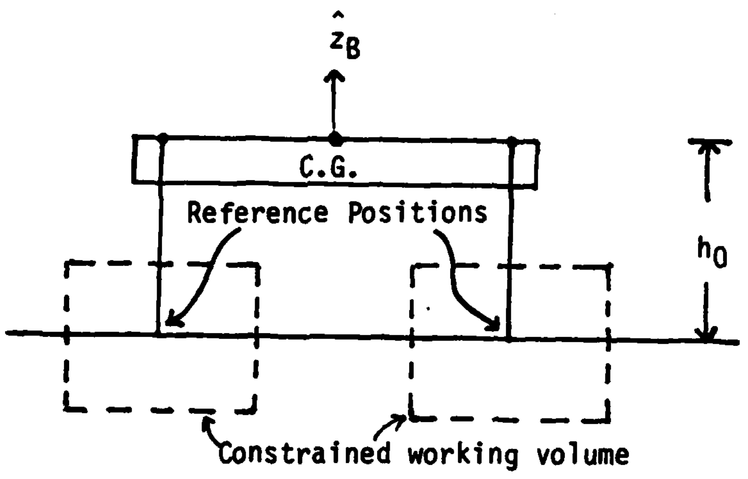
For locomotion over sloped terrain as shown in Figure 5.8b, the body may not always be parallel to the support plane. Also, the center of the constrained working volume may not be located on the support plane. Therefore, in order to include the more general case for locomotion over uneven terrain, the reference position of a leg is defined as a point inside its constrained working volume which is at the intersection of the estimated support plane and the "vertical" line which passes through the center of the constrained working volume in the $\hat{z}_B$ direction.

The reference position for a leg, R, as shown in Figure 5.9 may be computed in the following manner. The point C denotes the center of the constrained working volume, and R is at the intersection of the center line and the estimated support plane. Let the position vector of C in body coordinates, ${}^B\underline{p}_C$, be

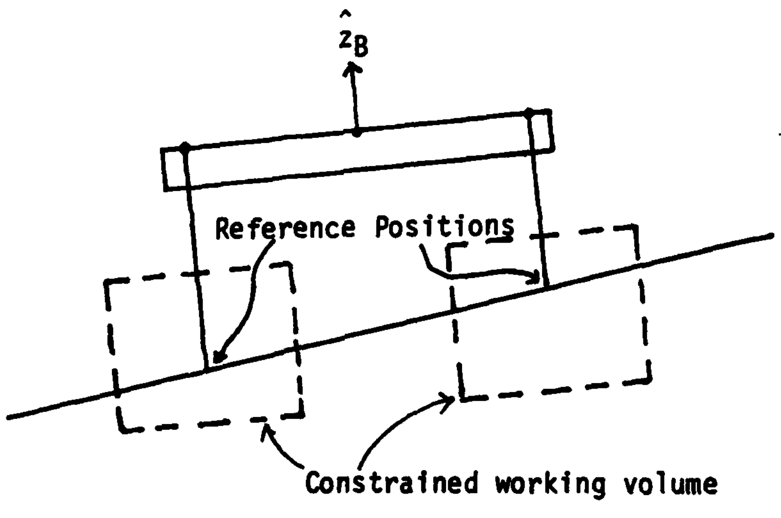
$${}^B\underline{p}_C = [X_C \; Y_C \; Z_C]^T. \tag{5.43}$$

The quantities X_C, Y_C, and Z_C are given in Section 4.2. Also, let the position vector of R in body coordinates, ${}^B\underline{p}_R$, be

$${}^B\underline{p}_R = [X_R \; Y_R \; Z_R]^T. \tag{5.44}$$



(a) Foot Reference Positions over Flat Level Terrain



(b) Foot Reference Positions over Sloped Terrain

Figure 5.8. Foot Reference Positions over Level and Sloped Terrain.

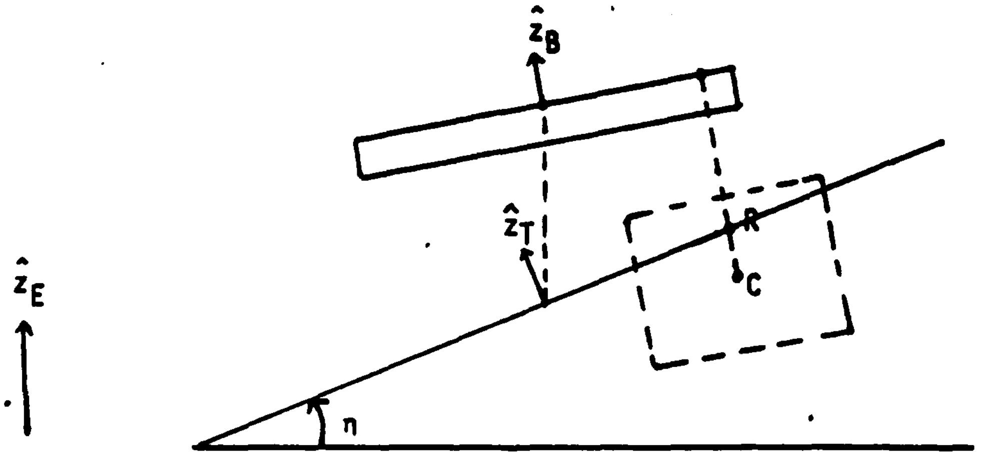


Figure 5.9. Computation of a Foot Reference Position over Sloped terrain.

From the foregoing discussion concerning the definition of the reference position, X_R and Y_R are given as follows:

$$X_R = X_C \tag{5.45}$$

$$Y_R = Y_C \ . \tag{5.46}$$

From Eq. (5.22), the equation for the estimated support plane in body coordinates is

$$A'X + B'Y + C'Z + D' = 0. \tag{5.47}$$

Thus, by substituting Eqs. (5.45) and (5.46) into Eq. (5.47), Z_R is obtained as

$$Z_R = -(A'X_R + B'Y_R + D')/C'. \tag{5.48}$$

5.4.2 Support Phase

In the support phase, the basic problem is to determine the updated values of the positions and rates of each supporting foot in the body-fixed coordinate system as the body moves through the desired trajectory. Since the foot positions in earth coordinates are assumed to be constant during the support phase, these known values may be transformed to body coordinates through the continuously updated H matrix as given in [12]:

$$^B\underline{p}_{Fi} = \begin{bmatrix} ^Bx_{Fi} \\ ^By_{Fi} \\ ^Bz_{Fi} \end{bmatrix} = H^{-1} \begin{bmatrix} ^Ex_{Fi} \\ ^Ey_{Fi} \\ ^Ez_{Fi} \end{bmatrix} \ . \tag{5.49}$$

The approach to determining the foot rates is different from that in [12]. Using Eqs. (5.19) and (5.20), by elementary physics, the foot velocity of leg i relative to the body motion, $\underline{v}_i$, is given by

$$\underline{v}_i = - \underline{v} - \underline{\omega} \times {}^B\underline{p}_{Fi}. \tag{5.50}$$

In matrix form, Eq. (5.50) is

$$\begin{bmatrix} \dot{X}_{Fi} \\ \dot{Y}_{Fi} \\ \dot{Z}_{Fi} \end{bmatrix} = - \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} - \begin{bmatrix} -\omega_z Y_{Fi} + \omega_y Z_{Fi} \\ \omega_z X_{Fi} - \omega_x Z_{Fi} \\ -\omega_y X_{Fi} + \omega_x Y_{Fi} \end{bmatrix}. \tag{5.51}$$

5.4.3 Transfer Phase

In the transfer phase, the problems to be addressed include those of foothold selection and the determination of the positions and rates of the foot of leg. The transfer trajectory is defined based on the foothold and terrain variations, using the transfer phase variable of each leg, ϕ_{Ti}, which has been defined in Section 2.2.2.

5.4.3.1 Desired Foothold

The desired foothold of a transfer leg is selected at a point on the surface of the constrained working volume such that its support trajectory is predicted to pass through the foot reference position. For computational purposes, it is assumed that the foot velocity for the support trajectory would be equal to the foot velocity at the reference position for the present body velocity.

While the reference position is valuable in selecting a foothold as well as defining the support trajectory, it should be noted that the actual support trajectory may not pass through the reference position. This results because of changes in the body velocity from that used in computing the foothold. Also, even with constant body velocity, the trajectory will usually not be a straight line as assumed. Figure 5.10 illustrates the above in 3-D space.

In general, for constant body velocity, the actual support trajectory will be an arc of a circle as shown in Figure 5.10. In this work, the desired foothold, which is the intersection point of the foot trajectory with the constrained working volume, is computed by approximating the circular foot trajectory with a straight line. This will simplify the computation considerably.

The desired foothold is computed as follows. First, by using Eqs. (5.44) and (5.51), the predicted foot velocity, $\underline{v}_R$, at the reference position with the present body velocity is obtained:

$$\underline{v}_R = [\dot{X}_R \; \dot{Y}_R \; \dot{Z}_R]^T. \tag{5.52}$$

Then by taking a point, P_0, outside the constrained working volume in the $-\underline{v}_R$ direction and connecting it to the reference position, P_R, with a straight line, there is an intersection point, P_{int}, on the edge of the constrained working volume as shown in Figure 5.11a. The components (X_0,Y_0,Z_0) and $(X_{int},Y_{int},Z_{int})$ denote the position of P_0 and P_{int}, respectively, in body coordinates.

To simplify the computation, P_{int} is first computed in the X_B-Y_B plane. For example, P_{int} as shown in Figure 5.11b is given by

$$X_{int} = X_0 + \frac{Y_2 - Y_0}{Y_R - Y_0}(X_R - X_0) \tag{5.53}$$

$$Y_{int} = Y_2. \tag{5.54}$$

In the same manner, the intersection points for segments crossing other edges may be computed. This method is employed in computer

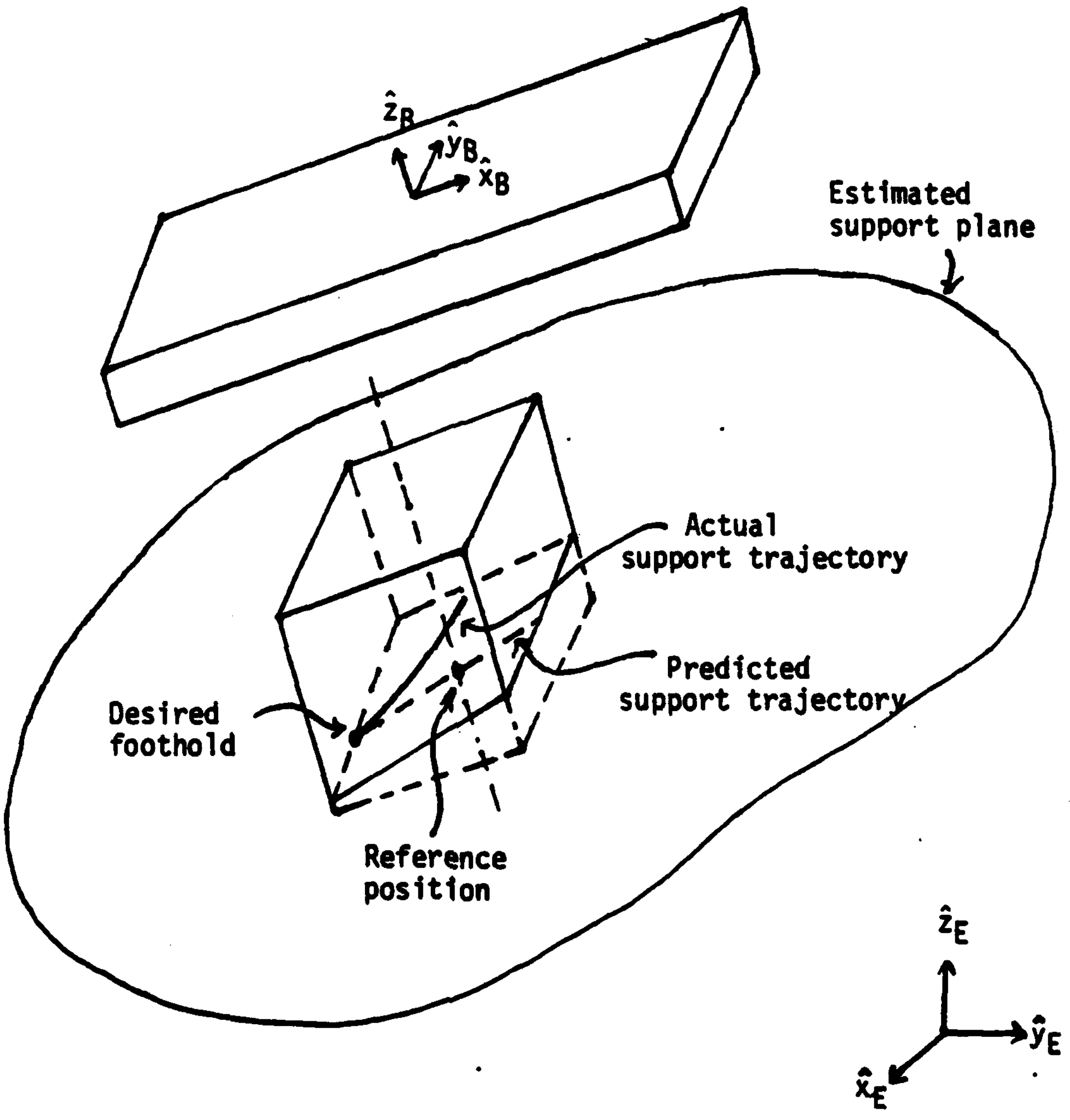
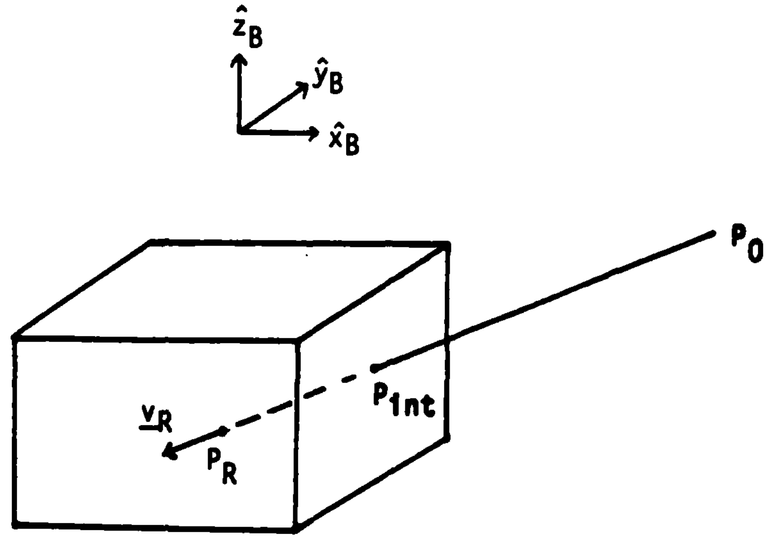
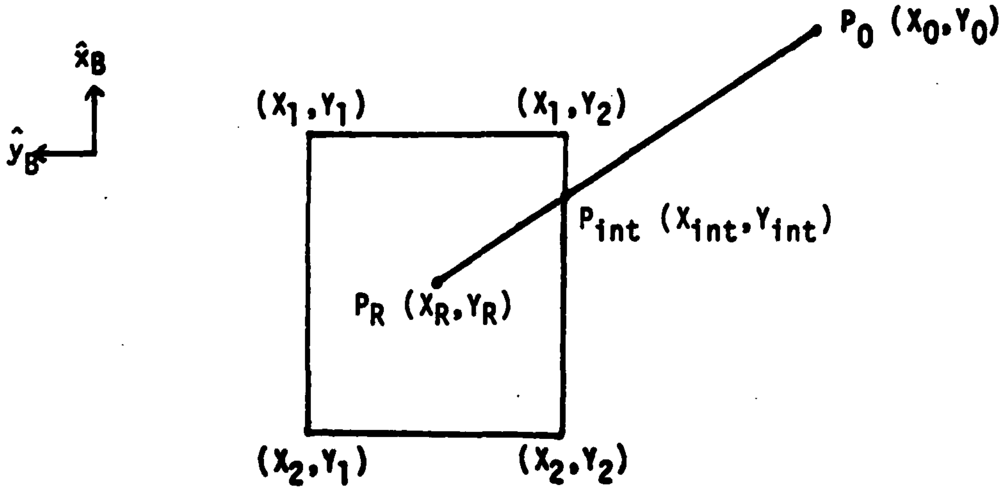


Figure 5.10. Relationship between the Predicted and Actual Support Trajectory.



(a)



(b)

Figure 5.11. Computation of the Desired Foothold.

graphics for clipping algorithms [39].

The values for X_{int} and Y_{int} given by Eqs. (5.53) and (5.54) may not be correct, because they are obtained only in the X_B-Y_B plane. In other words, they may be outside the constrained working volume in the $\hat{z}_B$ direction. In order to check this, the time, t_1, to go from P_R to P_{int} using X_{int} and Y_{int} of Eqs. (5.53) and (5.54) is computed as follows:

$$t_1 = \left|\frac{X_{int} - X_R}{\dot{X}_R}\right| = \left|\frac{Y_{int} - Y_R}{\dot{Y}_R}\right|. \tag{5.55}$$

Also, the time, t_2, to go from P_R to the edge of the constrained working volume in the $\hat{z}_B$ direction needs to be computed. The time t_2 is obtained from the z-dimension of the constrained working volume and is given as:

$$t_2 = \begin{cases} \left|\dfrac{Z_{max} - Z_R}{\dot{Z}_R}\right| & \text{for } \dot{Z}_R > 0 \\ \\ \left|\dfrac{Z_{min} - Z_R}{\dot{Z}_R}\right|, & \text{for } \dot{Z}_R < 0 \end{cases} \tag{5.56}$$

where $Z_{max} = Z_{Ci} + d_{zi}/2$ and $Z_{min} = Z_{Ci} - d_{zi}/2$ which are defined in Section 4.2. Thus, if $t_1 < t_2$, X_{int} and Y_{int} given by Eqs. (5.53) and (5.54) are correct and Z_{int} is simply

$$Z_{int} = Z_R - \dot{Z}_R \cdot t_1. \tag{5.57}$$

If $t_1 > t_2$, then

$$\begin{aligned} X_{int} &= X_R - \dot{X}_R \cdot t_2 \\ Y_{int} &= Y_R - \dot{Y}_R \cdot t_2 \\ Z_{int} &= Z_R - \dot{Z}_R \cdot t_2 \end{aligned} \tag{5.58}$$

Thus, the the components of the desired foothold of leg i, ${}^{B}\underline{p}_{Di}$, are:

$$ {}^{B}\underline{p}_{Di} = [X_{int} \ \ Y_{int} \ \ Z_{int}]^T \tag{5.59} $$

and, in earth coordinates,

$$ {}^{E}\underline{p}_{Di} = H \, {}^{B}\underline{p}_{Di}. \tag{5.60} $$

5.4.3.2 Foot Transfer Trajectory

The transfer trajectory is divided into three subphases: lift-off, transfer forward and placement. The trajectories for lift-off and placement may be generated by moving the foot straight up or down along the gravity vector from the lift-off point or to the desired foothold. Let the values of ϕ_{Ti} at the end of the lift-off subphase and at the beginning of the placement subphase be ϕ_L and ϕ_P, respectively, which are defined as constants. Then, the subphase to which leg i belongs is determined as follows:

$0 < \phi_{Ti} < \phi_L$	leg i in lift-off	(5.61)
$\phi_L < \phi_{Ti} < \phi_P$	leg i in transfer forward	(5.62)
$\phi_P < \phi_{Ti} < 1$	leg i in placement.	(5.63)

During the lift-off subphase, the total time left, T_{FLi}, until the foot reaches the foot lift height, h_F, which is preset to be a constant, is

$$ T_{FLi} = (\phi_L - \phi_{Ti})\tau_R, \tag{5.64} $$

where $\tau_R = (1 - \beta)\tau$, which is the total transfer time. In the same manner, the time left to the end of the transfer forward subphase, T_{FTi}, and the time left until foot placement during the placement subphase, T_{FPi}, are obtained as follows:

$$T_{FTi} = (\phi_P - \phi_{Ti})\tau_R \tag{5.65}$$

$$T_{FPi} = (1 - \phi_{Ti})\tau_R. \tag{5.66}$$

When leg i begins a new subphase, the foot position at the end of each subphase, ${}^E\underline{p}_{Ei}$, is computed as described in the following paragraphs. The movement from the present foot position to the end point is made to be linear over time in the earth-fixed coordinate system during each subphase. The following vector equation may be applied to all three subphases to update the foot position over an increment of time, Δt:

$${}^E\underline{p}_{Fi} \leftarrow {}^E\underline{p}_{Fi} + ({}^E\underline{p}_{Ei} - {}^E\underline{p}_{Fi})\Delta t/T_i, \tag{5.67}$$

where T_i is one of T_{FLi}, T_{FTi}, and T_{FPi}.

The end points of each subphase are computed in the following way. Let the foot position of leg i when $\phi_{Ti} = 0$ be

$${}^E\underline{p}_{Ti} = [X_{Ti} \quad Y_{Ti} \quad Z_{Ti}]^T. \tag{5.68}$$

Then, from Eq. (5.68), the end point for the lift-off subphase is

$${}^E\underline{p}_{Ei} = [X_{Ti} \quad Y_{Ti} \quad Z_{Ti} + h_F]^T. \tag{5.69}$$

Let the desired foothold obtained from Eq. (5.60) be

$${}^E\underline{p}_{Di} = [X_{Di} \quad Y_{Di} \quad Z_{Di}]^T, \tag{5.70}$$

then the end point for the transfer forward subphase is

$${}^E\underline{p}_{Ei} = [X_{Di} \quad Y_{Di} \quad Z_{Di} + h_F]^T. \tag{5.71}$$

However, in order to avoid foot contact with rough terrain during the transfer forward subphase, the proximity sensor is used to keep the foot altitude to be h_F above the terrain in the $\hat{z}_E$ direction. Therefore, the x and y components of ${}^E\underline{p}_{Fi}$, X_{Fi} and Y_{Fi}, may be updated using Eqs. (5.67) and (5.71), but the z component, Z_{Fi}, is updated using foot altitude information given by Eq. (5.7) as follows:

$$Z_{Fi} \leftarrow Z_{Fi} + (h_F - F_{Ai}). \tag{5.72}$$

During the placement subphase, the end point is measured by the proximity sensor and it is given by Eq. (5.3):

$${}^E\underline{p}_{Ei} = {}^E\underline{p}_{Pi}. \tag{5.73}$$

From Eq. (5.67), the foot velocity in earth coordinates is

$${}^E\underline{v}_i \cong ({}^E\underline{p}_{Ei} - {}^E\underline{p}_{Fi}) \,/\, T_i. \tag{5.74}$$

For the transfer forward subphase, the x and y components of ${}^E\underline{v}_i$, $\dot{X}_{Fi}$ and $\dot{Y}_{Fi}$, are obtained from Eq. (5.74), but the z component, $\dot{Z}_{Fi}$, is computed from Eq. (5.72) as

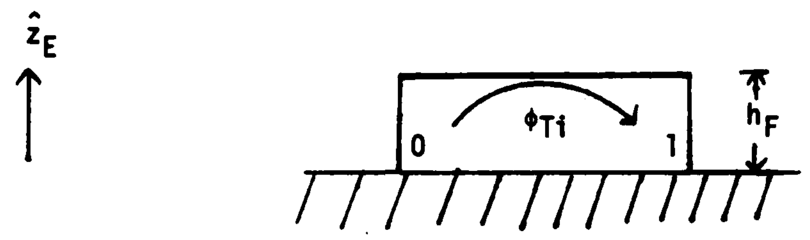
$$\dot{Z}_{Fi} \cong (h_F - F_{Ai}) \,/\, \Delta t. \tag{5.75}$$

The position and velocity vectors may be transformed into body coordinates as follows:

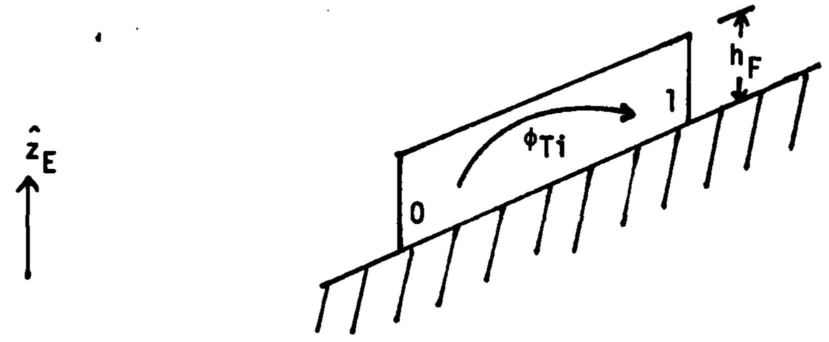
$${}^B\underline{p}_{Fi} = H^{-1} \; {}^E\underline{p}_{Fi} \tag{5.76}$$

$${}^B\underline{v}_i = H_r^{-1} \; {}^E\underline{v}_i \,. \tag{5.77}$$

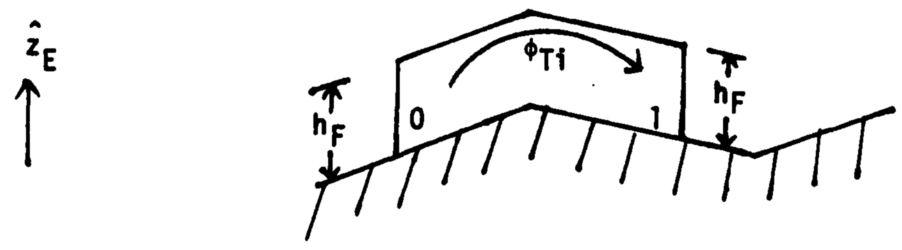
Figure 5.12 shows the transfer trajectory generated according to the algorithm as discussed above.



(a) Over Flat Level Terrain



(b) Over Sloped Terrain



(c) Over Rough Terrain

Figure 5.12. Foot Transfer Trajectory.

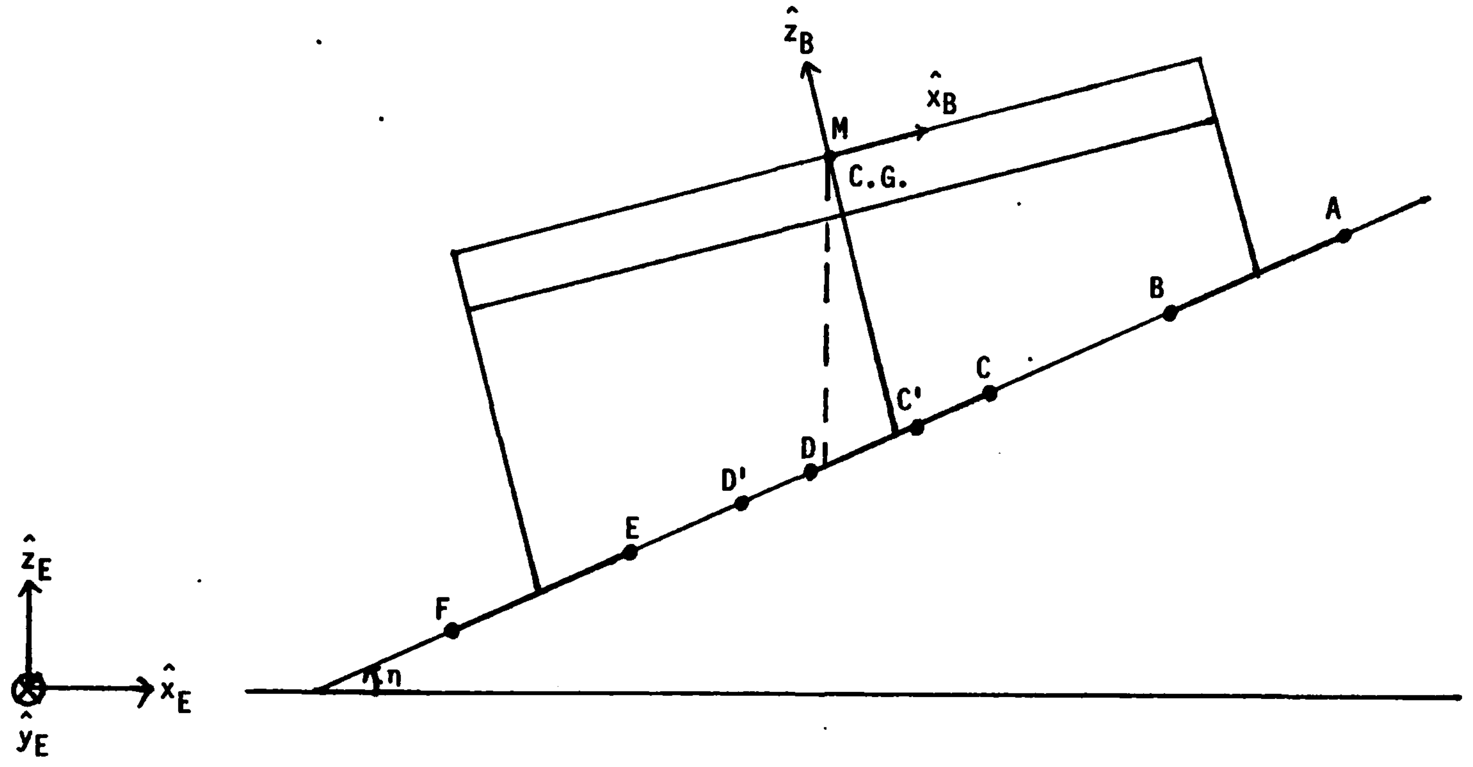
5.5 Adjustment of the Constrained Working Volume

The constrained working volume for each leg as discussed in Section 4.2, may be changed somewhat by changing its position and dimensions inside the original working volume. This flexibility may be utilized to increase the stability of the vehicle for locomotion over uneven terrain. Two schemes to increase stability are discussed in the following paragraphs.

In wave gaits for a hexapod vehicle, it may be observed that the stability margin mainly depends on the foot positions of the middle legs for straight-line locomotion, especially in the case of the tripod gait. This observation leads to a scheme to adjust the position of the constrained working volume of the middle legs when walking over sloped terrain.

For example, Figure 5.13 shows the change of the stability margin by moving the position of the constrained working volume of the middle leg, for straight-line locomotion with the tripod gait over sloped terrain. In Figure 5.13a, the cross sections of each constrained working volume with the slope are expressed as $\overline{AB}$ for the front leg, $\overline{CD}$ for the middle leg, and $\overline{EF}$ for the rear leg. Also, Figure 5.13b shows the vehicle on the X_E-Y_E plane and the stability margin $\overline{MK}$ before changing the position of $\overline{CD}$. If $\overline{CD}$ is moved down the slope to $\overline{C'D'}$, then the stability margin is $\overline{MK'}$ as shown in Figure 5.13b. Note that the length of $\overline{MK'}$ is greater than that of $\overline{MK}$.

Though the above discussion is specific to straight-line locomotion for the tripod gait over simple sloped terrain, it is also applicable to arbitrary motion of the vehicle for other gaits over uneven terrain.



(a)

Figure 5.13. Adjustment of the Constrained Working Volume of a Middle Leg over Sloped Terrain.

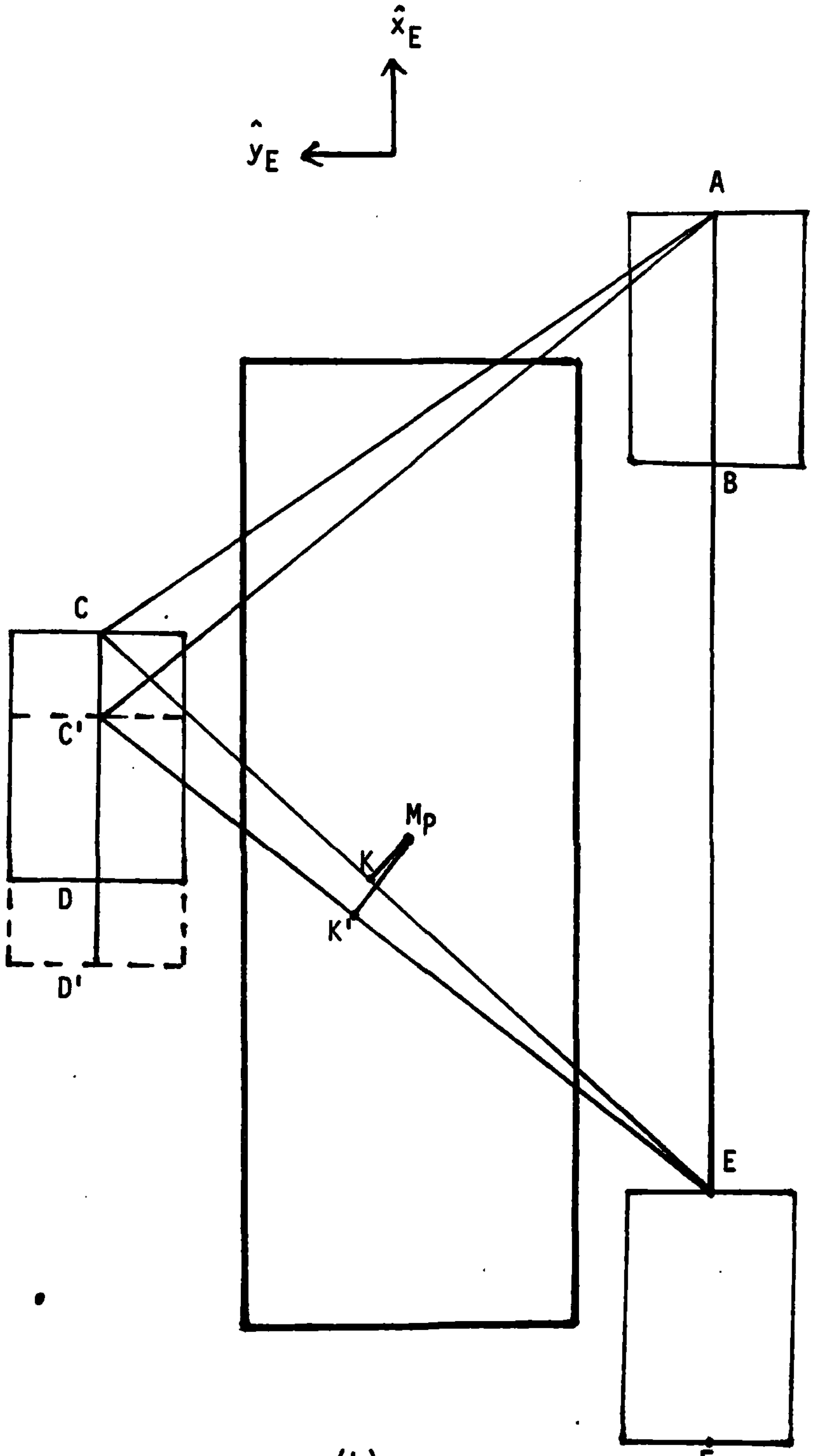


Figure 5.13 (continued).

Thus, the position of the constrained working volume of a middle leg may be dynamically changed according to the local terrain slope to increase stability. The position change is made in the $\hat{x}_B$ direction only, since the magnitude of the y position of each constrained working volume may be simply set to a maximum in order to increase the stability margin.

As shown in Figure 5.14, the points C and R are the center and the reference position of the constrained working volume for a middle leg, respectively, before adjustment. Point N is the projection of C onto the X_B-Y_B plane in the $\hat{z}_B$ direction. The points C' and R' are the new center and the new reference position, respectively, after adjustment such that N' is the projection of N along the gravity vector onto the estimated support plane. Let the position vector of C of middle leg i, which is given in Section 4.2, be

$$^{B}\underline{p}_C = [X_{Ci} \quad Y_{Ci} \quad Z_{Ci}]^T. \tag{5.78}$$

Then, the position vector of N in body coordinates is

$$^{B}\underline{p}_N = [X_{Ci} \quad Y_{Ci} \quad 0]^T, \tag{5.79}$$

and, in earth coordinates, it is

$$^{E}\underline{p}_N = H \;\, ^{B}\underline{p}_N = [X_N \quad Y_N \quad Z_N]^T. \tag{5.80}$$

Let the position vector of N' in earth coordinates be

$$^{E}\underline{p}_{N'} = [X_{N'} \quad Y_{N'} \quad Z_{N'}]. \tag{5.81}$$

From Eqs. (5.80) and (5.81),

$$X_{N'} = X_N \tag{5.82}$$

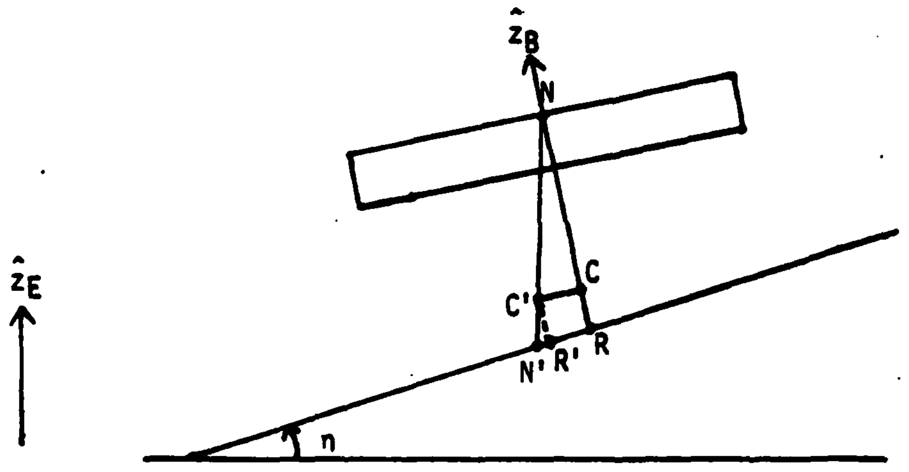


Figure 5.14. Computation of the New Reference Position for the Middle Leg.

and

$$Y_{N'} = Y_N. \tag{5.83}$$

From Eq. (5.8),

$$Z_{N'} = -(AX_{N'} + BY_{N'} + D) / C. \tag{5.84}$$

Thus, in body coordinates, the position vector of N' is

$${}^B\underline{p}_{N'} = H^{-1}\ {}^E\underline{p}_{N'} = [a\ \ b\ \ c]^T, \tag{5.85}$$

and the new center is obtained by replacing the x component of ${}^B\underline{p}_C$ of Eq. (5.78) with that of ${}^B\underline{p}_{N'}$ of Eq. (5.85):

$${}^B\underline{p}_{C'} = [a\ \ Y_{Ci}\ \ Z_{Ci}]^T. \tag{5.86}$$

The corresponding reference position R' is determined by Eqs. (5.43) through (5.48).

An observation has been made in Section 4.2 that the stability of vehicle motion increases by moving the soft limits imposed by the constrained working volumes further away from the center of gravity. In other words, stability may increase by changing the reference position to be further away from the center of gravity. However, since the position of the constrained working volume for each leg is defined to be located at the corner of its working volume as mentioned in Section 4.2, the only way to change the reference position is to reduce the size of the constrained working volume.

If the body speed is slow, the leg does not need the large constrained working volume since the support trajectory for a given support period may be short. Thus, for slower speeds, the size of the constrained working volume may be made smaller, which in turn causes the reference position to move further away from the center of gravity.

This approach may increase stability considerably, especially for locomotion over terrain with a stiff slope, because the body velocity is usually commanded to be very slow in this case.

The change of dimensions of the constrained working volume of a leg should be made while the leg is in the transfer phase. As mentioned in Section 5.4, the desired foothold of the transfer leg is specified to be on the boundary of the constrained working volume such that its predicted support trajectory passes through the center of its constrained working volume. If the dimensions are changed after the desired foothold is chosen, it is possible for the support point of the leg, while in the support phase, to go outside the constrained working volume due to body velocity changes. This complicates the algorithm for determining the optimal period as discussed in Chapter 4, and should be avoided.

The change of the constrained working volume is made on the x and y dimensions only, since the z component of the foot velocity is relatively small compared to the x and y components during normal operation of the vehicle. Thus, for simplicity, d_{xi} and d_{yi} of the constrained working volume of leg i are set to vary linearly, between minimum and maximum values, with the predicted foot velocity given by Eq. (5.52). The equations for the change of d_{xi} and d_{yi} are given as follows:

$$d_{xi} = d_{xmin} + \frac{(d_{xmax} - d_{xmin})}{(v_{xmax} - v_{xmin})} (|\dot{X}_R| - v_{xmin}) \tag{5.87}$$

$$d_{yi} = d_{ymin} + \frac{(d_{ymax} - d_{ymin})}{(v_{ymax} - v_{ymin})} (|\dot{Y}_R| - v_{ymin}), \tag{5.88}$$

where d_{xmin}, d_{xmax}, d_{ymin}, d_{ymax}, v_{xmin}, v_{xmax}, v_{ymin}, and v_{ymax} are constants.

It is possible to obtain minimum allowable values for d_{xi} and d_{yi} for straight-line locomotion over flat level terrain in order to avoid vehicle slow-down (see Section 4.4). These values are a function of the body velocity, and the theoretical curves obtained in the following analysis will aid in setting reasonable values for d_{xmin}, d_{xmax}, d_{ymin}, etc. For example, when the vehicle moves with the constant speed v_x in the $\hat{x}_B$ direction only and wave gaits are used, then the following equations hold [8]:

$$\lambda = \tau v_x \ , \tag{5.89}$$

$$d_{xi} = \beta\lambda \ , \tag{5.90}$$

and

$$\tau_R = (1 - \beta)\tau \ . \tag{5.91}$$

However, as discussed in Section 4.4, in order to avoid slow-down, the transfer time should be greater than the transfer time threshold:

$$\tau_R > \tau_{Rth} . \tag{5.92}$$

From Eqs. (5.89) through (5.92), values for d_{xi} which avoid slow-down may be computed:

$$d_{xi} > \frac{\beta}{1 - \beta} \tau_{Rth} v_x \ . \tag{5.93}$$

The results for d_{yi} are similar for pure lateral motion. Figure 5.15 gives a graph of the minimum allowable values for d_{xi} as a function of v_x for three different values of β when τ_{Rth} = 0.5 sec.

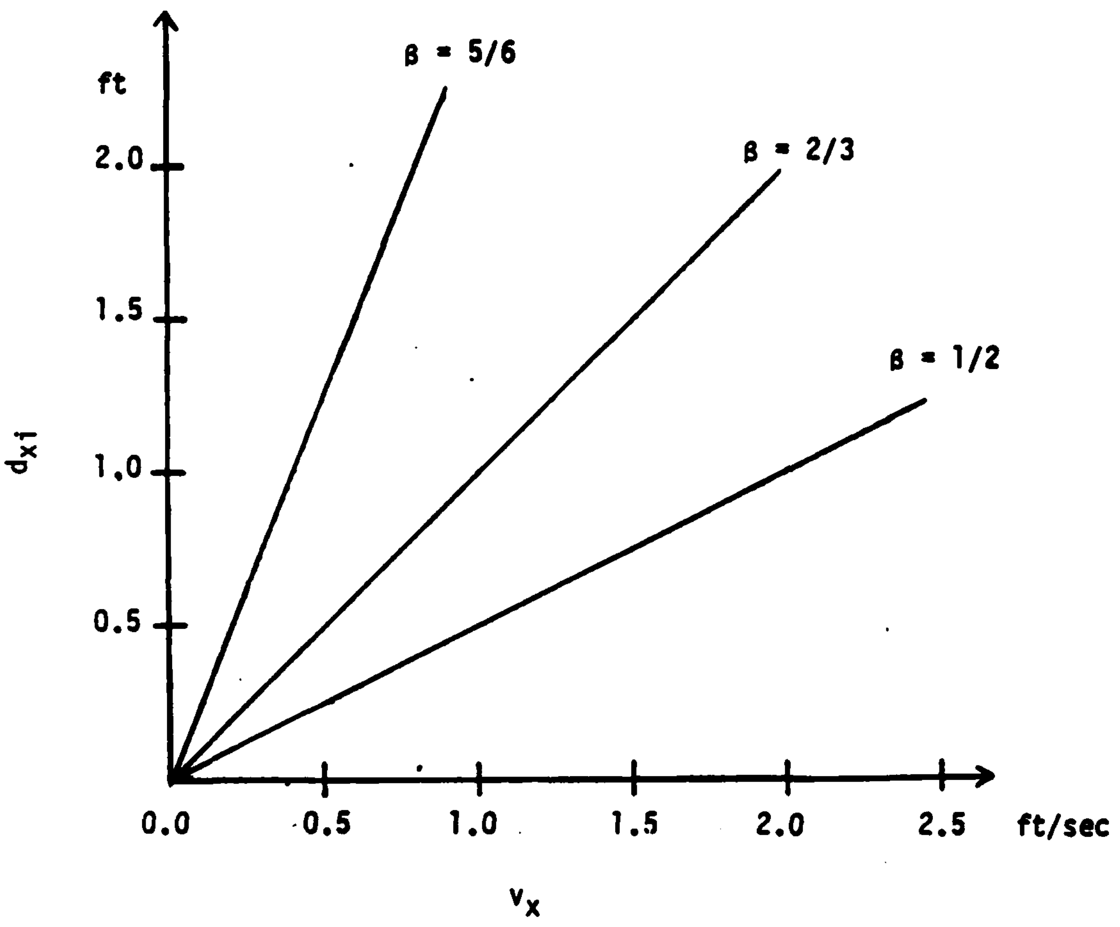


Figure 5.15. Minimum X-Dimension of the Constrained Working Volume as a Function of the Forward Velocity (v_x) and Duty Factor (β) in Order to Prevent Slow-down when the Minimum Leg Transfer Time (τ_R) is 0.5 sec.

5.6 Summary

Algorithms developed for motion planning over uneven terrain, which allow implementation of the Close Maneuvering Mode of operation for the ASV, were presented in this chapter. These algorithms make use of proximity sensors for each leg and these were also described in detail. Since in this simulation work the actual foothold on the terrain is exactly measured by proximity sensors, there is no distinction between these and contact sensors. However, for the vehicle itself, contact sensors shoud be used to ensure foot contact. The proximity sensors provide local terrain information before the legs touch the ground. This enables the vehicle to adapt to terrain irregularities by making soft-contact on the terrain surface.

The estimation algorithm for the support plane has been developed in order to make use of proximity sensors for each leg by defining the rectangular transfer trajectory. The plane of support is estimated by using the support points and the predicted footholds through linear regression. Due to the limited number of estimation points, sometimes, the estimation may result in large error, especially in the region where the slope changes abruptly. If it is assumed that the vehicle speed is relatively slow and the terrain is relatively smooth in the Close Maneuvering Mode, the estimation error may not cause any trouble to the vehicle motion.

In order to describe the body motion over uneven terrain, three coordinate systems have been defined: the earth-fixed coordinate system, the body-fixed coordinate system, and the terrain coordinate system. The terrain coordinate system has been defined as a moving frame with respect to the earth-fixed coordinate system, which characterizes the local terrain conditions.

Using the three coordinate systems, the automatic body attitude and altitude control scheme has been presented in a mathematically straightforward manner according to a simple body regulation plan. The body regulation plan has been designed in such a way that the body attitude and altitude can be related to the slope angle through piece-wise linear curves and the desired unit normal vector of the body plane lies between the gravity vector and the normal vector of the estimated support plane.

The support and transfer trajectories for the foot of the leg have been defined. Especially, the transfer trajectory is generated by using the proximity sensor in order to avoid foot contact with the terrain. The intersection point of the estimated support plane with the center line of the constrained working volume of each leg has been chosen as a reference position to select its desired foothold, while in the transfer phase. The desired foothold is selected at a point on the surface of the constrained working volume by approximating the circular foot trajectory during the support phase with a straight line. This approximation results in simple computation.

Finally, in order to increase the stability margin over uneven terrain, two schemes have been presented: one is for the change of the position of the constrained working volume for each of the middle legs and the other is for the change of dimensions of the constrained working volume for each of all legs. The position of the constrained working volume of the middle leg is changed in the $\hat{x}_B$ direction only toward the gravity vector, while in the transfer phase. The size of the constrained working volume is adjusted according to the predicted foot velocity during the transfer phase without causing the vehicle to unnecessarily slow down. These schemes result from the flexibility in changing the position and dimensions of the constrained working volume and enhance the stability of the vehicle for locomotion over uneven terrain.

Chapter 6
CONTROL SOFTWARE

6.1 Introduction

Recently, a control software architecture for the Hexapod vehicle has been proposed by Orin and Pugh [41], in which the control program is partitioned into functional blocks, or subtasks, for the multicomputer system. Since communication between the blocks is still under investigation, the software structure of the control program for the ASV developed in this work was based on the previous control programs for the OSU Hexapod [11,12] and modularized into the several routines. Each routine consists of several subroutines which will be discussed in Section 6.2.

In Section 6.3, evaluation of the control algorithms developed in Chapter 5 will be presented. Goodness of the algorithms is measured by the stability margin when the vehicle moves forward over sloped terrain.

6.2 Software Organization

There are six functional blocks defined in this simulation software as shown in Figure 6.1 with slight modification of the control task partitioning of the OSU Hexapod contol program version 3.0 given in [11].

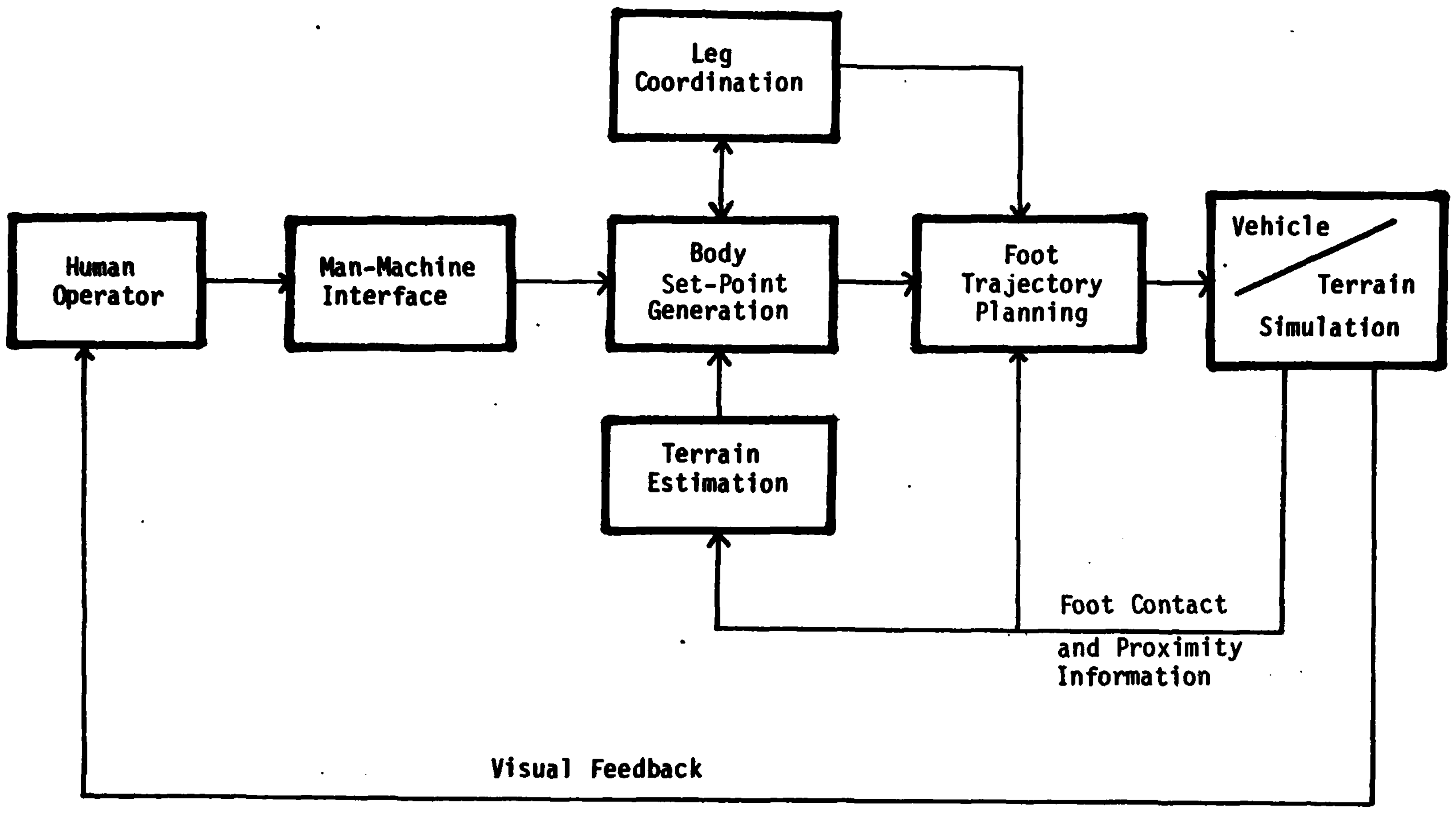


Figure 6.1. Functional Block Diagram of the Simulation Program for Control of the ASV.

The servo control functional block which is essential to control the vehicle at the servomechanism level is replaced by the Vehicle Simulation block.

Figure 6.2 shows the structured flow chart [42] for the control software for the ASV in which the solid lines indicate the subroutine calls and the dashed arrows indicate file transfer between two modules. For example, module BODY invokes module LEGCORD and LEGCORD is subordinate to BODY. Each module in Figure 6.2 represents the actual routine implemented and corresponds to each functional block. The Terrain Simulation block consists of two independent modules which pass the database for the prismatic terrain model and the viewing angle to the control software. The software is implemented in PASCAL except the interface routines for the graphics device and the joystick on the PDP-11/70 minicomputer.

6.2.1 Man-Machine Interface

The Man-Machine Interface block provides the interface between the human operator and the control program for the interactive change of the related parameters for the vehicle motion using the keyboard and for the velocity commands through a three-axis joystick. Before starting to move the vehicle, the operator can interactively change the duty factor for the desired wave gaits, the maximum height of footlift for the foot transfer trajectory and then enter a command to initialize the vehicle position through the keyboard.

When the initialization is finished, another keyboard command is entered to start motion. After that, the vehicle motion is directed by

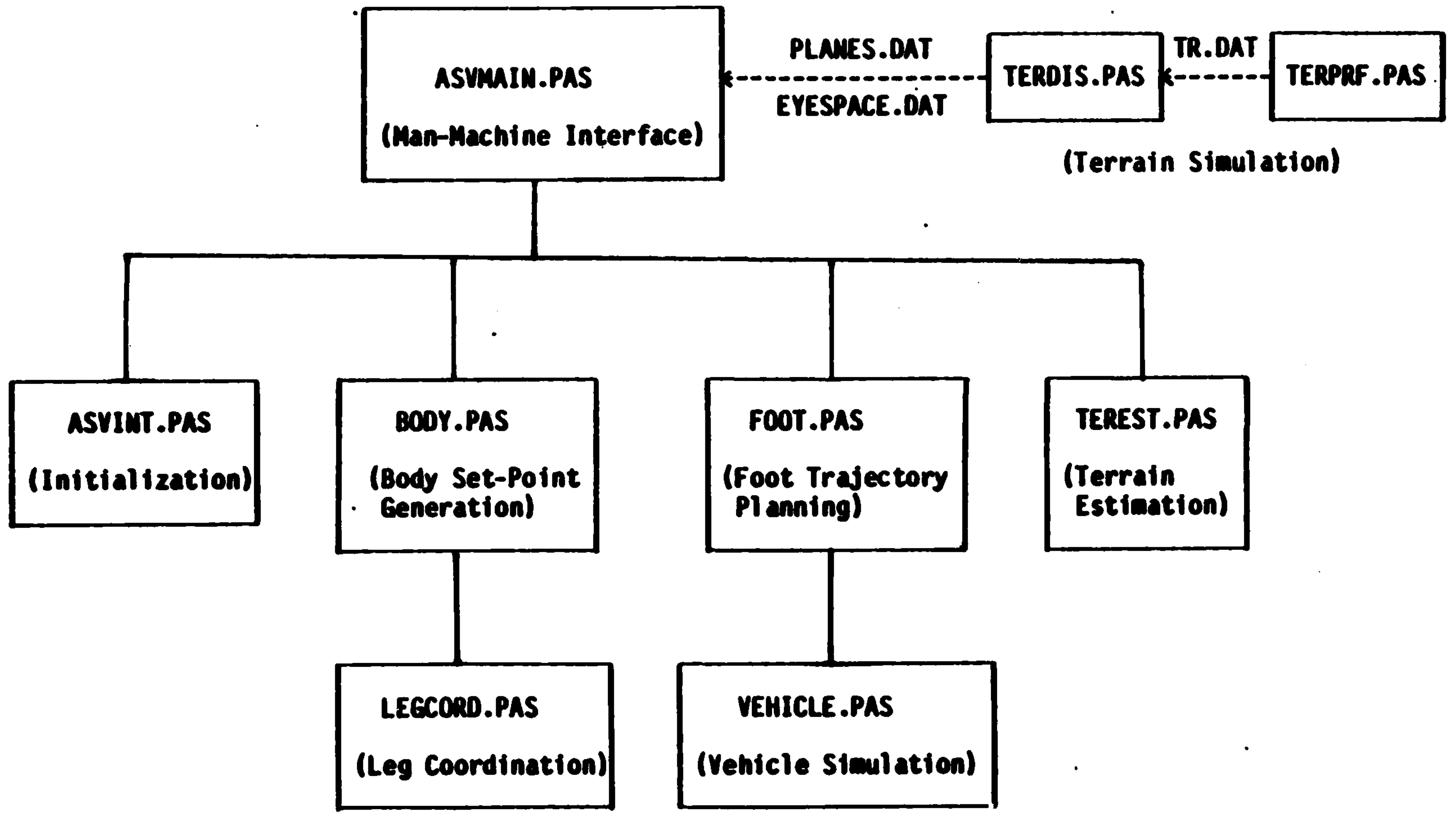


Figure 6.2. Structured Flow Chart of the Control Software for the ASV.

the joystick velocity commands until a keyboard command which is for the vehicle to halt is entered. Figure 6.3 depicts the exact relationship between the three components of the body velocity and the axes of the joystick [8].

6.2.2 Body Set-Point Generation

This block generates body position and velocity set-points for the six-degree-of-freedom body motion based on the velocity commands from the Man-Machine Interface block according to the automatic body regulation plan in coorperation with the terrain information from the Terrain Estimation block. All the six velocity components are filtered to get the realizable values to the vehicle system. The body velocity set-points are passed to the Leg Coordination block in order to compute the optimal period which is used to determine whether the vehicle needs to slow down. If it is necessary to slow down the vehicle speed, each of the three velocity components: v_x, v_y and ω_z, is reduced by the amount of the slowdown factor which is passed from the Leg Coordination block.

The body position is updated by the three body translational velocity components, and the body orientation is defined by the three body rotational velocity components. The updated information of the body position and orientation is stored in the homogeneous transformation matrix, H. Figure 6.4 shows the flow chart for this block.

6.2.3 Leg Coordination

The change of the size of the constrained working volume is also made in this block according to the predicted foot velocity of the leg while in the transfer phase. Then, the optimal period is computed using

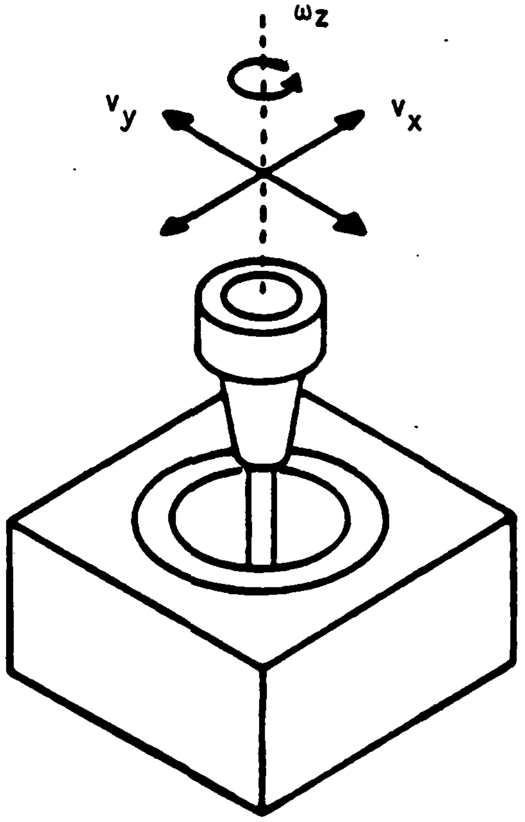


Figure 6.3. Three-Axis Joystick for Control of Forward (v_x), Side (v_y), and Rotational (ω_z) Components of Body Velocity [6].

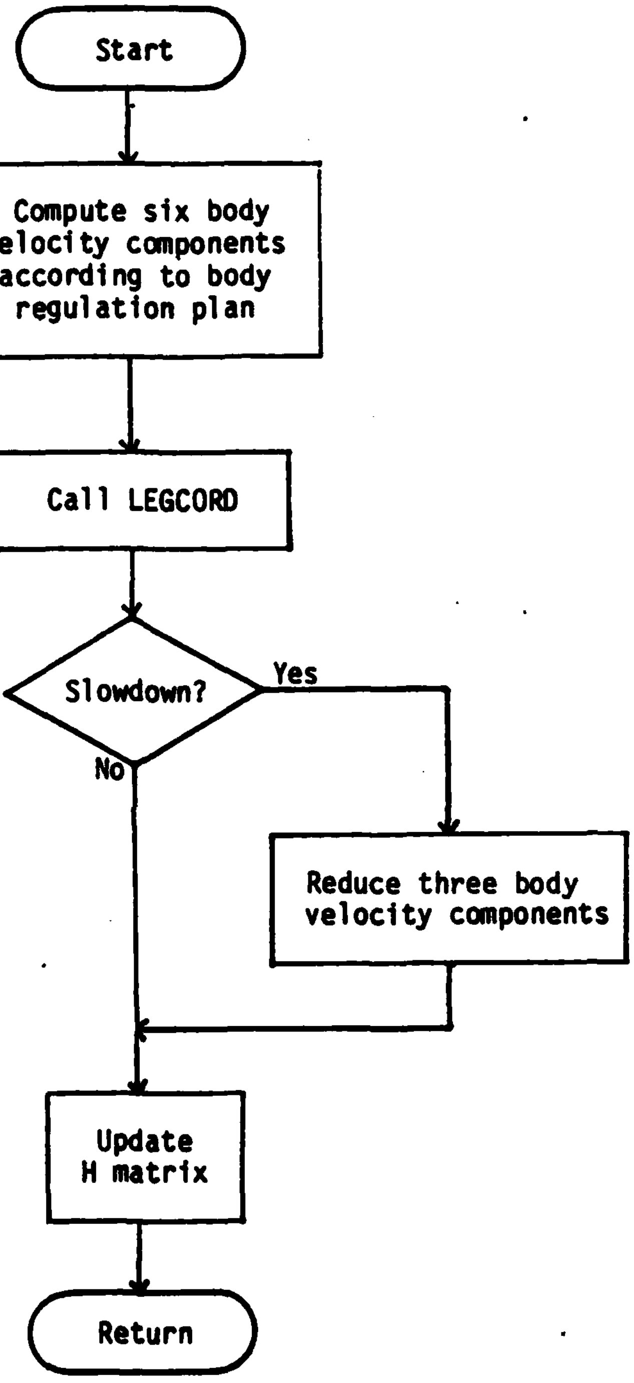

Start
Compute six body velocity components according to body regulation plan
Call LEGCORD
Slowdown?
Yes
No
Reduce three body velocity components
Update H matrix
Return

Figure 6.4. Flow Chart for BODY.

the new constrained working volume. If the optimal period obtained is too short to properly coordinate the leg stepping sequence, the deceleration plan is invoked to compute the slowdown factor which is passed to the Body Set-Point Generation block. Also, the transfer time is set to the transfer time threshold and the new period for slowdown is computed.

This block updates the kinematic cycle phase variable, ϕ, for each execution cycle of the program according to the period, τ, to determine the periodic leg stepping sequence. By using the updated value of ϕ, the leg phase variable, ϕ_{Li}, of leg i is defined. Then the leg phase variables are passed to the Foot Trajectory Generation block. Figure 6.5 shows the flow chart for this block.

6.2.4 Foot Trajectory Planning

The leg phase variable passed from the Leg Coordination block specifies the state of leg i: support phase or transfer phase. When leg i is in the support phase, the support trajectory is generated to move the vehicle according to the body velocity and position defined in the Body Set-Point Generation block. When leg i is in the transfer phase, the transfer phase variable, ϕ_{Ti}, is computed to determine which subphase leg i belongs to, and to generate the appropriate foot trajectory for the corresponding subphase. Figure 6.6 shows the flow chart for this block.

6.2.5 Terrain Estimation

This block estimates the local terrain surface as a 2-D plane using six points of estimation passed from the Foot Trajectory Planning block

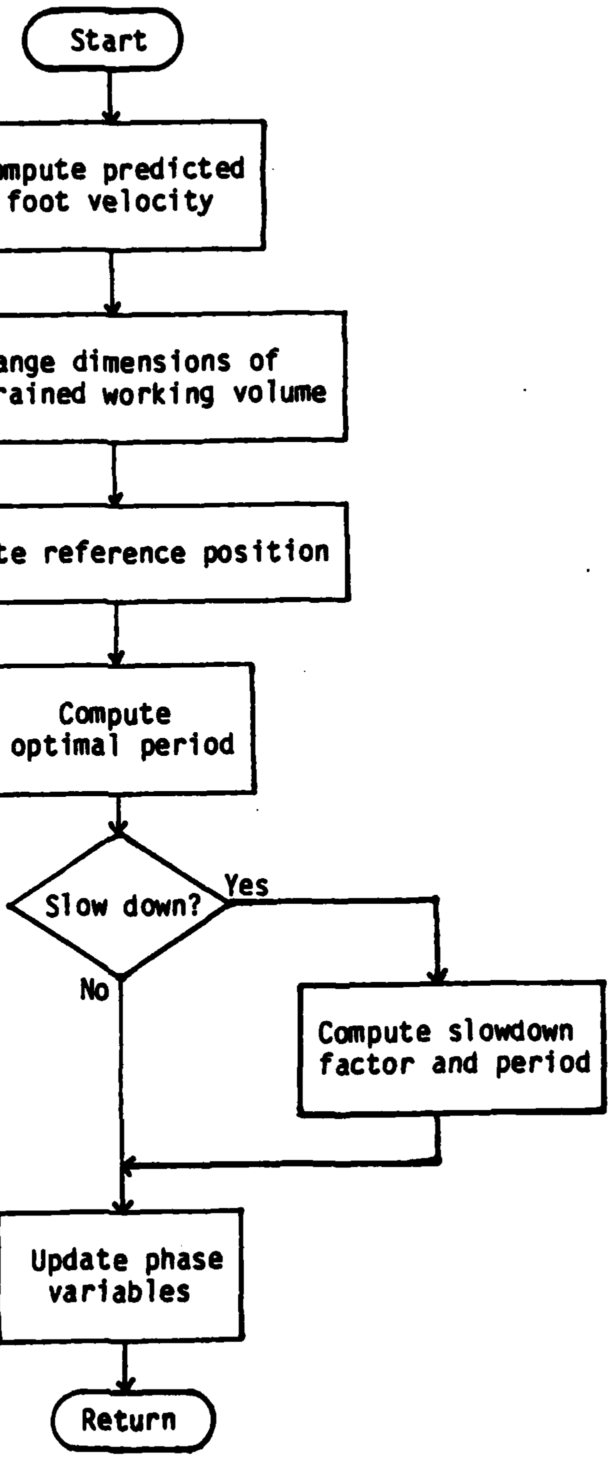

Start
Compute predicted foot velocity
Change dimensions of constrained working volume
Compute reference position
Compute optimal period
Slow down?
Yes
No
Compute slowdown factor and period
Update phase variables
Return

Figure 6.5. Flow Chart for LEGCORD.

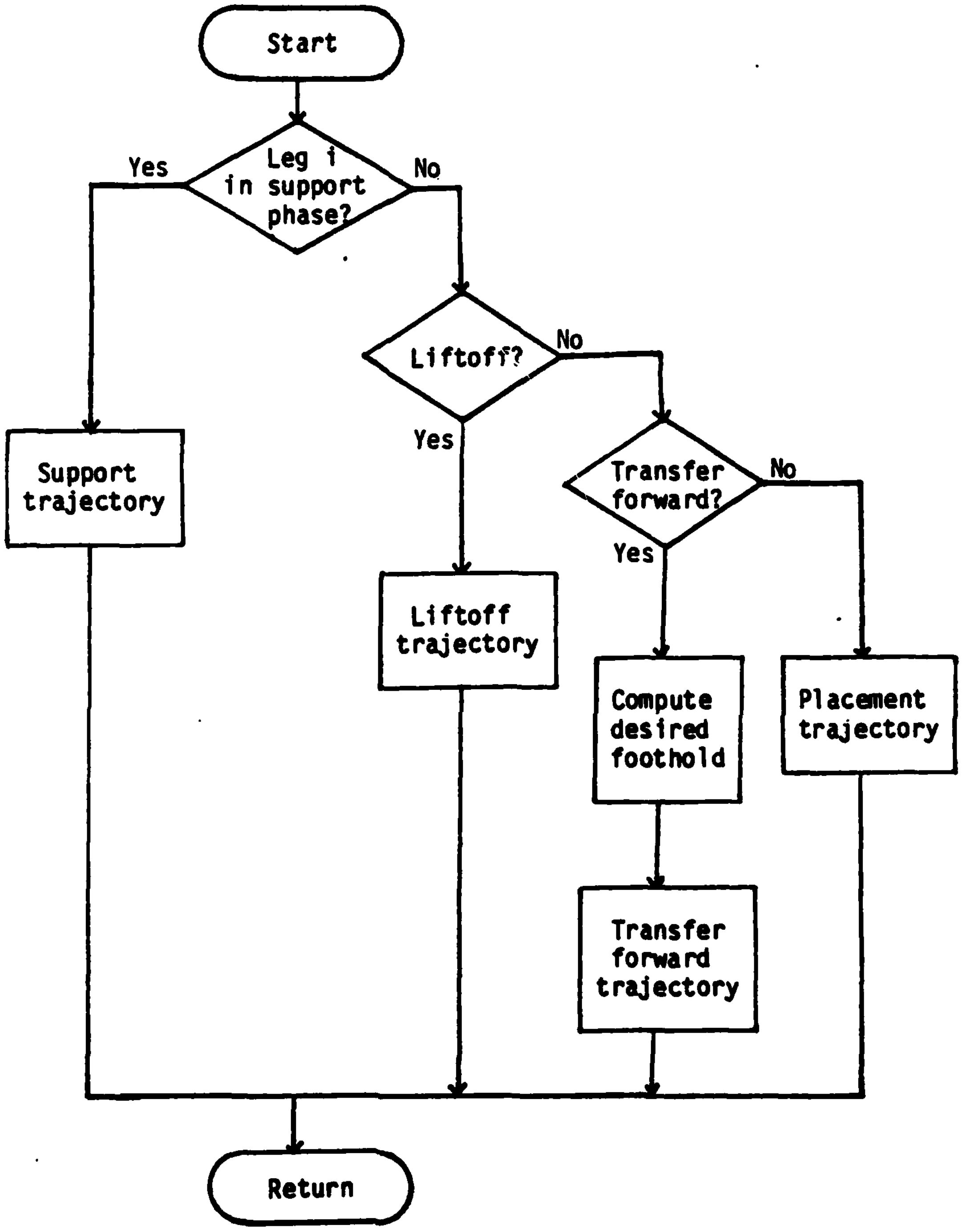


Figure 6.6. Flow Chart for FOOT.

through linear regression. The three coefficients of the plane equation are defined as the components of ${}^{E}\hat{z}_T$ of the terrain coordinate system. Then the homogeneous transformation matrices, T and G, are computed. Figure 6.7 shows the flow chart for this block.

6.2.6 Vehicle/Terrain Simulation

There are two independent modules developed for interactive generation of the prismatic terrain models. As shown in Figure 6.8, in the terrain profile generation routine (TERPRF), the terrain profile is interactively generated on the display screen using the joystick. Then the terrain profile (TR.DAT), which is defined in the X_E-Z_E plane, is passed to the terrain display routine (TERDIS). The display routine is responsible for generating the 3-D terrain database defined as "Planes" and a homogeneous viewing transformation matrix defined as "EyeSpace" for the perspective view of the prismatic terrain by using the joystick. The database for the prismatic terrain (PLANES.DAT) and that for the viewing angle (EYESPACE.DAT) are passed to the control program.

The vehicle is drawn as a stick figure. Upon receiving the desired body and foot positions from the motion planning block, the perspective view of the ASV is displayed on the prismatic terrain according to the viewing transformation matrix. In addition, software implementation for contact and proximity sensors is also included in the Vehicle Simulation block using the prismatic terrain database.

The object description data files for the display of the terrain and vehicle models are defined in the format described in [43]. In this format, a solid object is defined as a set of adjoining polygons. Since neighboring polygons share vertices along common edges, objects can be

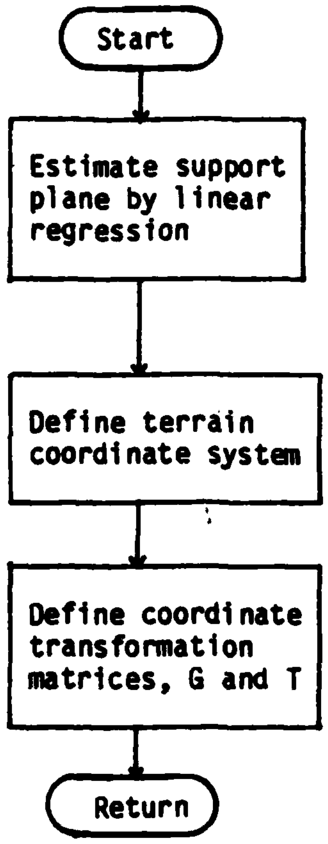
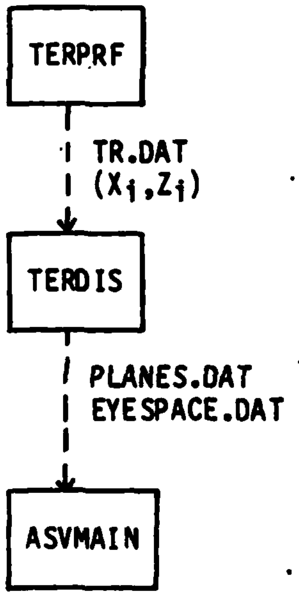

Start
Estimate support plane by linear regression
Define terrain coordinate system
Define coordinate transformation matrices, G and T
Return

Figure 6.7. Flow Chart for TEREST.



```
Type OnePlane = record
                UnitNormal: Vector;
                Dist: real;
                Xmin, Xmax: real;
                end;

var Planes: array[1..50] of OnePlane;
    EyeSpace: array[1..4,1..4] of real;
```

Figure 6.8. Database for Terrain and Viewing Angle.

more compactly defined by first listing all the vertices belonging to the object and then listing polygons by the numbers of the vertices they use.

6.3 Evaluation of Control Algorithms

The control algorithms for locomotion over uneven terrain developed in this simulation work are composed of three different algorithms:

1) Algorithm for body altitude control,
2) Algorithm for the position change of the constrained working volume of the middle leg,
3) Algorithm for the change of the size of the constrained working volume.

As discussed in Chapter 5, each algorithm provides more stability for the vehicle motion than that for the vehicle motion without it. Thus it may increase adaptability of the vehicle to terrain irregularities.

To evaluate the performance of each algorithm, it was desired to decide the type of the terrain model and the vehicle motion for purposes of comparison. Since the motion planning of the vehicle is based on the estimated plane of support which may be regarded as a simple slope, it was reasonable to choose a sloped terrain model. Also, for simplicity, the vehicle motion was limited to the forward motion only with the crab angle zero, and the body was kept parallel to the slope.

In this test, thirty different slopes from 0 to 30 degrees were used for the terrain models. The length of each slope in the direction of the forward motion was 40 ft and the stability margin of the vehicle motion over each slope was chosen as the smallest one while the vehicle was moving 10 ft long up the slope.

Each algorithm was tested independently. When one algorithm was tested, the other two were not implemented in the test program. The position of the constrained working volume was defined in the same way as shown in Figure 4.2, unless otherwise stated.

For the evaluation of the first algorithm, the positions and dimensions of the constrained working volumes of the six legs were fixed such as d_{xi} = 2.8 ft, d_{yi} = 2.0 ft, and d_{zi} = 2.8 ft. The forward velocity was fixed to 1.5 ft/sec. The test was done for three different values of the body height, which were 4.5 ft, 5.0 ft, and 5.5 ft, for each of three different duty factors: 1/2, 2/3, and 5/6. Figure 6.9 shows the results of this test. It clearly shows that the stability margin for the given slope angle increases when the body height decreases for each duty factor.

The test of the second algorithm was performed under same conditions as given for that of the first algorithm except that the body height was fixed to 5.4 ft. For purpose of comparison, the stability margins were also computed for the case that no position changes of the constrained working volumes of the middle legs were made. As done in the test of the first algorithm, the test results were obtained for three different duty factors. Figure 6.10 shows that the second algorithm resulted in the drastic increase in the stability margin for the given slope angle compared to the case of no position changes made in the constrained working volumes of the middle legs.

For the test of the third algorithm, the body height was fixed to 5.4 ft above the slope, and the body velocity was chosen to be 0.5 ft/sec. The x and y dimensions of the constrained working volumes of the six legs were computed from Eqs. (5.85) and (5.86), where

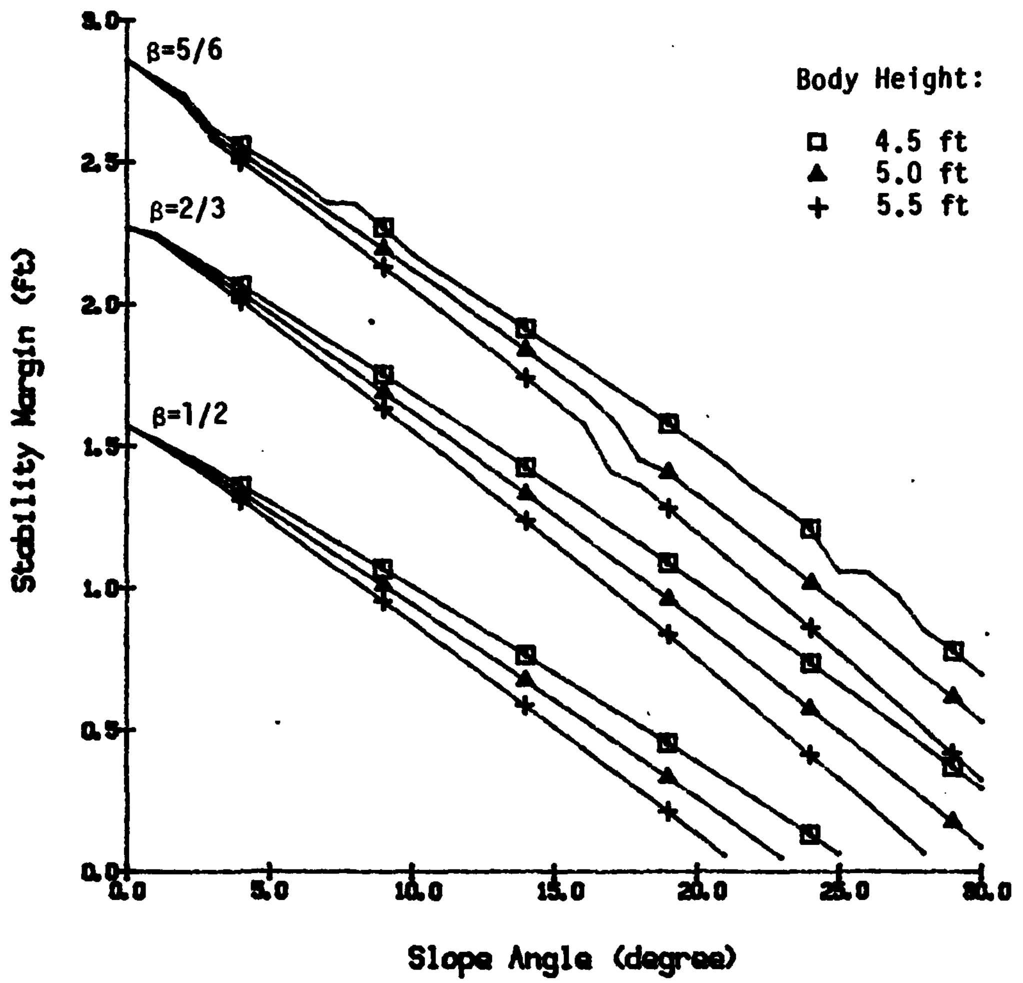


Figure 6.9. Stability Margin as a Function of Body Height and Leg Duty Factor over Varying Degrees of Slope. (Body parallel to terrain, constant forward velocity--1.5 ft/sec, constant size and position of the constrained working volumes.)

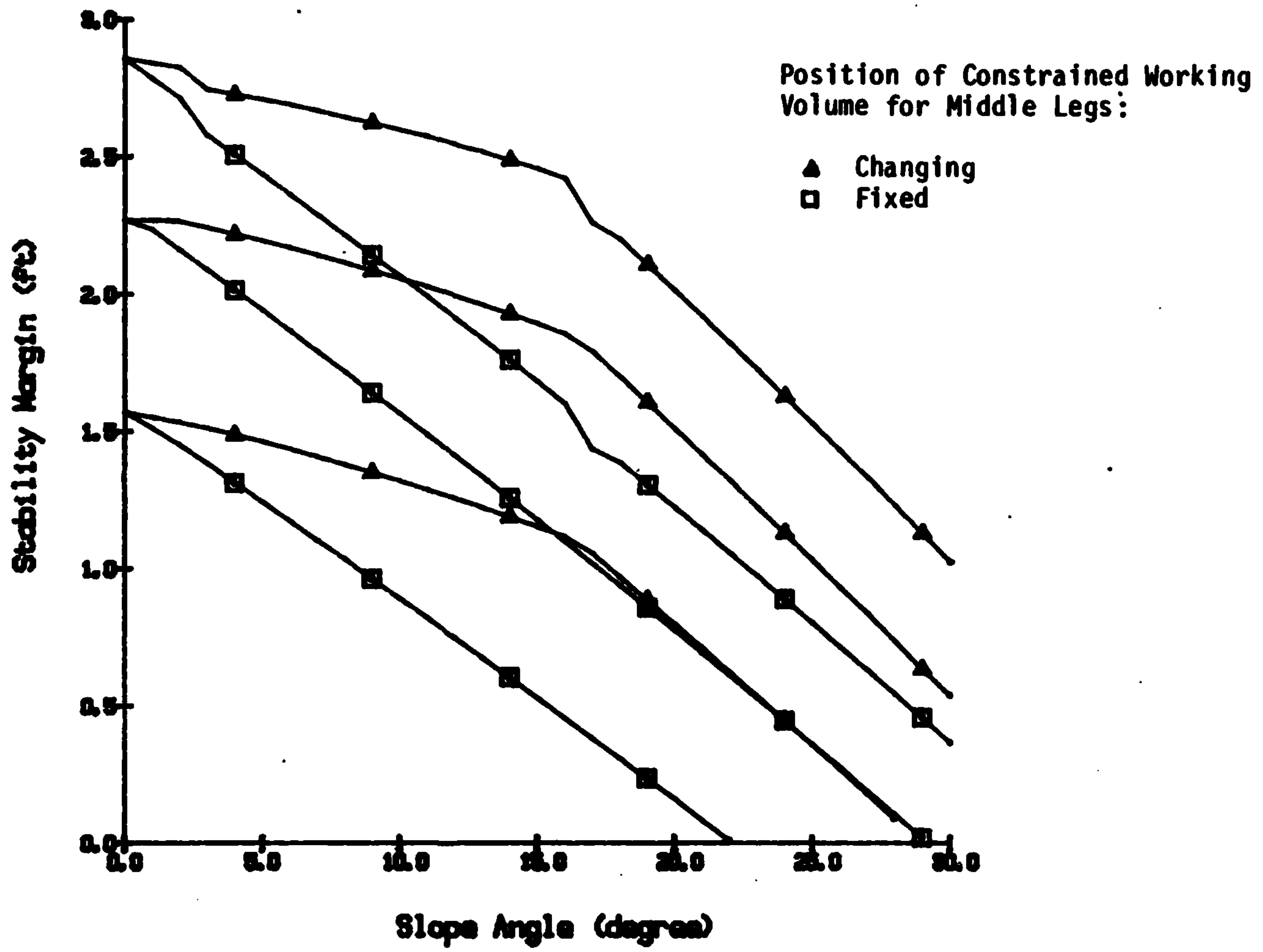


Figure 6.10. Stability Margin for Changing and Fixed Positions of the Constrained Working Volumes for the Middle Legs. (Body parallel to terrain, constant forward velocity--1.5 ft/sec, constant size of the constrained working volumes, constant body height--5.4 ft.)

$v_{xmax} = v_{ymax} = 2.0$ ft/sec, $v_{xmin} = v_{ymin} = 0.25$ ft/sec, $d_{xmax} = 3.0$ ft, $d_{ymax} = 2.2$ ft, and $d_{xmin} = d_{ymin} = 0.25$ ft. The resulting dimensions were $d_{xi} = 0.65$ ft and $d_{yi} = 0.53$ ft. The stability margins obtained by employing the third algorithm is compared to those of the case when the dimensions of the constrained working volumes of the six legs were fixed to be $d_{xi} = 2.8$ ft and $d_{yi} = 2.0$ ft. Figure 6.11 shows the result when the duty factor is 2/3. The result indicates the better stability of the vehicle motion as expected.

From the results obtained by the above three tests, it is expected that the better result may be obtained, if all the algorithms for stabillity enhancement are functioning. Figure 6.12 shows the result obtained when the overall control algorithms developed in this work was applied to the same terrain as used in the above tests with the constant forward velocity 1.5 ft/sec, and Table 6.1 shows the actual parameters used in the control algorithms. Compared to the results of the above three tests, the result given in Figure 6.12 shows the considerable increase in stability, especially, for the large slope. It is also noted that the stability margin increases very significantly when $\beta=1/2$. This may indicate that the algorithms for the change of dimensions of the constrained working volume are more effective for the small values of β than for the large ones.

6.4 Summary

In this chapter, a brief description about the software to implement the control algorithms and interactive generation of the terrain models developed in this simulation work was presented. Also, evaluation of the performance of the control algorithms was discussed through software simulation.

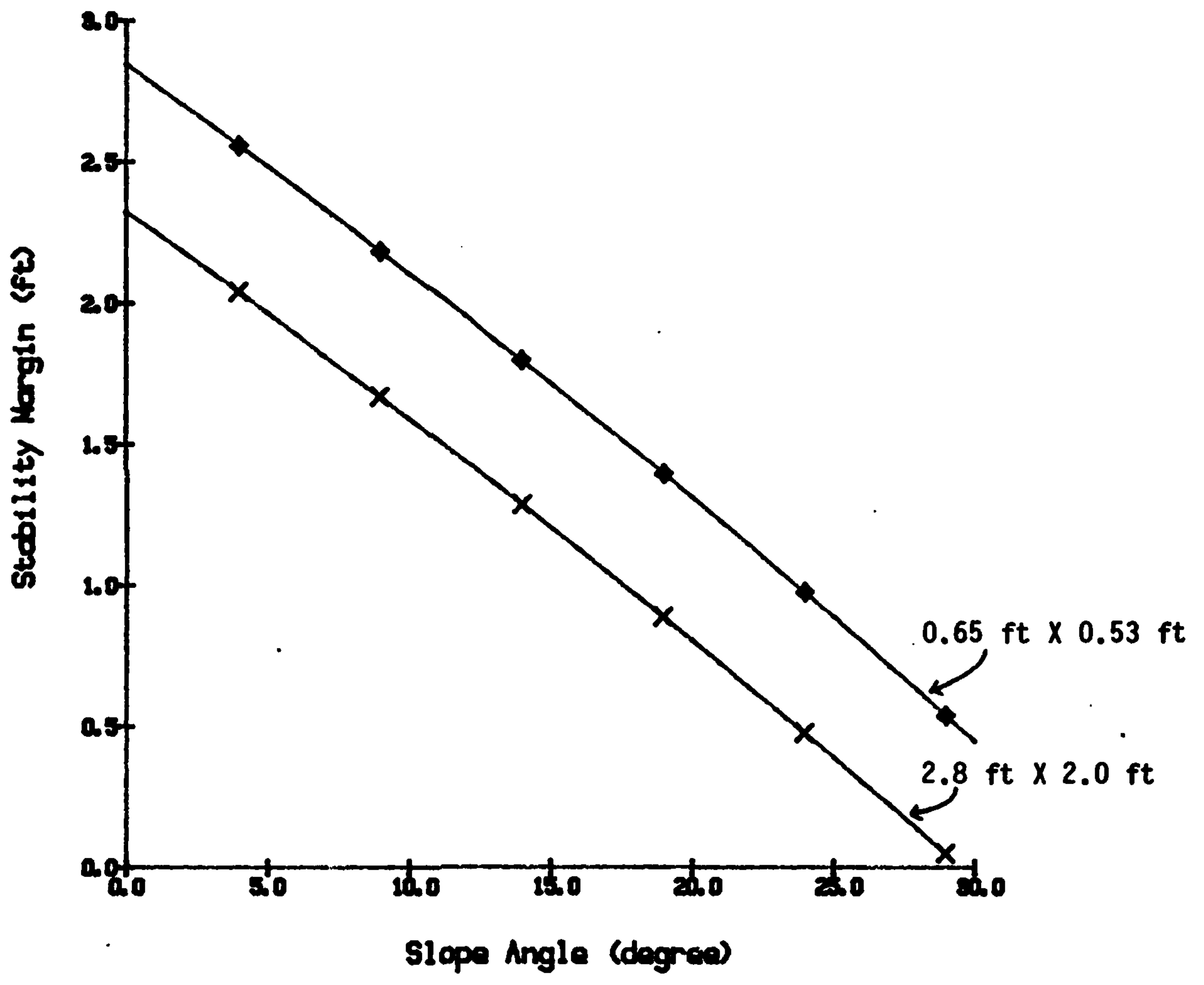


Figure 6.11. Stability Margin for Two Different Sizes of the Constrained Working Volumes. (Body parallel to terrain, constant forward velocity--0.5 ft/sec, β=2/3, constant body height--5.4 ft.)

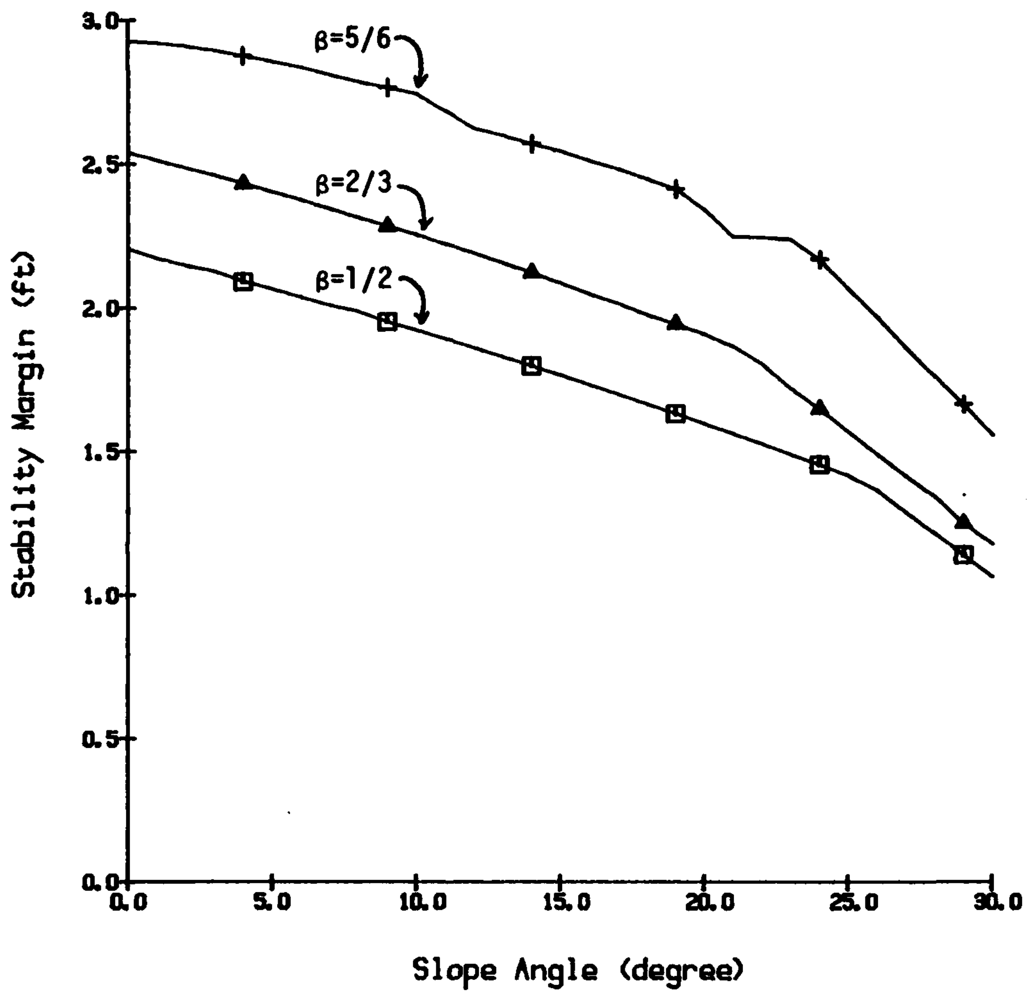


Figure 6.12. Stability Margin of the ASV over Sloped Terrain when All Algorithms for Stability Enhancement are Functioning. (Altering size and position of the constrained working volume, body altitude adjustment.) Note that parameters used appear in Table 6.1.

TABLE 6.1

ACTUAL PARAMETERS USED IN CONTROL ALGORITHMS FOR THE ASV

(1) Size of the constrained working volume

	v_{xmax}	v_{ymax}	v_{xmin}	v_{ymin}	d_{xmax}	d_{ymax}	d_{xmin}	d_{ymin}
$\beta=5/6$	1.25	1.25	0.25	0.25	3.0	2.2	0.75	0.75
$\beta=2/3$	2.0	2.0	0.25	0.25	3.0	2.2	0.25	0.25
$\beta=1/2$	3.0	3.0	0.25	0.25	2.0	2.2	0.25	0.25

Units for v_{xmax}, v_{ymax}, v_{xmin}, and v_{ymin} are ft/sec.

Units for d_{xmax}, d_{ymax}, d_{xmin}, and d_{ymin} are ft.

(2) Center position of the constrained working volume

$$X_{Ci} = \begin{cases} X_{Hi} + 3.0 - d_{xi}/2 & \text{for } i=1,2 \\ X_{Hi} - 3.0 + d_{xi}/2 & \text{for } i=5,6 \\ X_{Hi} & \text{for } i=3,4 \end{cases}$$

$$Y_{Ci} = \begin{cases} Y_{Hi} + 2.2 - d_{yi}/2 & \text{for } i=1,3,5 \\ Y_{Hi} - 2.2 + d_{yi}/2 & \text{for } i=2,4,6 \end{cases}$$

$$Z_{Ci} = -5.4 \text{ ft} \qquad \text{for } i=1,..,6$$

$X_{H1}=X_{H2}=5.0$ ft; $X_{H3}=X_{H4}=0.0$ ft; $X_{H5}=X_{H6}=-5.0$ ft;

$Y_{H1}=Y_{H3}=Y_{H5}=2.25$ ft; $Y_{H2}=Y_{H4}=Y_{H6}=-2.25$ ft.

(3) Body attitude and altitude

$\eta_1 = 5^\circ$; $\eta_2 = 45^\circ$; $\eta_{max} = 45^\circ$;

$h_0 = 5.4$ ft; $h_{min} = 4.0$ ft.

The program modules were defined to coincide with the functional blocks. However, they were loosely modularized by allowing the global parameters to pass through modules. The problems of parameter passing between modules need further investigation in order to implement the control software on a multicomputer system.

The tests of the performance of the three different control algorithms were done independently by computing the stability margin over the thirty different slopes. The results show the significant increase in stability for each of the three algorithms. Furthermore, when all the algorithms were functioning, the increase in stability was more significant.

The test results given in Section 6.3 may be changed by the different values of parameters: the dimensions of the constrained working volumes, the body height, the body velocity, terrain conditions, etc. However, the results show the general trend of the performance of the control algorithms developed in this work.

Chapter 7
SUMMARY AND CONCLUSIONS

In this dissertation, the problem of omnidirectional control for locomotion of a walking vehicle over rough terrain, specifically, "Type-0" terrain [6], has been investigated through computer simulation. As a result of this research, the first complete control algorithm based on the vehicle kinematics for rough-terrain locomotion has been developed. It should allow the vehicle to move over rough terrain in any direction with a high degree of stability. The results have shown that if the appropriate control algorithms are developed, the use of periodic gaits, which may provide a simpler means of implementing leg coordination than that of free gaits [7,28,29], is possible for rough-terrain locomotion.

7.1 Research Contributions

The implementation of periodic gaits for omnidirectional control has required that the simple notion of stride length adopted in previous control algorithms [8] be abandoned. Instead, a new concept has been introduced in order to select the locomotion cycle period for periodic gaits and is based on the reachability of the leg. The key concept is the notion of the constrained working volume for a leg which is defined as a rectangular subset of the overall leg working volume. The merits of the use of the constrained working volume include these:

1) Eliminating leg interference by avoiding overlapping constrained working volumes,

2) Increasing the stability by changing the position and dimensions of the constrained working volumes over rough terrain.

To provide omnidirectional control, the optimal cycle period is selected in such a way that at least one leg fully utilizes its constrained working volume. Also, each foothold is selected in such a way that the foot is predicted to pass through the center line of the constrained working volume. In addition, a simple deceleration strategy for the body velocity has been developed so as to keep each leg within the physical limits of its transfer time. While omnidirectional control was not possible in previous work, the use of the constrained working volume to define the locomotion cycle period and leg footholds has resulted in this new capability.

To control body motion over rough terrain, the local terrain surface is estimated by using six points of estimation based on the measurements from the six legs. Since the six estimation points may include the potential footholds detected through use of the proximity sensors, they may allow the body to pitch into a depressed surface even before a leg steps into that area. This aids in the leg being able to reach that surface. Also, in order to increase the safety and adaptability of the vehicle for rough-terrain locomotion, a simple body regulation plan has been designed which results in the body attitude adjusting to the terrain slope and the body height decreasing with greater slope.

To control the leg motion in the transfer phase, a simple rectangular type of foot trajectory has been developed. It allows the proximity sensor to detect the potential foothold. Also, the proximity sensor is used to control the footlift height during the transfer forward subphase in such a way that the foot trajectory follows terrain variations. Therefore, the use of proximity sensors for the leg transfer trajectory has been completely defined.

7.2 Research Extensions

As a result of this research, it is clear that the control algorithms developed need to be improved to some extent and several problems may need to be solved in order to actually implement the control algorithms on the actual vehicle (ASV-84). The following paragraphs describe the future work proposed.

One of the major aspects of the algorithms developed in this research is that they make use of periodic gaits. Since the optimal period is determined each computation cycle, it is constantly changing during locomotion. By freeing the cycle period so that it may continually change, the results of using periodic gaits tend to give the flexibility previously achieved only with free gaits. However, the algorithms for periodic gaits tend to be computaionally simpler and to date, have provided the only basis for control of an actual vehicle. An appropriate direction of future work may be to loosen the limitations imposed by using periodic gaits so as to gain the advantages of free gaits while at the same time continuing to take advantage of their ease of implementation.

The body regulation plan was designed in a simple manner. It needs more refinement in order to achieve better safety and stability of the vehicle. Since body attitude and altitude control is limited by the leg working volumes, it is desirable to find an optimal regulation plan based on the kinematic limits of the leg.

The use of proximity sensors for the body and leg motion planning has been investigated in this work through simulation and turns out to be quite useful in generating body and leg trajectories. There are a number of difficult problems involved in practically implementing proximity sensors and these issues need to be addressed. It is clear to the author, though, that the adaptability of the vehicle to rough terrain will improve considerably by using proximity sensors in future vehicles.

From the test results given in Section 6.3, it may be noted that the stability margin increases as the duty factor increases. However, a larger value of the leg duty factor requires a shorter transfer time which is mechanically limited. The desired result is that a large leg duty factor be used at low speeds but smaller values be used at high speeds so as that these speeds may be achieved. This suggests that if an automatic "gait shift" algorithm is implemented as a function of the vehicle speed, the mobility of the vehicle may be improved.

The deceleration plan developed is based on the simple notion of the transfer time threshold. To apply this scheme to the real vehicle, it should be modified so that the actual limitations in the acceleration and deceleration capability of the leg may be carefully considered.

One of the major problems that must be solved is to reduce the computation time for real-time control. Otherwise, the control may not be stable especially at high speeds. This problem may be solved by implementing the control software on a multicomputer system and investigations into this area should proceed.

An interesting approach to the development of the control algorithms for locomotion of walking vehicles that has first been used in this work, lies in the application of the Computer-Aided Design (CAD) techniques. The capability of the CAD technique developed in this research, though, is limited to the interactive change of two control parameters--duty factor and footlift height. Also, the capability of interactive terrain generation is limited in generating the simple prismatic terrain models. By extending the present CAD technique to the extent that future researchers may change many of the parameters related to the control algorithms and more complex terrain models, and interactively obtain results similar to those given in Section 6.3, the CAD technique will provide a versatile design tool in developing and evaluating control algorithms through computer simulation.

Future work was proposed in the proceeding paragraphs as a result of this dissertation work. Further extensions to this work will hopefully lead to more sophisticated and efficient control algorithms for rough-terrain locomtion of walking vehicles.

REFERENCES

1. Mosher, R.S., "Exploring the Potential of a Quadruped," SAE Paper No. 690191, International Automotive Engineering Conference, Detroit, MI, January, 1969.

2. McGhee, R.B., "Robot Locomotion," in Neural Control of Locomotion, Ed. by R.M. Herman, et al., Plenum Publishing Corp., New York, 1976, pp. 237-246.

3. Orin, D.E., Interactive Control of a Six-Legged Vehicle with Optimization of Both Stability and Energy, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, March, 1976.

4. Gurfinkel, V.S., et al., "Walking Robot with Supervisory Control," Mechanism and Machine Theory, Vol. 16, 1981, pp. 31-36.

5. Russell, M., "ODEX 1: The First Functionoid," Robotics Age, Vol. 5, No. 5, 1983, pp. 12-18.

6. Hirose, S., Nose, M., Kikuchi, H., and Umetani, Y., "Adaptive Gait Control of a Quadruped Walking Vehicle," Proc. of First International Symposium of Robotics Research, Aug., 1983.

7. McGhee, R.B., and Iswandhi, G.I., "Adaptive Locomotion of a Multilegged Robot over Rough Terrain," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-9, No. 4, April, 1979, pp. 176-182.

8. Orin, D.E., "Supervisory Control of a Multilegged Robot," International Journal of Robotics Research, Vol. 1. No. 1, Spring, 1982, pp. 79-91.

9. Okhotsimsky, D.E., and Platonov, A.K., "Control Algorithm of the Walker Climbing over Obstacles," Proc. Third Int. Conf. Artificial Intelligence, Stanford, pp. 317-323, Aug., 1973.

10. Orin, D.E., McGhee, R.B., and Jaswa, V.C., "Interactive Computer-Control of a Six-Legged Robot Vehicle with Optimization of Stability, Terrain Adaptability, and Energy," Proc. of 1976 IEEE Conference on Decision and Control, pp. 382-391, 1976.

11. Pugh, D.R., An Autopilot for a Terrain-Adaptive Hexapod Vehicle, M.S. thesis, The Ohio State University, Columbus, OH, 1982.

12. Chang, T.W., Motion Planning for Locomotion of A Multilegged Robot over Uneven Terrain, M.S. thesis, The Ohio State University, Columbus, OH, Dec., 1982.

13. Klein, C.A., Olson, K.W., and Pugh, D.R., "Use of Force and Attitude Sensors for Locomotion of A Legged Vehicle over Irregular Terrain," International Journal Of Robotics Research, Vol. 2, No. 2, Summer, 1983, pp. 3-17.

14. McGhee, R.B., Orin, D.E., Pugh, D.R., and Patterson, M.R., "A Hierarchically-Structured System for Computer Control of A Hexapod Walking Machine", Proc. of 5th IFToMM Symposium on Robots and Manipulator Systems, Udine, Italy, June, 1984.

15. Muybridge, E., Animals in Motion, Dover Publications, Inc., New York, 1957 (first published in 1899).

16. Muybridge, E., The Human Figure in Motion, Dover Publications, Inc., New York, 1955 (first published in 1901).

17. Hildebrand, M., "Symmetrical Gaits of Horses," Science, Vol. 150, pp. 701-708, November, 1965.

18. Anon., Menzi Muck Climbing Hoe, Climbing Hoe of America, Ltd., Fayettesville, Ga., 1981.

19. Anon., Kaiser Spyder Model X5M, Industrial and Municipal Engineering Corp., Galva, Illinois, 1982.

20. Waldron, K.J., Vohnout, V.J., Pery, A., and McGhee, R.B., "Configuration Design of the Adaptive Suspension Vehicle," to appear in Robotics Research, Spring 1984.

21. McGhee, R.B., "Some Finite State Aspects of Legged Locomotion," Mathematical Biosciences, Vol. 2, No, 1/2, pp. 67-84, Feb., 1968.

22. McGhee, R.B., and Jain, A.K., "Some Properties of Regularly Realizable Gait Matrices," Mathematical Biosciences, Vol. 13, No. 1/2, pp. 179-193, Feb., 1972.

23. McGhee, R.B., and Frank, A.A., "On the Stability of Quadruped Creeping Gaits," Mathematical Biosciences, Vol. 3, No. 3/4, pp. 331-353, Oct., 1968.

24. Bessonov, A.P., and Umnov, N.V., "The Analysis of Gaits in Six-Legged Vehicles According to Their Static Stability," Proc. of CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators, Udine, Italy, Sep., 1973.

25. McGhee, R.B., "Vehicular Legged Locomotion," in Advances in Automation and Robotics, ed. by G.N. Sardis, Jai Press, Inc., 1984.

26. Ozguner, F., Tsai, S.J., and McGhee, R.B., "Rough Terrain Locomotion by A Hexapod Robot Using A Binocular Ranging System," Proceedings of First International Symposium of Robotics Research, Bretton Woods, N.H., August 28, 1983.

27. Kugushev, E.I., and Jaroshevskij, V.S., "Problems of Selecting A Gait for An Integrated Locomotion Robot," Proc. Fourth Int. Conf. Artificial Intelligence, Tbilisi, Georgian SSR, USSR, pp. 789-793, Sep., 1975.

28. Kwak, S.H., A Simulation Study of Frre-Gait Algorithms for Omni-Directional Control of Hexapod Walking Machines, M.S. thesis, The Ohio State University, Columbus, OH, March, 1984.

29. Patterson, M.R., Reidy, J.J., and Brownstein, B.J., Guidance and Actuation Techniques for an Adaptively Controlled Vehicle, Final Technical Report, Battelle, Columbus, Ohio, March, 1983.

30. Klein, C.A., and Briggs, R.L., "Use of Active Compliance in the Control of Legged Vehicles," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-10, No. 7, July, 1980, pp. 393-400.

31. Tsai, S.J., An Experimental Study of a Binocular Vision System for Rough-Terrain Locomotion of a Hexapod Walking Robot, Ph.D. dissert-ation, The Ohio State University, Columbus, OH, March, 1983.

32. Whitney, D.E., "Force Feedback Control of Manipulator Fine Motions," Proc. of 1976 Joint Automatic Control Conference, Purdue University, West Lafayette, IN, July, 1976, pp. 687-693.

33. Orin, D.E., Tsai, C.K., and Cheng, F.T., "Dynamic Computer Control of A Robot Leg," IECON '83 San Fransisco, CA, Nov., 1983.

34. Vohnout, V.J., Mechanical Design of an Energy Efficient Robotic Leg for Use on a Multi-Legged Walking Vehicle, M.S. thesis, The Ohio State University, Columbus, Ohio, 1982.

35. Okhotsimsky, D.E., and Platonov, A.K., "Perspective Robot Moving In 3D World," Proc. Fourth Int. Conf. Attificial Intelligence, Tbilisi, Georgian SSR, USSR, pp. 789-802, Sep., 1975.

36. Paul, R.P., Robot Manipulators: Mathematics, Programming, and Control, M.I.T. Press, Cambridge, Massachusetts, 1981.

37. Walker, M.W., and Orin, D.E., "Efficient Dynamic Computer Simulation of Robotic Mechanisms," Journal of Dynamic Systems, Measurement, and Control, Vol. 104, Sep., 1982, pp. 205-211.

38. Broerman, K.R., Development of a Proximity Sensor System for Foot Altitude Control of a Terrain-Adaptive Hexapod Robot, M.S. thesis, The Ohio State University, Columbus, Ohio, June, 1983.

39. Strang, G., Linear Algebra and Its Applications, Academic Press, New York, N.Y., 1976.

40. Newman, W.M., and Sproull, R.F., Principles of Interative Computer Graphics, 2nd Edition, McGraw-Hill, Inc., N.Y., 1979.

41. Orin, D.E. and Pugh, D.R., "Hexapod Control Software Architecture," Technical Note No. 26, Digital Systems Lab., The Ohio State University, July, 1982.

42. Stevens, W.P., Myers, G.J., and Constantine, L.L., "Structured Design," IBM Systems Journal, Vol. 13, No. 2, 1974.

43. Crow, F.C., "Three-Dimensional Computer Graphics, Part 1," BYTE Magazine, BYTE Publication Inc., March, 1981, pp. 54-75.

APPENDIX

COMPUTER SIMULATION PROGRAMS

```
{*********************************************************************}
{ File : GLOBAL.PAS                                                   }
{ This file is for the global type and constant decleration.         }
{                                                                     }
{*********************************************************************}

type    Vector  = record
                     X,Y,Z : real;
                  end;
        Polygon = record
                     Start,PolyVtces: integer;
                  end;
        Plane   = record
                     UnitNormal: Vector; Dist : real;  { plane coefficients }
                  end;
        OnePlane= record
                     UnitNormal: Vector;{ plane coefficients of terrain }
                     Dist : real;
                     Xmin, Xmax: real;{ boundaries of a plane in X-direction }
                  end;
        Matrix = array [1..4] of Vector;
        Matrix4 = array [1..4,1..4] of real;
        Array6  = array [1..6] of real;
        Tactile = set of 1..6;
        Array3 =  array [0..2] of real;
        String = packed array[1..10] of char;
        Array6Vector = array[1..6] of Vector;
        Array52Vector = array[1..52] of Vector;
        ArrayMaxPtsVector = array[1..50] of Vector;
        ArrayMaxPolysPolygon = array[1..50] of Polygon;
        ArrayMaxPolysOnePlane = array[1..50] of OnePlane;
        ArrayMaxVtcesInteger = array[1..100] of integer;
        Array6Boolean = array[1..6] of boolean;


const   MaxPts   = 50;
        MaxPolys = 50;
        MaxVtces = 100;
        PI = 3.1415927;
        PI2 = 1.5707963;          { PI/2 }
        NominalHeight = 5.4;      { nominal body height in feet }
        Xhip = Array6( 5.0,5.0,0.0,0.0,-5.0,-5.0 ); { position of hip in feet }
        Yhip = 2.25;              { hip joint position }

        { dimension of rectangular hexahedron working volume }
        XL = 3.0;      YL = 2.2;     { for original WV }
        Zmax = -4.0;  Zmin = -6.8; { vertical kinematic limit in body }
        LiftPhase = 0.2;            { transfer phase value to begin liftoff }
        PlacePhase = 0.8;           { transfer phase value to begin placement }
```

```
{*******************************************************************}
{ File: MAINVAR.PAS                                                 }
{ This file is for the variable decleration for ASVMAIN.            }
{                                                                   }
{*******************************************************************}

const   VTMax = 1.5;              { max foot velocity in transfer }

type    Capitals = 'A'..'Z';
        CharSet  = set of Capitals;

var     BetaMode,                 { leg duty factor index        }
        I, J, FilNum
              : integer;
        Letter    :char;          { loop index                   }
        HaltSet   :CharSet;       { set of commands for which    }
                                  { vehicle halts afterwards     }
        Velocity
              : Array3;
        FPE
              : Vector;
        EyeSpace                  { homogeneous matrix from object to eye space }
              : Matrix4;
        H, INVH,                  { transform matrix from body to earth }
        G, INVG,                  { transform matrix from body to terrain }
        T, INVT                   { transform matrix from terrain to earth }
              : Matrix;
        Body                      { points of vehicle in body coord }
              : Array52Vector;
        Points                    { points of object in reference frame }
              : ArrayMaxPtsVector;
        Polygons
              : ArrayMaxPolysPolygon;
        Planes
              : ArrayMaxPolysOnePlane;
        Vertices
              : ArrayMaxVtcesInteger;
        RefPosB,                  { reference position of CWV }
        FPDE,                     { end point for foot trajectory in earth }
        FPSE,                     { six estimating points for support plane }
        FPB,                      { present foot position in body }
        FRB                       { present foot velocity }
              : Array6Vector;

        BodyTransRate,            { body translational rate      }
        BodyRotateRate            { body rotational rate         }
              : Vector;
        Contact                   { set of supporting legs       }
              : Tactile;
        NumPolys                  { total number of polygons to draw }
              : integer;
        dx, dy,                   { x and y dimensions of CWV    }
        RPhase,                   { relative leg phase           }
        LPhase                    { leg phase variable           }
              : Array6;
        Command                   { operator input command       }
              : char;
        Vmax, Vmin,
        dxmax, dxmin,
        dymax, dymin,
Beta,                     { leg duty factor                  }
DT,                       { delta time ( sec )               }
FootLift,                 { foot lifting height              }
NVELX, NVELY, NRVELZ,
Period,                   { period of kinematic cycle        }
Phase,                    { kinematic cycle phase            }
Etal                      { angle between gravity and        }
                          { terrain normal vector            }
        : real;
SPB,                      { support plane in body coord   }
SPE                       { support plane in earth coord  }
        : Plane;

EstimateFlag              { flag for estimating support plane }
        : boolean;

LiftoffFlag,              { flag for LiftOff phase }
TransferFlag,             { flag for Transferforward phase }
PlaceFlag                 { flag for Placement phase }
        : Array6Boolean;
```

```
{******************************************************************}
{ File: EXTERNAL.PAS                                                }
{ This file is for the external decleration.                        }
{                                                                   }
{******************************************************************}

function  Sign( X : real ) : real;
External;

function  Atan2(Y, X : real) : real;
External;

function ArcCos( S: real): real;
External;

function Tan( S: real ): real;
External;

function DotProd( Pt1,Pt2: Vector ): real;
External;

procedure VectSub( Pt1,Pt2: Vector; var Pt3: Vector );
External;

procedure CrossProd( Pt1,Pt2: Vector; var UnitNormal: Vector );
External;

procedure Ident( var Mtx : Matrix4 );           { identity matrix }
External;

procedure RotateMat( Axis : char; CosA,SinA: real; var Mtx : Matrix4 );
External;

procedure TransMat( X,Y,Z: real; var Mtx: Matrix4 );
External;

procedure Vect4Transform( Mt : Matrix4; Pt : Vector; var NewPt : Vector );
External;

procedure VectTransform( Mt : Matrix; Pt : Vector; var NewPt : Vector );
External;

procedure PlaneTransform( P : Plane; Mt : Matrix; var NewP : Plane );
External;

procedure Mat4Mult( Mt1,Mt2 : Matrix4; var Result : Matrix4 );
External;

procedure MatTranspose( Mt : Matrix; var TMt : Matrix );
External;

procedure MatMult( Mt1,Mt2 : Matrix; var Result : Matrix );
External;

procedure MatInverse( Mt : Matrix; var IMt : Matrix );
External;

procedure Orthogonalization( var Mt : Matrix );
External;
```

```
procedure ReadObject( var Filename : String;
                      var NumPolys : integer;
                      var Points : ArrayMaxPtsVector;
                      var Polygons : ArrayMaxPolysPolygon;
                      var Vertices : ArrayMaxVtcesInteger
                    );
External;

procedure MakeDisplayable( var Pt : Vector );
External;

procedure DisplayObject(       Mt : Matrix4;
                               DrawAgain : Boolean;
                           var Numpolys : integer;
                           var Points : ArrayMaxPtsVector;
                           var Polygons: ArrayMaxPolysPolygon;
                           var Vertices : ArrayMaxVtcesInteger
                         );
External;

PROCEDURE JOYSTICK(  VAR DATA   : ARRAY3;
                         SCALE  : REAL
                   );
External;

procedure WorkVolume(  var Polygons : ArrayMaxPolysPolygon;
                       var Vertices : ArrayMaxVtcesInteger;
                       var Body : Array52Vector
                     );
External;

procedure Vehicle(  var FPE,
                        FPDE : Vector;
                        PrxSensor,
                        ConSensor : boolean;
                    var Contact : Tactile;
                    var H : Matrix;
                    var dx,
                        dy : Array6;
                    var RefPos0 : Array6Vector;
                    var FPB : Array6Vector;
                    var EyeSpace : Matrix4;
                    var NumPolys : integer;
                    var Body : Array52Vector;
                    var Points : ArrayMaxPtsVector;
                    var Polygons : ArrayMaxPolysPolygon;
                    var Planes : ArrayMaxPolysOnePlane;
                    var Vertices : ArrayMaxVtcesInteger
                  );
External;

procedure SupportPattern( var H : Matrix;
                          var Points : ArrayMaxPtsVector;
                          var Contact : Tactile
                        );
External;

procedure EstimatePlane( var SupportPlane : Plane;
                         var FPSE : Array6Vector
```

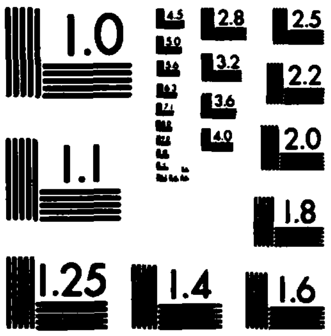
```
                                   );
          External;

          procedure TerrainCoord( var H,
                                      T,
                                      INVT,
                                      G,
                                      INVG : Matrix;
                                  var Etal : real;
                                  var SPE,
                                      SPB : Plane
                                 );
          External;

          procedure GetIntersection( var ClipXT,
                                         ClipXB,
                                         ClipYL,
                                         ClipYR,
                                         X1,
                                         Y1,
                                         X2,
                                         Y2 : real;
                                     var Error : integer
                                    );
          External;

          procedure OptimalPeriod(  var BodyTransRate,
                                        BodyRotateRate : Vector;
                                    var RefPosB,
                                        FPB : Array6Vector;
                                    var dx,
                                        dy,
                                        LPhase : Array6;
                                    var Contact : Tactile;
                                    var Beta,
                                        Period : real
                                  );
          External;

          procedure LegCoordination(  var H,
                                          INVH : Matrix;
                                      var SPB,
                                          SPE : Plane;
                                      var BodyTransRate,
                                          BodyRotateRate : Vector;
                                      var RefPosB,
                                          FPB : Array6Vector;
                                      var dx,
                                          dy,
                                          RPhase,
                                          LPhase : Array6;
                                      var Contact : Tactile;
                                      var Vmax, Vmin,
                                          dxmax, dxmin,
                                          dymax, dymin,
                                          DT,
                                          Beta,
                                          Phase,
                                          Period,
                                          SlowdownFactor : real
```


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

```
                                   );
External;

procedure BodyServo( var H,
                         INVH : Matrix;
                         var SPB,
                         SPE : Plane;
                     var BodyTransRate,
                         BodyRotateRate : Vector;
                         var RefPosB,
                         FPB : Array6Vector;
                     var dx,
                         dy,
                         RPhase,
                         LPhase : Array6;
                     var Contact : Tactile;
                     var Vmax, Vmin,
                         dxmax, dxmin,
                         dymax, dymin,
                         DT,
                         Beta,
                         Phase,
                         Period,
                         Etal,
                         NVELX,
                         NVELY,
                         NRVELZ : real
                    );
External;

procedure FootTrajectory(  var H,
                               INVH : Matrix;
                           var BodyTransRate,
                               BodyRotateRate : Vector;
                           var RefPosB,
                               FPDE,
                               FPSE,
                               FPB,
                               FRB : Array6Vector;
                           var dx,
                               dy,
                               LPhase : Array6;
                           var Contact : Tactile;
                           var FootLift, DT,
                               Beta,
                               Period : real;
                           var EstimateFlag : boolean;
                           var LiftoffFlag,
                               TransferFlag,
                               PlaceFlag : Array6Boolean;
                           var EyeSpace : Matrix4;
                           var NumPolys : integer;
                           var Body : Array52Vector;
                           var Points : ArrayMaxPtsVector;
                           var Polygons : ArrayMaxPolysPolygon;
                           var Planes : ArrayMaxPolysOnePlane;
                           var Vertices : ArrayMaxVtcesInteger
                         );
External;
```

```
procedure ASVINIT(  var H,
                        INVH : Matrix;
                    var dx,
                        dy : Array6;
                    var Contact : Tactile;
                    var dxmin,
                        dymin,
                        DT,
                        FootLift : real;
                    var BetaMode : integer;
                    var RefPosB,
                        FPDE,
                        FPSE,
                        FPB : Array6Vector;
                    var EyeSpace : Matrix4;
                    var NumPolys : integer;
                    var Body : Array52Vector;
                    var Points : ArrayMaxPtsVector;
                    var Polygons : ArrayMaxPolysPolygon;
                    var Planes : ArrayMaxPolysOnePlane;
                    var Vertices : ArrayMaxVtcesInteger
                 );
External;

procedure Halt(  var NVELX,
                     NVELY,
                     NRVELZ : real;
                 var BodyTransRate,
                     BodyRotateRate : Vector;
                 var FRB : Array6Vector
               );
External;

procedure Initialize(  var H : Matrix;
                       var dx,
                           dy,
                           RPhase,
                           LPhase : Array6;
                       var RefPosB,
                           FPDE,
                           FPE : Array6Vector;
                       var FootLift, Phase,
                           Period,
                           Beta : real;
                       var Contact : Tactile;
                       var EstimateFlag : boolean;
                       var LiftoffFlag,
                           PlaceFlag : Array6Boolean;
                       var EyeSpace : Matrix4;
                       var NumPolys : integer;
                       var Body : Array52Vector;
                       var Points : ArrayMaxPtsVector;
                       var Polygons : ArrayMaxPolysPolygon;
                       var Planes : ArrayMaxPolysOnePlane;
                       var Vertices : ArrayMaxVtcesInteger
                    );
External;

procedure KEYINT; NonPascal;
procedure KEYCX( var Command : char ); NonPascal;

procedure KYWAIT; NonPascal;

procedure ICKILL; NonPascal;

procedure INQUE; NonPascal;

procedure NamFil( var FilNum : integer ); NonPascal;

procedure StpNam; NonPascal;

procedure Moveto( var IX,IY : integer ); NonPascal;

procedure Drawto( var IX,IY : integer ); NonPascal;

procedure Stover( var FilNum : integer ); NonPascal;
```

```
(****************************************************************************)
( File: ASVMAIN.PAS                                                          )
( This file is for the main and the man-machine interface.                  )
(                                                                            )
(****************************************************************************)

program ASVMAIN;

%include Global;
%include external;
%include Mainvar;


begin   ( main program )

( parameter initialization )

ASVINIT( M,INVM,dx,dy,Contact,dxmin,dymin,DT,FootLift,
         BetaMode,RefPosB,FPDE,FPSE,FPB,EyeSpace,NumPolys,
         Body,Points,Polygons,Planes,Vertices );

HaltSet      := ['M','N','I','H','X'];

( set terminal characteristics )

write( CHR(27), '[?6h' );        ( set orign to relative mode )
write( CHR(27), '[?4l' );        ( set to noscroll )
write( CHR(27), '[2J' );         ( erase screen )
write( CHR(27), '[1;1H' );       ( cursor to home)
write( CHR(27), '[2J' );         ( erase screen )
write( CHR(27), '[1;1H' );       ( cursor to home )

( display command set )
writeln;
writeln;
writeln( CHR(27), '#6***** COMMANDS *****');
writeln;
writeln('Enter ', CHR(27),'#6I',' TO INITIALIZE THE LEG POSITIONS');
writeln('      ', CHR(27),'#6S',' TO START MOTION');
writeln('      ', CHR(27),'#6H',' TO HALT MOTION');
writeln('      ', CHR(27),'#6M',' TO MODIFY THE PARAMETERS');
writeln('      ', CHR(27),'#6X',' TO EXIT THE PROGRAM EXECUTION');
writeln;
writeln;
writeln;
writeln( CHR(27),'#6***** REAL TIME OPERATION *****');
write( CHR(27), '[18;22r');      ( set scrolling region )

Command := 'M';  ( enter modify sequence initially )
KEYINT;

repeat ( until Command = 'X' )

    if Command in HaltSet
       then
          begin
          case Command of

              'I':    begin
                      Initialize( M,dx,dy,RPhase,LPhase,RefPosB,FPDE,
                           FPS,FootLift,Phase,Period,Beta,
                           Contact,EstimateFlag,LiftoffFlag,
                           PlaceFlag,EyeSpace,NumPolys,Body,
                           Points,Polygons,Planes,Vertices );

                Halt ( NVELX,NVELY,NRVELZ,BodyTransRate,
                       BodyRotateRate,FR5 );
                end;

        'H':    Halt ( NVELX,NVELY,NRVELZ,BodyTransRate,
                       BodyRotateRate,FR8 );

        'M':    begin
                writeln;
                writeln( 'DOES BETAMODE = ', BetaMode:1,
                         ' NEED TO BE CHANGED?');
                write('   TYPE A SPACE BEFORE [Y/N]:');
                IOKILL; ( cancel input request )
                readln( Letter );

                if Letter = 'Y' then
                   repeat
                   writeln('PLEASE ENTER DESIRED BETAMODE:');
                   writeln(' 1:  BETA = 5/6        2:  BETA = 3/4');
                   writeln(' 3:  BETA = 2/3        4:  BETA = 1/2');
                   writeln(' 5:  BETA = 11/12 ');
                   readln(BetaMode);
                   until BetaMode in [1,2,3,4,5];

                case BetaMode of
                  1: begin    ( Beta = 5/6 )
                     Vmax := 1.25; dxmax := 3.0;  dymax := 2.2;
                     Vmin := 0.25; dxmin := 0.75;  dymin := 0.75;
                     end;
                  2: begin    ( Beta = 3/4 )
                     Vmax := 1.5; dxmax := 3.0;  dymax := 2.2;
                     Vmin := 0.25; dxmin := 0.5;  dymin := 0.5;
                     end;
                  3: begin    ( Beta = 2/3 )
                     Vmax := 2.0; dxmax := 3.0;  dymax := 2.2;
                     Vmin := 0.25; dxmin := 0.25;  dymin := 0.25;
                     end;
                  4: begin    ( Beta = 1/2 )
                     Vmax := 3.0; dxmax := 3.0;  dymax := 2.2;
                     Vmin := 0.25; dxmin := 0.25;  dymin := 0.25;
                     end;
                end; ( case end )


                case BetaMode of
                    1:   Beta := 0.8333;  ( 5/6 )
                    2:   Beta := 0.75;    ( 3/4 )
                    3:   Beta := 0.6667;  ( 2/3 )
                    4:   Beta := 0.5;     ( 1/2 )
                    5:   Beta := 0.916    (11/12)
                end;( case )

                ( assign relative leg phase )
                RPhase[1] := 0.0;   RPhase[2] := 0.5;
                RPhase[3] := Beta;  RPhase[4] := Beta - 0.5;
```

```
                    RPhase[5] := 2 * Beta - 1.0; RPhase[6]:= RPhase[5]+0.5;
                    RPhase[6] := RPhase[6] - trunc(RPhase[6]);

                    write( 'Change FootLift[Y/N]? ' );
                    readln( Letter );
                    if Letter = 'Y' THEN
                       repeat
                       writeln( 'Enter new value of FootLift');
                       readln( FootLift );
                       until (FootLift >= 1.0) and (FootLift <= 3.0);



                    INQUE; ( REASSERT INPUT REQUEST )
                    writeln;
                    writeln( CHR(27), '#6INITIALIZE HEXAPOD ');
                    end;

                end; ( case Command )

            KYWAIT;
            KEYCK(Command);
            end ( then )

        else
            begin
            Joystick( Velocity, VTmax);
            NVELX := Velocity[1];
            NVELY := -Velocity[0];
            NRVELZ := Velocity[2] * 0.1;

            if EstimateFlag
                then begin
                     EstimatePlane( SPE,FPSE );
                     EstimateFlag := false;
                     end;

            TerrainCoord( H,T,INVT,G,INVG,Etal,SPE,SPB );

            BodyServo( H,INVH,SPB,SPE,BodyTransRate,BodyRotateRate,
                       RefPosB,FPB,dx,dy,RPhase,LPhase,Contact,
                       Vmax,Vmin,dxmax,dxmin,dymax,dymin,
                       DT,Beta,Phase,Period,Etal,NVELX,NVELY,NRVELZ );

            FootTrajectory( H,INVH,BodyTransRate,BodyRotateRate,
                            RefPosB,FPDE,FPSE,FPB,FRB,dx,dy,
                            LPhase,Contact,FootLift,DT,Beta,Period,
                            EstimateFlag,LiftOffFlag,TransferFlag,
                            PlaceFlag,EyeSpace,NumPolys,Body,
                            Points,Polygons,Planes,Vertices );

            KEYCK( Command );
            end; ( else )

until Command = 'X';

ICKILL;
write ( CHR(27), '[?6l');       ( set origin to absolute mode )
write ( CHR(27), '[?4h');       ( set to smooth scroll )
write ( CHR(27), '[1;22r');     ( restore scrolling region )
write ( CHR(27), '[2J');        ( erase screen )
write ( CHR(27), '[1;1H');      ( cursor to home )

end.    ( end of ASVMAIN )
```

```
{$nomain}
{$own}
{****************************************************************************}
{ File: LIBR.PAS                                                             }
{ This file is library for ASVMAIN.                                          }
{                                                                            }
{****************************************************************************}

%include global;
%include external;

function  Sign( X : real ) : real;
begin
     if X >= 0.0
          then Sign := 1.0
          else Sign := -1.0
end;     { Sign }


function  Atan2(Y, X : real) : real;

begin

if abs(X) > 0.00001
     then   { X is nonzero }
          if X > 0.0
               then
                    Atan2 := ArcTan(Y/X)
               else
                    Atan2 := ARCTan(Y/X) + PI * Sign( Y )
          { end if X }
     else
          Atan2 := PI / 2.0 * Sign( Y );

end;   { Atan2 }


function ArcCos( S: real): real;
var      C : real;
begin
C := sqrt(1.0 - S*S);
ArcCos := Atan2( C,S);
end;     { ArcCos }


function Tan( S: real ): real;
begin
Tan := Sin( S )/ Cos( S );
end;


function DotProd( Pt1,Pt2: Vector ): real;
begin
         with Pt1 do
              DotProd := X*Pt2.X + Y*Pt2.Y + Z*Pt2.Z;
end;     { DotProd }


procedure VectSub( Pt1,Pt2: Vector; var Pt3: Vector );
{ vector sbtraction of Pt2 from Pt1 }
```

```
begin
        with Pt3 do
             begin
             X := Pt1.X - Pt2.X;
             Y := Pt1.Y - Pt2.Y;
             Z := Pt1.Z - Pt2.Z;
             end;
end;    { VectSub }


procedure CrossProd( Pt1,Pt2: Vector; var UnitNormal: Vector );
{ generate unitnormal vector by (Pt1 X Pt2) }

var     M : real;
begin
        with UnitNormal do
             begin
             X := Pt1.Y*Pt2.Z - Pt1.Z*Pt2.Y;
             Y := Pt1.Z*Pt2.X - Pt1.X*Pt2.Z;
             Z := Pt1.X*Pt2.Y - Pt1.Y*Pt2.X;
             M := sqrt( X*X + Y*Y + Z*Z );
             X := X/M; Y := Y/M; Z := Z/M;
             end;
end;    { CrossProd }


procedure Ident( var Mtx : Matrix4 );   { identity matrix }
var     I,J : integer;
begin
for I := 1 to 4 do
        for J := 1 to 4 do
                if I = J then Mtx[I,J] := 1.0
                         else Mtx[I,J] := 0.0;
end;    { Ident }


procedure RotateMat( Axis : char; CosA,SinA: real; var Mtx : Matrix4 );
begin                   { get rotation matrix of a given axis }
Ident(Mtx);
case  Axis of
        'X' : begin
              Mtx[2,2] := CosA; Mtx[2,3] := -SinA;
              Mtx[3,2] := SinA; Mtx[3,3] := CosA;
              end;
        'Y' : begin
              Mtx[1,1] := CosA; Mtx[1,3] := SinA;
              Mtx[3,1] := -SinA; Mtx[3,3] := CosA;
              end;
        'Z' : begin
              Mtx[1,1] := CosA; Mtx[1,2] := -SinA;
              Mtx[2,1] := SinA; Mtx[2,2] := CosA;
              end;
    end;        { case end }
end;    { RotateMat }


procedure TransMat( X,Y,Z: real; var Mtx: Matrix4 );
begin                   { get translation matrix }
Ident( Mtx );
Mtx[1,4] := X; Mtx[2,4] := Y; Mtx[3,4] := Z;
end;    { TransMat }
```

```
procedure Vect4Transform( Mt : Matrix4; Pt : Vector; var NewPt : Vector );
{ NewPt = Mt * Pt }
begin
NewPt.X := Pt.X*Mt[1,1] + Pt.Y*Mt[1,2] + Pt.Z*Mt[1,3] + Mt[1,4];
NewPt.Y := Pt.X*Mt[2,1] + Pt.Y*Mt[2,2] + Pt.Z*Mt[2,3] + Mt[2,4];
NewPt.Z := Pt.X*Mt[3,1] + Pt.Y*Mt[3,2] + Pt.Z*Mt[3,3] + Mt[3,4];
end;     { Vector4Transform }


procedure VectTransform( Mt : Matrix; Pt : Vector; var NewPt : Vector );
{ NewPt = Mt * Pt }
begin
NewPt.X := Pt.X*Mt[1].X + Pt.Y*Mt[2].X + Pt.Z*Mt[3].X + Mt[4].X;
NewPt.Y := Pt.X*Mt[1].Y + Pt.Y*Mt[2].Y + Pt.Z*Mt[3].Y + Mt[4].Y;
NewPt.Z := Pt.X*Mt[1].Z + Pt.Y*Mt[2].Z + Pt.Z*Mt[3].Z + Mt[4].Z;
end;     { VectTransform }


procedure PlaneTransform( P : Plane; Mt : Matrix; var NewP : Plane );
{ NewP = P * Mt }
var A, B, C, D, M : real;
begin

A := P.UnitNormal.X*Mt[1].X + P.UnitNormal.Y*Mt[1].Y
               +P.UnitNormal.Z*Mt[1].Z ;
B := P.UnitNormal.X*Mt[2].X + P.UnitNormal.Y*Mt[2].Y
               +P.UnitNormal.Z*Mt[2].Z ;
C := P.UnitNormal.X*Mt[3].X + P.UnitNormal.Y*Mt[3].Y
               +P.UnitNormal.Z*Mt[3].Z ;
D := P.UnitNormal.X*Mt[4].X + P.UnitNormal.Y*Mt[4].Y
               +P.UnitNormal.Z*Mt[4].Z + P.Dist;
M := sqrt( A*A + B*B + C*C );
with NewP do
        begin
        with UnitNormal do
                begin X := A/M; Y := B/M; Z := C/M; end;
        Dist := D/M;
        end;
end;     { PlaneTransform }


procedure Mat4Mult( Mt1,Mt2 : Matrix4; var Result : Matrix4 );
var      I,J,K : integer;
begin
for I := 1 to 4 do       { Mt1 X Mt2 }
        for J := 1 to 4 do
                begin
                Result[I,J] := 0.0;
                for K := 1 to 4 do
                Result[I,J] := Result[I,J] + Mt1[I,K]*Mt2[K,J];
                end;
end;     { Mat4Mult }


procedure MatTranspose( Mt : Matrix; var TMt : Matrix );
begin
        with TMt[1] do
```

```
                begin
                X := Mt[1].X; Y := Mt[2].X; Z := Mt[3].X;
                end;
          with TMt[2] do
                begin
                X := Mt[1].Y; Y := Mt[2].Y; Z := Mt[3].Y;
                end;
          with TMt[3] do
                begin
                X := Mt[1].Z; Y := Mt[2].Z; Z := Mt[3].Z;
                end;
          with TMt[4] do
                begin
                X := Mt[4].X; Y := Mt[4].Y; Z := Mt[4].Z;
                end;
end;      ( MatTranspose )


procedure MatMult( Mt1,Mt2 : Matrix; var Result : Matrix );
var       TMt : Matrix;
          I   : integer;
begin
          MatTranspose( Mt1,TMt );
          for I := 1 to 4 do
                 with Result[I] do
                         begin
                         X := DotProd( Tmt[1],Mt2[I] );
                         Y := DotProd( Tmt[2],Mt2[I] );
                         Z := DotProd( TMt[3],Mt2[I] );
                         if I = 4
                              then begin
                                   X := X + TMt[I].X;
                                   Y := Y + Tmt[I].Y;
                                   Z := Z + Tmt[I].Z;
                                   end;
                         end;
end;      ( MatMult )


procedure MatInverse( Mt : Matrix; var IMt : Matrix );
begin
          MatTranspose( Mt,IMt );
          with IMt[4] do
                 begin
                 X := -DotProd( Mt[4],Mt[1] );
                 Y := -DotProd( Mt[4],Mt[2] );
                 Z := -DotProd( Mt[4],Mt[3] );
                 end;
end;      ( MatInverse )


procedure Orthogonalization( var Mt : Matrix );
( use Gram-Schimit orthogonalization process )
var       A, M     : real;
begin
          with Mt[1] do           ( first orthogonal vector )
                begin
                M := sqrt( X*X + Y*Y + Z*Z );
                X := X/M; Y := Y/M; Z := Z/M;
                end;
                A := DotProd( Mt[1],Mt[2] );
                with Mt[2] do           { second orthogonal vector }
                     begin
                     X := X - A*Mt[1].X;
                     Y := Y - A*Mt[1].Y;
                     Z := Z - A*Mt[1].Z;
                     M := sqrt( X*X + Y*Y + Z*Z );
                     X := X/M; Y := Y/M; Z := Z/M;
                     end;
                CrossProd( Mt[1],Mt[2],Mt[3] ); { third orthogonal vector }
        end;
```

```
{$nomain}
{$own}
{*******************************************************************}
{ File: DIPLAY.PAS                                                  }
{ This file is for graphics display routines.                       }
{                                                                   }
{*******************************************************************}

%include Global;
%include External;

procedure ReadObject( var Filename : String;
                      var NumPolys : integer;
                      var Points : ArrayMaxPtsVector;
                      var Polygons : ArrayMaxPolysPolygon;
                      var Vertices : ArrayMaxVtcesInteger
                     );

var     I, J , NumVtces, NumPts : integer;
        F : text;
begin
reset(F, Filename, '.DAT');     { open input file }
readln(F,NumPts, NumPolys); { read in points }
for I := 1 to NumPts do
        with Points[I] do
                readln(F, J, X, Y, Z);
NumVtces := 0;                  { initialize size of vertex array }
for I := 1 to NumPolys do       { read in polygon descriptions }
        with Polygons[I] do
                begin
                start := NumVtces;      { start point in vertex array }
                read(F, PolyVtces );    { number of vertices }
                for J := 1 to PolyVtces do      { read vertex pointers }
                        read(F, Vertices[ NumVtces + J ]);
                readln(F);              { go to next line of input }
                NumVtces := NumVtces + PolyVtces;
                end;
Close(F);       { close the file }

end;    { ReadObject }


procedure MakeDisplayable( var Pt : Vector );

const   Middle = Vector( 500.0,500.0,0.0 );
        Scale = 4000.0;

begin           { perspective projection }
Pt.X := Scale * Pt.X/Pt.Z + Middle.X;
Pt.Y := Scale * Pt.Y/Pt.Z + Middle.Y;
end;    { MakeDisplayable }


procedure DisplayObject(     Mt : Matrix4;
                              DrawAgain: Boolean;
                         var Numpolys: integer;
                         var Points : ArrayMaxPtsVector;
                         var Polygons: ArrayMaxPolysPolygon;
                         var Vertices : ArrayMaxVtcesInteger
                        );
```

```
var     TmpPt   : Vector;
        I,J,K,
        IX,IY   : integer;
        DisplayPts
                : array[1..10] of record X,Y: integer; end;

begin
for I := 1 to NumPolys do
        with Polygons[I] do
                begin
                Vect4Transform( Mt, Points[Vertices[Start+PolyVtces]],TmpPt);
                MakeDisplayable( TmpPt );
                with TmpPt do
                        begin
                        IX := trunc(X); IY := trunc(Y); Moveto(IX,IY);
                        end;
                for J := 1 to PolyVtces do
                        begin
                        Vect4Transform( Mt,Points[Vertices[Start+J]],TmpPt );
                        MakeDisplayable( TmpPt );
                        with TmpPt do
                                begin
                                IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
                                end;
                        if DrawAgain
                                then with DisplayPts[J] do
                                        begin X := IX; Y := IY; end;
                        end;
                if DrawAgain
                        then
                        begin
                        with DisplayPts[PolyVtces] do
                                     Moveto(X,Y);
                        for J := 1 to PolyVtces do
                                with DisplayPts[J] do
                                     Drawto(X,Y);
                        end;
                end;
end;    { DisplayObject }
```

```
{$nomain}
{$own}
{*********************************************************************}
{ File: JOYSTICK.PAS                                                  }
{ This is for the joystick interface routine.                         }
{                                                                     }
{*********************************************************************}

%include Global;
%include External;

PROCEDURE JOYSTICK(  VAR DATA   : ARRAY3;
                         SCALE  : REAL
                  );


VAR       ADSR    ORIGIN 170400B  : INTEGER;       { A/D STATUS REGISTER }
          ADBUF   ORIGIN 170402B  : INTEGER;       { A/D BUFFER REGISTER }

          CHANNEL                 : INTEGER;       { CHANNEL INDEX }
          INTEGERDATA             : INTEGER;       { A/D INTEGER DATA }


BEGIN { JOYSTICK }

FOR CHANNEL := 0 TO 2 DO
    BEGIN

    ADSR := CHANNEL * 400B;  { SELECT MULTIPLEXER CHANNEL }

    ADSR := ADSR + 1;  { SET CONVERSION COMMAND BIT }

    WHILE ADSR AND 200B = 0 DO  { WAIT FOR CONVERSION };

    INTEGERDATA := ADBUF + 177000B;  { CONVERT TO 2'S COMPLEMENT FORMAT }

    DATA[ CHANNEL ] := INTEGERDATA / 1000B * SCALE; { -SCALE <= DATA < SCALE }

    END;

END; { JOYSTICK }
```

```
{$nomain}
{$own}
{************************************************************************}
{ File: VEHICLE.PAS                                                      }
{ This file is for drawing the ASV-84, working volumes, and support      }
{ pattern of the vehicle motion.                                         }
{                                                                        }
{************************************************************************}

%include Global;
%include External;

procedure Vehicle( var FPE,
                       FPDE : Vector;
                       PrxSensor,
                       ConSensor : boolean;
                   var H : Matrix;
                   var dx,
                       dy : Array6;
                   var RefPosB : Array6Vector;
                   var FPB : Array6Vector;
                   var EyeSpace : Matrix4;
                   var NumPolys : integer;
                   var Body : Array52Vector;
                   var Points : ArrayMaxPtsVector;
                   var Polygons : ArrayMaxPolysPolygon;
                   var Planes : ArrayMaxPolysOnePlane;
                   var Vertices : ArrayMaxVtcesInteger
                 );

var     FilNum  : integer;


procedure ProximitySensor( FPE: Vector; var FPDE: Vector );
{ get leg touchdown point in earth for leg in placement phase    }

var     J    : integer;

begin

J := 1;
{ find plane index where foot is on and return index }
while not( FPE.X <= Planes[J].Xmax ) do
            J := J + 1;

with Planes[J] do       { get touchdown point in earth coord }
        with UnitNormal do
                FPDE.Z := -( X*FPE.X + Y*FPE.Y + Dist ) / Z;

with FPDE do
        begin
        X := FPE.X;  Y := FPE.Y;
        end;

end;    { ProximitySensor }


procedure ContactSensor( var FPE: vector);
{ FPE.Z is modified so that foot tip can contact the ground     }
```

```
var     I,J
                : integer;
        D       : real;

begin

        J := 1;                 { index for array of Planes }
        { find plane index where foot is on and return index }
        while not( FPE.X <= Planes[J].Xmax ) do
                J := J + 1;

        with Planes[J] do       { compute distance from foot to plane }
                with UnitNormal do
                     FPE.Z:=-(X*FPE.X+Y*FPE.Y+Dist)/Z;

end;    { ContactSensor }


procedure Kinematics;

{ reverse kinematics equation of ASV84 vehicle                  }
{ Z axis goes upward in body and earth coordinate               }

const   L1 = 0.8; L2 = 4.0; L3 = 1.45;
        L4 = 0.5;       { leg demensions in feet }

var     C1, S1,
        CosA, SinA,
        D, D1, D2,
        Sign1, Sign2,
        Alpha, Beta, Gamma,
        Theta, T1, T2,
        XPC, YPC, ZPC
                : real;
        I, J    : integer;


begin   { Kinematics }

for I := 1 to 16 do                     { transform points of body }
        VectTransform( H,Body[I],Points[I] );

for I := 1 to 6 do                      { tranform points of legs }
        begin
        if I in [1,3,5] then Sign1 := 1.0 else Sign1 := -1.0;
        if I in [1,4,6] then Sign2 := -1.0 else Sign2 := 1.0;
        { compute abduction angle, translations of leg }
        XPC := FPB[I].X - Xhip[I];
        YPC := FPB[I].Y - Sign1 * Yhip;
        ZPC := FPB[I].Z;
        T1  := sqrt( YPC*YPC + ZPC*ZPC - L4*L4 );
        { abduction or adduction  angles }
        Theta := Atan2( Sign1*YPC, -Sign1*ZPC ) - Atan2( L4,Sign1*T1 );
        C1 := Cos( Theta );  S1 := Sin( Theta );

        case I of
           1,2      : D2 := 0.2 * XPC;
          3,4,5,6   : D2 := -0.2 * XPC;
         end;   { case end }
        T2 := 0.25 * ( -YPC*S1 + ZPC*C1 + L3 );
        D1 := Sign2 * T2;

        Alpha := Atan2( D2,-Sign2*D1 );
        D := 0.5 * sqrt( D1*D1 + D2*D2 ) / L1;
        Beta := ArcCos( D );
        Gamma := -Sign2 * (PI2 -  Alpha - Beta );  { angle between sys2 & 4 }
        CosA := Cos( Gamma );  SinA := Sin( Gamma );

        J := (I-1) * 3;
        with Body[17+J] do               { updown sliding joint position }
             begin
             X := Xhip[I];
             Y := Sign2*D1*S1 + L3*S1 + Sign1*Yhip;
             Z := -Sign2*D1*C1 - L3*C1;
             end;

        with Body[18+J] do               { knee position }
             begin
             if I in [1,2] then X := L2*CosA + Xhip[I]
                           else X :=-L2*CosA + Xhip[I];
             Y := -Sign2*L2*S1*SinA + Sign2*D1*S1 + L3*S1 + Sign1*Yhip;
             Z := Sign2*L2*C1*SinA - Sign2*D1*C1 - L3*C1;
             end;

        VectTransform( H,Body[17+J],Points[17+J] );(transform in base system}
        VectTransform( H,Body[18+J],Points[18+J] );
        VectTransform( H,FPB[I],Points[19+J] );

        end;    { end of loop for I }

end;    { Kinematics }


procedure DrawLeg;
{ draw a segment from hip joint to foot tip }

const   Middle = Vector( 300.0,800.0,0.0 );
        Scale  = 15.0;

var     I, IX, IY: integer;
        Xwidth, Ywidth : real;


procedure DashLine( X1,Y1,X2,Y2: real;
                    Horizontal : boolean );
const   N = 5;

var     I, IX1, IY1, IX2, IY2
                     : integer;
        Int     : real;

begin
        if Horizontal
             then begin
                  IX1 := trunc( Scale * X1 + Middle.X );
                  IY1 := trunc( Scale * Y1 + Middle.Y );
                  end
             else begin
                  IY1 := trunc( Scale * X1 + Middle.Y );
```

```
                        IX1 := trunc( Scale * Y1 + Middle.X );
                        end;
        Moveto( IX1,IY1 );

        Int := ( X2 - X1 ) / N;
        for I := 1 to N do
              begin
              X1 := X1 +  Int;
              if Horizontal
                    then begin
                         IX2 := trunc( Scale * X1 + Middle.X );
                         if odd( I ) then Drawto( IX2,IY1 )
                                     else Moveto( IX2,IY1 );
                         end
                    else begin
                         IY2 := trunc( Scale * X1 + Middle.Y );
                         if odd( I ) then Drawto( IX1,IY2 )
                                     else Moveto( IX1,IY2 );
                         end;
              end;
end;    ( DashLine )


procedure DrawXlimit( Center: Vector; Xwidth,Ywidth : real;
                      DrawVolume : boolean );

var     XB, XT, YL, YR
              : real;
        IXB, IXT, IYL, IYR : integer;

begin   ( DrawXlimit )
        with Center do
             begin
             XB := X - Xwidth; YL := Y - Ywidth;
             XT := X + Xwidth; YR := Y + Ywidth;
             end;
        if DrawVolume
             then begin
                  DashLine( XB,YL,XT,YL,true );
                  DashLine( YL,XT,YR,XT,false );
                  DashLine( XT,YR,XB,YR,true );
                  DashLine( YR,XB,YL,XB,false );
                  end
             else begin
                  IXB := trunc( Scale*XB + Middle.X);
                  IYL := trunc( Scale*YL + Middle.Y);
                  IXT := trunc( Scale*XT + Middle.X);
                  IYR := trunc( Scale*YR + Middle.Y);
                  Moveto( IXB,IYL );
                  Drawto( IXT,IYL );
                  Drawto( IXT,IYR );
                  Drawto( IXB,IYR );
                  Drawto( IXB,IYL );
                  end;
end;             ( DrawXlimit )


begin   ( DrawLeg )
        for I := 1 to 6 do
            begin
```

```
                Xwidth := 0.5 * dx[I]; Ywidth := 0.5 * dy[I];
                DrawXlimit( RefPosS[I], Xwidth, Ywidth, true );

                if I in Contact
                    then begin
                         if I in [1,3,5] then IY := trunc( Scale*Yhip + Middle.Y )
                                         else IY := trunc(-Scale*Yhip + Middle.Y );
                         IX := trunc( Scale*Xhip[I] + Middle.X );
                         Moveto( IX,IY );
                         IX := trunc( Scale*FPB[I].X + Middle.X );
                         IY := trunc( Scale*FPB[I].Y + Middle.Y );
                         Drawto( IX,IY );
                         end
                    else begin      ( put dummy vector into display buffer )
                         IX := 0; IY := 0; Moveto( IX,IY ); Moveto( IX,IY );
                         end;
                end;
end;    ( DrawLeg )


begin   ( procedure Vehicle body )

if PrxSensor then ProximitySensor( FPE,FPOE );

if ConSensor then ContactSensor( FPE );

if not( PrxSensor or ConSensor )
        then begin
              FilNum :=2; Stover( FilNum );
              Kinematics;
              DisplayObject( EyeSpace,true,NumPolys,Points,
                                   Polygons,Vertices );
              DrawLeg;   ( draw support legs in topview of vehicle )
              StpNam;
        end;

end;    ( Vehicle )


procedure SupportPattern( var M : Matrix;
                              var Points : ArrayMaxPtsVector;
                              var Contact : Tactile
                            );

( Draw support pattern in earth coordinate                         )

const   Middle = Vector( 700.0, 800.0, 0.0 );
        Scale  = 15.0;

var     SupLeg
                : array[1..6] of Vector;
        CntrPress, Line
                : Vector;
        I, J, K, M, L,
        IX, IY, FilNum
                : integer;
        D
                : real;
```

```
procedure DrawRectangle( X,Y,DIST : integer; Flag : boolean );
var       XL,XR,YB,YT        : integer;

begin     { DrawRectangle }

          XL := X - DIST; XR := X + DIST;
          YB := Y - DIST; YT := Y + DIST;
          Moveto( XL,YB );
          if Flag
                then begin         { draw rectangle }
                     Drawto( XL,YT ); Drawto( XR,YT );
                     end
                 else Drawto( X,YT );      { draw triangle }
          Drawto( XR,YB ); Drawto( XL,YB );
end;      { DrawRectangle }


procedure GetLines( Pt1,Pt2: Vector; var Line: Vector );
{ get coefficients of equation of a line.      }
{ line vector starts from Pt1 and ends to Pt2. }

begin     { GetLines }
          with Line do
                begin
                X := Pt2.Y - Pt1.Y;
                Y := Pt1.X - Pt2.X;
                Z := Pt1.Y * Pt2.X - Pt1.X * Pt2.Y;
                end;
end;      { GetLines }


procedure StabilityMargin( Center: Vector; K: integer );

var       MinDist, Dist, D, E,
          Xint, Yint, Xmin, Ymin
                : real;
          IX, IY, I
                : integer;
          Line     : Vector;

begin     { StabilityMargin }
          MinDist := 1000.0;
          for I := K downto 1 do
              begin
              if I = 1 then GetLines( SupLeg[1],SupLeg[K],Line )
                       else GetLines( SupLeg[I],SupLeg[I-1],Line );
              with Line do
                  begin
                  D := X*X + Y*Y;
                  E := X*Center.Y - Y*Center.X;
                  Xint := ( -X*Z - Y*E )/D;
                  Yint := ( -Y*Z + X*E )/D;
                  end;
              Dist := DotProd( Center,Line )/sqrt(D);
              if Dist < MinDist   { get minimum distance and intersection }
                  then begin
                       MinDist := Dist; Xmin := Xint; Ymin := Yint;
                       end;
```

```
            end;
        with Center do          { draw stability margin }
            begin
            IX := trunc( Scale*X + Middle.X );
            IY := trunc( Scale*Y + Middle.Y );
            Moveto( IX,IY );
            end;
         IX := trunc( Scale*Xmin + Middle.X );
         IY := trunc( Scale*Ymin + Middle.Y );
         Drawto( IX,IY );
end;    { StabilityMargin }


begin   { procedure SupportPattern body }
        FilNum := 5; NamFil( FilNum ); Stover( FilNum );

        IX := trunc(Middle.X);
        IY := trunc(Middle.Y);          { center of body in body coord }
        DrawRectangle( IX,IY,5,true );  { draw center of body }

        with CntrPress do
             begin
             X := 0.0; Y := 0.0; Z := 1.0;
             end;

        for I:=1 to 6 do        { initialize SupLeg to get coefficients of line }
            with SupLeg[I] do Z:=1.0; { Y value of right leg is positive}

        K := 0;
        for I:=1 to 3 do
            begin
            J := (I-1)*6 + 19;  { array index of Points for tips of left legs}
            L := (I-1)*2 + 1;
            if L in Contact
                then begin
                     K := K + 1;
                     with SupLeg[K] do
                        begin
                        -Y := Points[J].Y-H[4].Y; X := Points[J].X-H[4].X;
                        end;
                     end;
            end;
        if K = 3            { check covexity of left sides }
            then begin
                 GetLines( SupLeg[3],SupLeg[1],Line ); { coeff. of a line }
                 D := DotProd( Line,SupLeg[2] );
                 if D > 0.0     { leg3 is inside of the line }
                     then with SupLeg[3] do
                                begin   { remove leg3 in support pattern }
                                K := K-1; SupLeg[2].X := X; SupLeg[2].Y := Y;
                                end;
                 end;

        M := 0;
        for I:=3 downto 1 do
            begin
            J := (I-1)*6 + 22; { array index of tips of right legs }
            L := I*2;
            if L in Contact
                then begin
```

```
                      K := K+1; M := M+1;
                      with SupLeg[K] do
                         begin
                         Y := Points[J].Y-M[4].Y;
                         X := Points[J].X-M[4].X;
                         end;
                      end;
                end;
            if M = 3
                then begin
                     GetLines( SupLeg[K],SupLeg[K-2],Line );
                     D := DotProd( Line,SupLeg[K-1] );
                     if D > 0.0
                         then with SupLeg[K] do
                                  begin
                                  SupLeg[K-1].X := X; SupLeg[K-1].Y := Y;
                                  K := K-1;
                                  end;
                     end;

            IX := trunc( Scale*SupLeg[K].X + Middle.X );
            IY := trunc( Scale*SupLeg[K].Y + Middle.Y );
            Moveto( IX,IY );
            for I := 1 to 6 do
                  if I <= K
                       then begin
                            IX := trunc( Scale*SupLeg[I].X + Middle.X );
                            IY := trunc( Scale*SupLeg[I].Y + Middle.Y );
                            Drawto( IX,IY );
                            end
                       else begin   { put dummy vector into display buffer }
                            IX := 0; IY := 0; Moveto( IX,IY );
                            end;

{ if CntrPress is passed from other routine,              }
{ then sign of X value should be changed                  }
{ before calling StabilityMargin with X and Y interchanged. }
        StabilityMargin( CntrPress,K );
        StpNam;

end;    { SupportPattern }


procedure WorkVolume( var Polygons : ArrayMaxPolysPolygon;
                      var Vertices : ArrayMaxVtcesInteger;
                      var Body : Array52Vector
                     );

const   Middle = Vector( 300.0,800.0,0.0 );
        Scale = 15.0;
        L11 = 2.2; L21 = 0.44; L31 = 3.0; L41 = 3.0;
var
        Xwidth1, Ywidth1 : real;
        I, J, IX, IY, FilNum    : integer;
        Center : Vector;


procedure DashLine( X1,Y1,X2,Y2: real;
                    Horizontal : boolean );
const   N = 5;
```

```
var     I, IX1, IY1, IX2, IY2
                       : integer;
        Int     : real;

begin
        if Horizontal
                then begin
                     IX1 := trunc( Scale * X1 + Middle.X );
                     IY1 := trunc( Scale * Y1 + Middle.Y );
                     end
                else begin
                     IY1 := trunc( Scale * X1 + Middle.Y );
                     IX1 := trunc( Scale * Y1 + Middle.X );
                     end;
        Moveto( IX1,IY1 );

        Int := ( X2 - X1 ) / N;
        for I := 1 to N do
                begin
                X1 := X1 +  Int;
                if Horizontal
                      then begin
                           IX2 := trunc( Scale * X1 + Middle.X );
                           if odd( I ) then Drawto( IX2,IY1 )
                                       else Moveto( IX2,IY1 );
                           end
                      else begin
                           IY2 := trunc( Scale * X1 + Middle.Y );
                           if odd( I ) then Drawto( IX1,IY2 )
                                       else Moveto( IX1,IY2 );
                           end;
                end;
end;    ( DashLine )

procedure DrawKlimit( Center: Vector; Xwidth,Ywidth : real;
                      DrawVolume : boolean );

var     XB, XT, YL, YR
                : real;
        IXB, IXT, IYL, IYR : integer;

begin   ( DrawKlimit )
        with Center do
             begin
             XB := X - Xwidth; YL := Y - Ywidth;
             XT := X + Xwidth; YR := Y + Ywidth;
             end;
        if DrawVolume
             then begin
                  DashLine( XB,YL,XT,YL,true );
                  DashLine( YL,XT,YR,XT,false );
                  DashLine( XT,YR,XB,YR,true );
                  DashLine( YR,XB,YL,XB,false );
                  end
             else begin
                  IXB := trunc( Scale*XB + Middle.X);
                  IYL := trunc( Scale*YL + Middle.Y);
```

```
                    IXT := trunc( Scale*XT + Middle.X);
                    IYR := trunc( Scale*YR + Middle.Y);
                    Moveto( IXB,IYL );
                    Drawto( IXT,IYL );
                    Drawto( IXT,IYR );
                    Drawto( IXB,IYR );
                    Drawto( IXB,IYL );
                    end;
end;    ( DrawKlimit )


begin   ( WorkVolume procedure body )
        FilNum := 6; NamFil( FilNum );

        with Polygons[2] do      ( draw body      )
                begin
                with Body[Vertices[Start+PolyVtces]] do
                        begin
                        IX := trunc( Scale*X + Middle.X);
                        IY := trunc( Scale*Y + Middle.Y);
                        Moveto( IX,IY );
                        end;
                for J :=1 to PolyVtces do
                        with Body[Vertices[Start+J]] do
                             begin
                             IX := trunc( Scale*X + Middle.X);
                             IY := trunc( Scale*Y + Middle.Y);
                             Drawto( IX,IY );
                             end;
                end;

        ( draw kinematic limit )
        Xwidth1 := ( L31+L41 )*0.5; Ywidth1 := ( L11+L21 )*0.5;
        for I := 1 to 6 do
             begin
             with Center do
                begin
                if I in [1,2]
                        then X := Xwidth1-L41+Xhip[I]
                        else X :=-Xwidth1+L41+Xhip[I];
                if I in [1,3,5]
                        then Y := Ywidth1-L21+Yhip
                        else Y := -Ywidth1+L21-Yhip;

                end;
             ( draw original working volume )
             DrawKlimit( Center,Xwidth1,Ywidth1,false);
             end;

        StpNam;

end;    ( WorkVolume )
```

```
{$nomain}
{$own}
{*************************************************************************}
{ File: TEREST.PAS                                                        }
{ This file is for terrain estimation through linera regression.         }
{                                                                         }
{*************************************************************************}

%include Global;
%include External;

procedure EstimatePlane( var SupportPlane : Plane;
                         var FPSE : Array6Vector
                       );

{ estimate support plane using variance                    }

var     N,
        Xbar, Ybar, Zbar,
        Num1, Num2, Num3,
        Den1, Den2,
        YRbar, ZRbar,
        A0, A1, A2, A3, A4,
        A, B, C, D, M
                : real;
        I       : integer;
        YR, ZR
                : Array6;
begin
    { Compute average of FPSE }
    N := 6.0; Xbar := 0.0;   Ybar := 0.0;   Zbar := 0.0;
    for I := 1 to 6 do
        begin
        Xbar := Xbar + FPSE[I].X;
        Ybar := Ybar + FPSE[I].Y;
        Zbar := Zbar + FPSE[I].Z;
        end;
    Xbar := Xbar/N;
    Ybar := Ybar/N;
    Zbar := Zbar/N;

    {  compute A0, A1, A2, and A3 }
    Num1 := 0.0; Den1 := 0.0; Num2 := 0.0;
    for I := 1 to 6 DO
         with FPSE[I] do
              begin
              Num1 := Num1 + ( X - Xbar ) * ( Z - Zbar );
              Den1 := Den1 + ( X - Xbar ) * ( X - Xbar );
              Num2 := Num2 + ( X - Xbar ) * ( Y - Ybar );
              end;

    A1 := Num1/Den1;
    A2 := Num2/Den1;
    A0 := Zbar - A1 * Xbar;
    A3 := Ybar - A2 * Xbar;

    { compute YRbar and ZRbar }
    YRbar := 0.0; ZRbar := 0.0;
    for I := 1 to 6 do
         with FPSE[I] do
```

```
                    begin
                    YR[I] := Y - A2 - A3 * X;
                    YRbar := YRbar + YR[I];
                    ZR[I] := Z - A0 - A1 * X;
                    ZRbar := ZRbar + ZR[I];
                    end;

    YRbar := YRbar/N;
    ZRbar := ZRbar/N;

    Num3 := 0.0; Den2 := 0.0;
    for I := 1 to 6 do
          begin
          Num3 := Num3 + ( YR[I] - YRbar ) * ( ZR[I] - ZRbar );
          Den2 := Den2 + ( YR[I] - YRbar ) * ( YR[I] - YRbar ) ;
          end;

    A4 := Num3/Den2;

    ( get plane coefficients : A*X + B*Y + C*Z + D = 0.0 )
    A := -A1 + A4*A3; B := -A4; C := 1; D := -A0 + A2*A4;
    M := sqrt( A*A + B*B + C*C );
    with SupportPlane do
         begin
         with UnitNormal do ( Normal vector always goes positive Z direction )
                 begin X := A/M; Y := B/M; Z := C/M; end;
         Dist := D/M; ( if plane is above origin, Dist <0.0 )
                      ( if Plane is below origin, Dist >0.0 )
         end;

end;     ( EstimatePlane )


procedure TerrainCoord( var M,
                            T,
                            INVT,
                            G,
                            INVG : Matrix;
                        var Et1 : real;
                        var SPE,
                            SPB : Plane
                      );


var      Beta,             ( rotation angle from XE to XT )
         M,
         SBeta, CBeta, SEta1, CEta1
                        : real;

begin
         ( get position vector of T )
         T[4].X := M[4].X;    T[4].Y := M[4].Y;
         with SPE do
                 with UnitNormal do
                           begin
                           T[4].Z := -( X*M[4].X + Y*M[4].Y + Dist )/ Z;
                           if abs( 1.0 - Z ) < 0.000001
                                   then begin ( if no slope, then terrain coord )
                                              ( is same as earth coord         )
                                        Beta := 0.0; Eta1 := 0.0;
```

```
                                     end
                                else begin { if slope, get slope angle and }
                                     Beta := Atan2( X,-Y); { rotation angles }
                                     Etal := ArcCos( Z );
                                     end;
                         end;
        { get rotation matrix of T }
        SBeta := Sin( Beta );   SEtal := Sin( Etal );
        CBeta := Cos( Beta );   CEtal := Cos( Etal );
        with T[1] do
             begin
             X := CBeta;  Y := SBeta;  Z := 0.0;
             end;
        with T[2] do
             begin
             X := -SBeta * CEtal;  Y := CBeta * CEtal;  Z := SEtal;
             end;
        with T[3] do
             begin
             X := SBeta * SEtal;   Y :=-CBeta * SEtal;  Z := CEtal;
             end;
        MatInverse( T,INVT );    { get inverse matrix of T }
        MatMult( INVT,H,G );    {get transform matrix G from body to terrain };
        MatInverse( G,INVG );    { get inverse matrix of G }
        PlaneTransform( SPE,H,SPB );   { transform support plane }
                                       { in body coord }
end;    { TerrainCoord }
```

```
{$nomain}
{$own}
{**********************************************************************}
{ File: LEGCORD.PAS                                                    }
{ This file is for leg coordination with computation of optimal period }
{ and the change of the position and size of the constrained working   }
{ volumes.                                                             }
{                                                                      }
{**********************************************************************}

%include Global;
%include External;

procedure GetIntersection( var ClipXT,
                               ClipXB,
                               ClipYL,
                               ClipYR,
                               X1,
                               Y1,
                               X2,
                               Y2 : real;
                           var Error : integer
                          );

{ X1,Y1 are the position outside kinematic limit              }
{ X2,Y2 are the present foot position                         }
{ if X2,Y2 are outside kinematic limit, return error=1        }
{ X1,Y1 return the intersection point in body coordinate      }

label   10;

type    OutCode = set of 1..4;

var     Empty1, Empty2 : boolean;
        C, C1, C2      : OutCode;
        X, Y    : real;


procedure Code(     X,Y : real;
                var C : OutCode;
                var Empty : boolean
               );
begin
        C := [];
        if X < ClipXB
                then C := [ 4 ]
                else if X > ClipXT then C := [ 3 ];
        if Y < ClipYR
                then C := C + [ 2 ]
                else if Y > ClipYL then C := C + [ 1 ];
        if C = [] then Empty := true
                  else Empty := false;
end;    { Code }

begin
        Code( X1,Y1,C1,Empty1 );
        Code( X2,Y2,C2,Empty2 );
        if not ( Empty1 or Empty2 )
              then
                   if ( C1 * C2 ) <> []
```

```
                                then begin
                                     Error := 1; goto 10;
                                     end
                                else Error := 0;

              while( C1 <> [] ) or ( C2 <> [] ) do
                   begin
                   if ( C1 * C2 ) <> [] then goto 10;
                   C := C1; if C = [] then C := C2;
                   if 4 in C
                         then begin
                              Y := Y1 + (Y2-Y1)*(ClipXB-X1)/(X2-X1);
                              X := ClipXB;
                              end
                   else if 3 in C
                              then begin
                                   Y := Y1 + (Y2-Y1)*(ClipXT-X1)/(X2-X1);
                                   X := ClipXT;
                                   end
                   else if 1 in C
                              then begin
                                   X := X1 + (X2-X1)*(ClipYL-Y1)/(Y2-Y1);
                                   Y := ClipYL;
                                   end
                   else if 2 in C
                              then begin
                                   X := X1 + (X2-X1)*(ClipYR-Y1)/(Y2-Y1);
                                   Y := ClipYR;
                                   end;
                   if C = C1
                        then begin
                             X1 := X;  Y1 := Y;  Code( X1,Y1,C1,Empty1 );
                             end
                        else begin
                             X2 := X;  Y2 := Y;  Code( X2,Y2,C2,Empty2 );
                             end;
                    end;
10:
end;    { GetIntersection }


procedure OptimalPeriod( var BodyTransRate,
                             BodyRotateRate : Vector;
                         var RefPosB,
                             FPB : Array6Vector;
                         var dx,
                             dy,
                             LPhase : Array6;
                         var Contact : Tactile;
                         var Beta,
                             Period : real
                        );

label   20;

const   Middle = Vector( 300.0,800.0,0.0 );
        Scale = 15.0;
        MaxPeriod = 10000.0;
        MinPeriod = 0.03;
        InnerVel = 0.05;
        OuterVel = 0.2;

var     XFoot, YFoot, ZFoot,
        XFootVel, YFootVel, ZFootVel,
        Xwidth, Ywidth,
        ClipXT, ClipXB, ClipYL, ClipYR,
        Speriod,
        FWDPeriod, BWDPeriod,
        FWDSPeriod, BWDSPeriod,
        KXint, KYint, KXint1, KYint1,
        VEL,
        Y1, X2, Y3, X4
                : real;
        IX1, IY1,
        IX2, IY2,
        FilNum,
        I, J, J1
                : integer;
        RightLeg
                : boolean;


procedure Margin( var XFoot,YFoot,ZFoot,
                      XFootVel,YFootVel,ZFootVel : real );

label   200;

const   LongTime = 1000000.0;
        MinVel = 0.00001;

var     FootVel,
        XMargin, SPTime, ZTime,
        SPhase,
        Xint, Yint
                : real;
        Error   : integer;

begin   { procedure Margin body }

if RightLeg                         { transform right to left }
        then begin
             YFoot := -YFoot;
             YFootVel := -YFootVel;
             end;

FootVel := sqrt( XFootVel*XFootVel + YFootVel*YFootVel + ZFootVel*ZFootVel );
if FootVel < MinVel      { if foot velocity is almost zero, }
        then begin       { then set support period to 10000.0 }
             FWDPeriod := MaxPeriod;
             BWDPeriod := MaxPeriod;
             if FWDPeriod < FWDSPeriod then FWDSPeriod := FWDPeriod;
             if BWDPeriod < BWDSPeriod then BWDSPeriod := BWDPeriod;
             goto 200;
             end;

{ define an imaginary point outside kinematic limit to compute intersection }
Xint := XFoot + XFootVel * LongTime;
Yint := YFoot + YFootVel * LongTime;
```

```
GetIntersection( ClipXT,ClipXB,ClipYL,ClipYR,Xint,Yint,XFoot,YFoot,Error );

if Error = 1 then begin
                  FWDSPeriod := MinPeriod;
                  BWDSPeriod := MinPeriod;
                  goto 200;
                  end;

XMargin := sqrt( (Xint-XFoot)*(Xint-XFoot) + (Yint-YFoot)*(Yint-YFoot) );

( supporting time to reach kinematic limit  volume )
FootVel := sqrt( XFootVel*XFootVel + YFootVel*YFootVel );
if FootVel < MinVel then SPTime := 100000.0 * XMargin    ( support time )
                    else SPTime := XMargin / FootVel;    ( for x and y vel )

if ZFootVel > 0.0          ( support time for z velocity )
        then ZTime := ( Zmax - ZFoot ) / ZFootVel
        else if ZFootVel < 0.0
                  then ZTime := -( ZFoot - Zmin ) / ZFootVel
                  else ZTime := SPTime;
if SPTime > ZTime then SPTime := ZTime;

( support period for support legs )
SPhase := LPhase[I]/Beta;
if 1.0-SPhase < 0.00001
         then FWDPeriod := 100000.0*SPTime
         else FWDPeriod := SPTime/(1.0-SPhase);
if SPhase < 0.00001
          then BWDPeriod := 100000.0*SPTime
          else BWDPeriod := SPTime/SPhase;


( take minimum period of support periods for forward sequencing )
if FWDPeriod < FWDSPeriod
        then begin
             FWDSPeriod := FWDPeriod;
             J := I;  ( STORE LEG NUMBER )
             KXint := Xint;
             if RightLeg then KYint := -Yint
                         else KYint :=  Yint ;
             end;

( take minimum period of support periods for backward sequencing )
if BWDPeriod < BWDSPeriod
        then begin
             BWDSPeriod := BWDPeriod;
             KXint1 := Xint ;
             J1 := I;
             if RightLeg then KYint1 := -Yint
                         else KYint1 :=  Yint ;
             end;

200:
end;    ( Margin )


begin ( procedure OptimalPeriod body )

FWDSPeriod := MaxPeriod; ( initialize support period )
BWDSPeriod := MaxPeriod;
```

```
for I := 1 to 6 do
        if I in Contact
                then begin

                     Xwidth := 0.5 * dx[I]; Ywidth := 0.5 * dy[I];
                     if I in [1,3,5] then RightLeg := false
                                     else RightLeg := true;
                     { define kinematic limit of each leg }
                     ClipXT := RefPosB[I].X + Xwidth;
                     ClipXB := RefPosB[I].X - Xwidth;
                     ClipYL := RefPosB[I].Y + Ywidth;
                     ClipYR := RefPosB[I].Y - Ywidth;
                     if RightLeg then
                              begin
                              ClipYL := -RefPosB[I].Y + Ywidth;
                              ClipYR := -RefPosB[I].Y - Ywidth;
                              end;

                      { compute new period using new foot velocity }
                     XFoot := FPB[I].X;
                     YFoot := FPB[I].Y;
                     ZFoot := FPB[I].Z;
                     XFootVel :=-BodyTransRate.X + BodyRotateRate.Z * YFoot
                               - BodyRotateRate.Y * ZFoot;
                     YFootVel :=-BodyTransRate.Y - BodyRotateRate.Z * XFoot
                               + BodyRotateRate.X * ZFoot;
                     ZFootVel :=-BodyTransRate.Z + BodyRotateRate.Y * XFoot
                               - BodyRotateRate.X * YFoot;

                     Margin( XFoot,YFoot,ZFoot,XFootVel,YFootVel,ZFootVel );

                     end;

{ consider only longitudinal or lateral velocity }
{ to choose between forward and backward gaits }
with BodyTransRate do
        Vel := sqrt( X * X + Y * Y );

if ( Vel <= OuterVel) and ( abs( BodyTransRate.X ) < InnerVel )
        then begin      { pure turn-in-place case }
              if Period > 0.0
                      then SPeriod := FWDSPeriod
                      else SPeriod := -BWDSPeriod;
              end
        else if Vel <= OuterVel
                 then begin      { gait change region }
                       if BodyTransRate.X > 0.0
                               then SPeriod := FWDSPeriod
                               else SPeriod := -BWDSPeriod;
                       end
                 else if Period > 0.0  { outside gait change region }
                                then SPeriod := FWDSPeriod
                                else SPeriod := -BWDSPeriod;
if SPeriod < 0.0
        then begin
              J := J1;  KXint := KXint1;  KYint := KYint1;
              end;

FilNum := 7; NamFil( FilNum ); Stover( FilNum );
```

```
Period := SPeriod / Beta;          { compute period }
if (SPeriod = MaxPeriod)           { if vehicle velocity is almost zero }
   or (SPeriod = -MaxPeriod)       { skip drawing kinematic margin }
   or (SPeriod = MinPeriod )
   or ( SPeriod = -MinPeriod )
        then begin                 { put dummy vectors to display buffer }
             IX1 := 0; IY1 := 0; IX2 := 0; IY2 := 0;
             Moveto( IX1,IY1 ); Moveto( IX2,IY2 ); goto 20;
             end;

{ draw line from footposition to intersection point }
IX1 := trunc( Scale*FPB[J].X + Middle.X );
IY1 := trunc( Scale*FPB[J].Y + Middle.Y );

IX2 := trunc( Scale*KXint + Middle.X );
IY2 := trunc( Scale*KYint + Middle.Y );
Moveto( IX1,IY1 ); Drawto( IX2,IY2 );

20: StpNam;

end;    { OptimalPeriod }


procedure LegCoordination( var M,
                               INVM : Matrix;
                           var SPB,
                               SPE : Plane;
                           var BodyTransRate,
                               BodyRotateRate : Vector;
                           var RefPosB,
                               FPB : Array6Vector;
                           var dx,
                               dy,
                               RPhase,
                               LPhase : Array6;
                           var Contact : Tactile;
                           var Vmax, Vmin,
                               dxmax, dxmin,
                               dymax, dymin,
                               DT,
                               Beta,
                               Phase,
                               Period,
                               SlowdownFactor : real
                          );

const   MinTRTime = 0.5;

var     I       : integer;
        TRTime  : real;

procedure NewCWV;
{ get new dimensions of constrained working volumes            }
{ depending on the foot velocities                             }

var     I       : integer;
        Xwidth, Ywidth,
        XFoot, YFoot, ZFoot,
        XFootVel, YFootVel, ZFootVel
```

```
                    : real;
        Tem         : vector;

begin { NewCWV }

for I := 1 to 6 do
        begin
        if not (I in Contact)
             then begin
                  XFoot := RefPosB[I].X;
                  YFoot := RefPosB[I].Y;
                  ZFoot := RefPosB[I].Z;

                  { get foot velocity components }
                  XFootVel := abs( -BodyTransRate.X
                                   +BodyRotateRate.Z * YFoot
                                   -BodyRotateRate.Y * ZFoot );
                  YFootVel := abs( -BodyTransRate.Y
                                   -BodyRotateRate.Z * XFoot
                                   +BodyRotateRate.X * ZFoot );

                  { get desired x and y dimensions of CWV }
                  if XFootVel < Vmin
                        then dx[I] := dxmin
                        else if XFootVel > Vmax
                                   then dx[I] := dxmax
                                   else dx[I] := ( dxmax-dxmin )
                                                 * ( XFootVel-Vmin )
                                                 / ( Vmax-Vmin )
                                                 + dxmin;
                  if YFootVel < Vmin
                        then dy[I] := dymin
                        else if YFootVel > Vmax
                                   then dy[I] := dymax
                                   else dy[I] := ( dymax-dymin )
                                                 * ( YFootVel-Vmin )
                                                 / ( Vmax-Vmin)
                                                 + dymin;

                  Xwidth := 0.5 * dx[I];  Ywidth := 0.5 * dy[I];

                  { get new center positions of CWV }
                  with RefposB[I] do
                       begin
                       if I in [1,2]
                            then X := Xhip[I] + XL - Xwidth
                            else if I in [5,6]
                                      then X := Xhip[I] - XL + Xwidth
                                      else X := Xhip[I];
                       if I in [1,3,5]
                            then Y := Yhip + YL - Ywidth
                            else Y :=-Yhip - YL + Ywidth;
                       Z := -NominalHeight;
                       end;

                   if I in [3,4] { change the center of constrained }
                                 { working volume of middle legs }
                     then begin
                          Tem := RefPosB[I]; Tem.Z := 0.0;
                          VectTransform( H,Tem,Tem );
```

```
                                        with SPE do
                                             with UnitNormal do
                                                  begin
                                                  Tem.Z := -(X*Tem.X + Y*Tem.Y
                                                          + Dist)/Z;
                                                  VectTransform( INVH,Tem,Tem );
                                                  RefPosB[I].X := Tem.X;
                                                  end;
                                         { check boundary of original working volume }
                                         with RefPosB[I] do
                                             begin
                                             if X - Xwidth < -XL
                                                        then X := Xwidth - XL;
                                             if X + Xwidth >  XL
                                                        then X := XL - Xwidth;
                                             end;
                                         end;
                                end;
                        end;

                for I := 1 to 6 do
                        begin
                        { update reference position     }
                        with SPB do
                            with UnitNormal do
                                 RefPosB[I].Z := -( X*RefPosB[I].X + Y*RefPosB[I].Y
                                                 + Dist ) / Z;
                        end;    { for I loop }

                end;    { NewCWV }


                begin  { LegCoordination }

                NewCWV;         { the new constrained working volume }

                OptimalPeriod( BodyTransRate,BodyRotateRate,RefPosB,FPB,
                               dx,dy,LPhase,Contact,Beta,Period);

                TRTime := ( 1.0 - Beta ) * abs( Period );
                if TRTime < MinTRTime
                        then begin
                             SlowdownFactor := TRTime / MinTRTime;
                             writeln('Slowdown Factor = ', SlowdownFactor:8:3 );
                             TRTime := MinTRTime;
                             Period := Sign( Period ) * TRTime / (1.0 - Beta );
                             end
                        else SlowdownFactor := 1.0;

                { update phase variables }
                Phase := Phase + DT / Period + 1.0;
                Phase := Phase - trunc( Phase );
                for I := 1 to 6 do { compute leg phase variable }
                      begin
                      LPhase[I] := Phase - RPhase[I] + 1.0;
                      LPhase[I] := LPhase[I] - trunc( LPhase[I] );
                      end;

                end;    { LegCoordination }
```

```
{$nomain}
{$own}
{*********************************************************************}
{ File: BODY.PAS                                                      }
{ This file is for the body set-point generation.                     }
{                                                                     }
{*********************************************************************}

%include Global;
%include External;

procedure BodyServo( var H,
                         INVH : Matrix;
                     var SPB,
                         SPE : Plane;
                     var BodyTransRate,
                         BodyRotateRate : Vector;
                     var RefPosB,
                         FPB : Array6Vector;
                     var dx,
                         dy,
                         RPhase,
                         LPhase : Array6;
                     var Contact : Tactile;
                     var Vmax, Vmin,
                         dxmax,dxmin,
                         dymax,dymin
                         DT,
                         Beta,
                         Phase,
                         Period,
                         Etal,
                         NVELX,
                         NVELY,
                         NRVELZ : real
                    );

var     SlowdownFactor
                : real;

procedure BodyRates(
                    var BodyTransRate,
                        BodyRotateRate
                                        : Vector
                    );

{ this is for automatic body attitude and altitude control      }

const
        D45 = 0.7854;           { 45 degree in radian           }
        MinAngle = 0.00001;     { 0 degree in radian            }
        MaxAngle = 0.7;         { 30 degree in radian           }
        MinHeight = 4.0;        { body height for 45 degree slope }

var     DVELX, DVELY, DRVELZ,   { velocity derivative terms      }
        M,
        Height,
        Gamma,                  { angle between normal vectors of }
                                { body and support Planes         }
        Kgain,                  { filter gain for attitude control }
        Eta2,                   { desired body attitude angle      }
        TimeConst               { time const for rotational velocity }
                : real;
        I
                : integer;
        K                       { vector for rotating axis }
                : Vector;
        DBPE, DBPB              { desired body plane in earth and in body }
                : Plane;

begin  { BodyRates }

if Eta1 < MinAngle              { body plane is parallel to support plane }
      then with SPB do
              with UnitNormal do
                   begin
                   Height := NominalHeight;
                   Kgain := 5.0;
                   M := sqrt( X*X + Y*Y );
                   if M = 0.0          { get rotation axis vector K }
                        then begin K.X := 0.0;   K.Y := 0.0; end
                        else begin K.X := -Y/M;  K.Y := X/M; end;

                   Gamma := ArcCos( Z );       { 0.0 < Gamma < PI/2 }
                   BodyRotateRate.X := Kgain * ( K.X * Gamma );
                   BodyRotateRate.Y := Kgain * ( K.Y * Gamma );
                   BodyTransRate.Z := Kgain * ( NominalHeight - Dist );
                   end

       else begin       { body plane is not parallel to support plane }
            Eta2 := MinAngle + ( MaxAngle - MinAngle ) / ( D45 - MinAngle )
                   * ( Eta1 - MinAngle );
            Height := NominalHeight - ( NominalHeight - MinHeight )
                   / ( D45 - MinAngle ) * ( Eta1 - MinAngle );
            Kgain := 5.0;
            with SPE do
                with UnitNormal do
                     M := sqrt( X*X + Y*Y );

            with DBPE do          { desired body plane in earth coord }
                with UnitNormal do
                      begin
                      X := SPE.UnitNormal.X / M * Sin( Eta2 );
                      Y := SPE.UnitNormal.Y / M * Sin( Eta2 );
                      Z := Cos( Eta2 );
                      end;

            PlaneTransform( DBPE,H,DBPB );
            with DBPB do
                 with UnitNormal do
                      begin
                      M := sqrt( X*X + Y*Y );
                      if M = 0.0          { get rotation axis vector K }
                         then begin K.X := 0.0;   K.Y := 0.0; end
                         else begin K.X := -Y/M;  K.Y := X/M; end;

                      Gamma := ArcCos( Z );     { 0.0 < Gamma < PI/2 }
                      BodyRotateRate.X := Kgain * ( K.X * Gamma );
                      BodyRotateRate.Y := Kgain * ( K.Y * Gamma );
```

```
                         BodyTransRate.Z := Kgain * ( Height - SPB.Dist );
                         end;
                 end;

TimeConst := 0.5;

{ filtered rate commands }
DVELX  := (NVELX - BodyTransRate.X) / TimeConst; { longitudinal acceleration }
DVELY  := (NVELY - BodyTransRate.Y) / TimeConst; { lateral acceleration }
DRVELZ := (NRVELZ - BodyRotateRate.Z) / TimeConst; { turning acceleration }

BodyTransRate.X := DVELX*DT + BodyTransRate.X; { logitudinal velocity  }
BodyTransRate.Y := DVELY*DT + BodyTransRate.Y; { lateral velocity      }
BodyRotateRate.Z := DRVELZ*DT + BodyRotateRate.Z; { turning rate          }

end;    { BodyRates }


procedure Deceleration( var SlowdownFactor
                                          : real;
                        var BodyTransRate,
                            BodyRotateRate
                                          : Vector
                       );


begin  { Deceleration }

with BodyTransRate do
        begin
        X := SlowdownFactor * X;
        Y := SlowdownFactor * Y;
        Z := SlowdownFactor * Z;
        end;

with BodyRotateRate do
        begin
        X := SlowdownFactor * X;
        Y := SlowdownFactor * Y;
        Z := SlowdownFactor * Z;
        end;

end;    { Deceleration }


procedure UpdateH(
                  var BodyTransRate,
                      BodyRotateRate
                                    : Vector;
                  var H, INVH
                                    : Matrix
                 );

var
        DistX, DistY, DistZ,    { differential translations      }
        DeltaX, DeltaY, DeltaZ  { differential rotations         }
                : real;
        I
                : integer;
        DeltaH, HDelta          { differential matrix of H       }
                : Matrix;

begin   { UpdateH }

{ get differential change of body }
with BodyTransRate do
        begin
        DistX := X * DT;   DistY := Y * DT;   DistZ := Z * DT;
        end;

with BodyRotateRate do
        begin
        DeltaX := X * DT;  DeltaY := Y * DT;  DeltaZ := Z * DT;
        end;

{ get differential change of H }
with HDelta[1] do
        begin X := 0.0; Y := DeltaZ; Z := -DeltaY; end;
with HDelta[2] do
        begin X := -DeltaZ; Y := 0.0; Z := DeltaX; end;
with HDelta[3] do
        begin X := DeltaY; Y := -DeltaX; Z := 0.0; end;
with HDelta[4] do
        begin X := DistX; Y := DistY; Z := DistZ; end;

MatMult( H,HDelta,DeltaH );     { DeltaH = H * HDelta }

{ get updated H }
for I := 1 to 4 do                        { H = H + DeltaH }
       with H[I] do
              if I <> 4
                   then begin
                        X := X + DeltaH[I].X;
                        Y := Y + DeltaH[I].Y;
                        Z := Z + DeltaH[I].Z;
                        end
                    else begin
                         X := DeltaH[I].X;
                         Y := DeltaH[I].Y;
                         Z := DeltaH[I].Z;
                         end;

Orthogonalization( H );
MatInverse( H,INVH );

end;    { UpdateH }


begin   { BodyServo }

BodyRates( BodyTransRate, BodyRotateRate );

LegCoordination( H,INVH,SPB,SPE,BodyTransRate,BodyRotateRate,
                 RefPosB,FPB,dx,dy,Rphase,LPhase,Contact,Vmax,
                 Vmin,dxmax,dxmin,dymax,dymin,
                 DT,Beta,Phase,Period,SlowdownFactor );

if SlowdownFactor < 1.0
        then Deceleration( SlowdownFactor,
                           BodyTransRate,
                           BodyRotateRate );

writeln( Period:8:3,' ',BodyTransRate.X:8:3,' ',BodyTransRate.Y:8:3,
        ' ',BodyRotateRate.Z:8:3);

UpdateH( BodyTransRate, BodyRotateRate, H, INVH );

end;    { BodyServo }
```

```
($nomain)
($own)
(*****************************************************************)
( File: ASVINT.PAS                                                )
( This file is for initialization of parameters and vehicle motion.  )
(                                                                 )
(*****************************************************************)

%include global;
%include External;

procedure ASVINIT( var H,
                         INVH : Matrix;
                     var dx,
                         dy : Array6;
                     var Contact : Tactile;
                     var dxmin,
                         dymin,
                         DT,
                         FootLift : real;
                     var BetaMode : integer;
                     var RefPosB,
                         FPDE,
                         FPSE,
                         FPS : Array6Vector;
                     var EyeSpace : Matrix4;
                     var NumPolys : integer;
                     var Body : Array52Vector;
                     var Points : ArrayMaxPtsVector;
                     var Polygons : ArrayMaxPolysPolygon;
                     var Planes : ArrayMaxPolysOnePlane;
                     var Vertices : ArrayMaxVtcesInteger
                 );

var     I, J, FilNum       : integer;
        F           : text;
        Filename    : String;
        G           : file of OnePlane;
        FPE         : Vector;
        Xwidth, Ywidth
                    : real;


begin   ( ASVINIT )

( fix dt for non-real time operation )
DT := 0.05;
FootLift := 1.0;
BetaMode := 4;
dxmin := 0.25; dymin := 0.25;

( initialize reference positions )
Xwidth := 0.5 * dxmin;  Ywidth := 0.5 * dymin;

for I := 1 to 6 do
     begin
     with RefPosB[I] do
          begin
          if I in [1,2]
               then X := Xhip[I] + XL - Xwidth
```

```
                  else if I in [5,6]
                            then X := Xhip[I] - XL + Xwidth
                            else X := Xhip[I];
             if I in [1,3,5]
                  then Y := Yhip + YL - Ywidth
                  else Y :=-Yhip - YL + Ywidth;
             Z := -NominalHeight;
             end;
        dx[I] := dxmin;  dy[I] := dymin;
        end;

   { initialize present foot positions }
   FPB := RefPosB;

   { initialize H matrix and inverse matrix INVH }
   with H[1] do
        begin X := 1.0;   Y := 0.0;   Z := 0.0;  end;
   with H[2] do
        begin X := 0.0;   Y := 1.0;   Z := 0.0;  end;
   with H[3] do
        begin X := 0.0;   Y := 0.0;   Z := 1.0;  end;
   with H[4] do
        begin X := 10.0;   Y := 10.0;   Z := NominalHeight;  end;
   MatInverse( H,INVH );

   Filename := 'EYESPACE  ';
   reset( F,Filename,'.dat;1' );   { open file for EyeSpace }
   for I := 1 to 4 do
            begin   { read in  vehicle data file }
            for J := 1 to 4 do
                read( F,EyeSpace[I,J] );
            readln( F );
            end;
   Close( F );

   Filename := 'PLANES    ';
   reset( G,Filename,'.DAT;1' );   { open data file for coefficients of planes }
   for I := 1 to MaxPolys do       { read in Planes }
            begin
            Planes[I] := G^;  get( G );
            end;
   Close( G );

   { read in  vehicle data file }
   Filename := 'Vehicle   ';
   ReadObject( Filename,NumPolys,Points,Polygons,Vertices );
   for I := 1 to 16 do     { store points of body for analysis}
        Body[I] := Points[I];

   WorkVolume( Polygons,Vertices,Body );

   FilNum := 2; NamFil( FilNum );  { Display file number for vehicle }

   Vehicle( FPE,FPE,false,false,Contact,H,dx,dy,RefPosB,FPB,
             EyeSpace,NumPolys,Body,Points,Polygons,Planes,Vertices);

   { initialize desired foot positions and estimating points }
   for I := 1 to 6 do
            begin
            J := (I-1)*3;
        FPDE[I] := Points[19+J];
        end;
    FPSE := FPDE;

    end;    { ASVINIT }


    procedure Halt( var NVELX,
                        NVELY,
                        NRVELZ : real;
                    var BodyTransRate,
                        BodyRotateRate : Vector;
                    var FRB : Array6Vector
                  );


    var     I       : integer;

    begin   { Halt }

    { initialize velocity vectors }
    NVELX   := 0.0;   NVELY   := 0.0;   NRVELZ   := 0.0;
    with BodyTransRate do
         begin
         X := 0.0; Y := 0.0; Z := 0.0;
         end;

    with BodyRotaterate do
         begin
         X := 0.0; Y := 0.0; Z := 0.0;
         end;

    for I := 1 to 6 do
        with FRB[I] do
             begin
             X := 0.0;  Y := 0.0;  Z := 0.0;
             end;

    writeln;
    writeln( CHR(27),'#6---ASV84 STOPPED---');

    end;  { Halt }


    procedure Initialize( var H : Matrix;
                          var dx,
                              dy,
                              RPhase,
                              LPhase : Array6;
                          var RefPosB,
                              FPDE,
                              FPB : Array6Vector;
                          var FootLift, Phase,
                              Period,
                              Beta : real;
                          var Contact : Tactile;
                          var EstimateFlag : boolean;
                          var LiftoffFlag,
                              PlaceFlag : Array6Boolean;
                          var EyeSpace : Matrix4;
                        var NumPolys : integer;
                        var Body : Array52Vector;
                        var Points : ArrayMaxPtsVector;
                        var Polygons : ArrayMaxPolysPolygon;
                        var Planes : ArrayMaxPolysOnePlane;
                        var Vertices : ArrayMaxVtcesInteger
                    );

type    LegStates = ( Liftoff, TransferForward, Placement );

var     Foot             : integer;
        TPhase, TRTime
                         : real;
        LegState
                : LegStates;
        FPE     : Vector;

begin   ( Initialize procedure body )
writeln;
writeln( CHR(27), '#6 INITIALIZING ...');
Phase := 0.0;  ( initialize kinematic cycle Phase )
Period := 1000.0;
TRTime := ( 1.0 - Beta ) * abs( Period );

( compute desired initial foot heights )
for Foot := 1 to 6 do
        begin
        ( initialize legphase and desired foot positions )
        LPhase[Foot] := Phase - RPhase[Foot] + 1.0;
        LPhase[Foot] := LPhase[Foot] - trunc( LPhase[Foot] );

        if LPhase[ Foot ] < Beta

            then begin  ( support phase )
                 Contact := Contact + [ Foot ]; ( initialize contact state )

                 FPB[ Foot ].Z := -NominalHeight;

                 ( turn off all flags of transfer leg )
                 LiftoffFlag[Foot] := false;
                 PlaceFlag[Foot] := false;
                 end

            else begin  ( transfer phase )
                 Contact := Contact - [ Foot ];

                 TPhase := ( LPhase[ Foot ] - Beta ) / ( 1.0 - Beta );

                 if TPhase < LiftPhase  ( specify leg state )
                            then LegState := Liftoff
                            else if TPhase > PlacePhase
                                           then LegState := Placement
                                           else LegState := TransferForward;

                 case LegState of
                      Liftoff: begin
                               FPDB[Foot].Z := FootLift;
                               LiftoffFlag[Foot] := true;
                               PlaceFlag[Foot] := false;
                               FPB[Foot].Z :=-NominalHeight + FootLift
```

```
                                                  * TPhase / LiftPhase;
                                        end;

                            TransferForward: begin
                                        LiftoffFlag[Foot] := false;
                                        PlaceFlag[Foot] := false;
                                        FPB[Foot].Z := -NominalHeight + FootLift;
                                        end;

                               Placement: begin
                                        FPDE[Foot].Z := 0.0;
                                        LiftOffFlag[Foot] := false;
                                        PlaceFlag[Foot] := true;
                                        FPB[Foot].Z := -NominalHeight + FootLift
                                                  *( 1.0-TPhase )/(1.0-PlacePhase);
                                        end;
                              end; ( case end )
                           end;
                     end;

        Vehicle( FPE,FPE,false,false,Contact,H,dx,dy,RefPosB,FPB,
                 EyeSpace,NumPolys,Body,Points,Polygons,Planes,Vertices );

        ( initialize terrain estimation flag )
        EstimateFlag := true;

        writeln;
        writeln( CHR(27), '#6 INITIALIZATION COMPLETE');
        writeln;

        end;  ( Initialize )
        ;HPGRAP.MAC
        ;WHA-JOON  LEE                10-27-82
        ;CALL STATEMENT:              CALL INITZ
        ;                             CALL NAMFIL(FILNUM)
        ;                             CALL STPNAM
        ;                             CALL STOVER(FILNUM)
        ;                             CALL MOVETO(IX,IY)
        ;                             CALL DRAWTO(IX,IY)
        ;
        .LIST TTM
        GTS = 167760      ;GRAPHICS TRANSLATOR STATUS
        GTB = 167762      ;GRAPHICS TRANSLATOR BUFFER
        ;
        ;MACRO TO OUTPUT DATA TO THE HP I/O DEVICE
        ;
        .MACRO OUTHP T,?GWAIT
GWAIT:  TSTB    @#GTS              ;CHECK STATUS OF HP
        BPL     GWAIT              ;IF BIT 7 IS 1, HP IS READY
        BIC     #002,@#GTS         ;CLEAR BIT 1 (DEVCMD)
        MOV     T,@#GTB            ;MOVE DATA TO OUTPUT BUFFER
        BIS     #002,@#GTS         ;SET BIT 1 FOR DATA AVAILABLE
        .ENDM
        ;
        ;TO INITIALIZE HP GRAPHICS TRANSLATOR
        ;
INITZ:: MOV     #INIT,R1
        MOV     #8.,R2
LOOP:   OUTHP   (R1)+
        SOB     R2,LOOP
        RTS     PC
INIT:   .WORD 146000,144000,101000,102000
        .WORD 100000,112000,110000,115000
        ;
        ;TO NAME DATA FILE
        ;
NAMFIL::        TST     (R5)+
        MOV     @(R5)+,R1          ;FILE NUMBER
        ADD     #107000,R1         ;NAME FILE
        OUTHP   R1
        RTS     PC
        ;
        ;TO STOP NAMING FILE
        ;
STPNAM::
        OUTHP   #112000            ;STOP NAMING FILE
        RTS     PC
        ;
        ;TO MOVE POINTER TO THE BEGINNING OF FILE
        ;
STOVER::        TST     (R5)+
        MOV     @(R5)+,R1          ;GET FILE NUMBER
        ADD     #133000,R1         ;FIND FILE
        OUTHP   R1                 ;OUTPUT "FIND FILE COMMAND"
        RTS     PC
        ;
        ;TO MOVE THE CURRENT BEAM POSITION TO A POINT
        ;TO START DRAWING.
        ;
MOVETO::        TST     (R5)+
        OUTHP   #143000            ;PEN CONTROL
        OUTHP   #141000         ;PEN UP
        OUTHP   @(R5)+          ;GET IX
        OUTHP   @(R5)+          ;GET IY
        RTS     PC
        ;
        ;TO GENERATE A VISIBLE VECTOR FROM THE CURRENT
        ;BEAM POSITION TO A POINT
        ;
DRAWTO::        TST     (R5)+
        OUTHP   #141001         ;PEN DOWN
        OUTHP   @(R5)+          ;GET IX
        OUTHP   @(R5)+          ;GET IY
        RTS     PC
        ;
        ;TO ERASE  ALL DATA OF A FILE
        ;ERASE SHOULD BE FOLLOWED BY STOVER TO PUT NEW DATA
        ;ON THE ERASED FILE
        ;
ERASE::         TST     (R5)+
        MOV     @(R5)+,R1       ;FILE NUMBER
        ADD     #103000,R1      ;ERASE FILE
        OUTHP   R1              ;OUTPUT "EF COMMAND"
        RTS     PC
        ;
        ;FIND ABSOLUTE LOCATION IN DISPLAY BUFFER
        ;
FINDL::         TST     (R5)+
        MOV     @(R5)+,R1       ;GET LOCATION
        OUTHP   #132000         ;OUTPUT "FL COMMAND"
        OUTHP   R1              ;OUTPUT LOCATION
        RTS     PC
        ;
        ;DISPLAY TEXT OF STRING1
        ;CALLING--TEXT(VAR IX,IY,SIZE,NCHAR: INTEGER; VAR TX: STRING1 )
        ;
TEXT1:: TST     (R5)+
        OUTHP   #143000         ;USING PEN CONTROL
        OUTHP   #141000         ;PEN UP
        OUTHP   @(R5)+          ;IX
        OUTHP   @(R5)+          ;IY
        OUTHP   #141001         ;PEN DOWN
        MOV     @(R5)+,R1       ;SIZE
        ADD     #145000,R1      ;CHAR SIZE AND ROTATION
        OUTHP   R1              ;TO HP
        OUTHP   #140000         ;COMMAND FOR "TEXT"
        MOV     @(R5)+,R4       ;NCHAR
        MOV     (R5)+,R3        ;ADDRESS OF FIRST ELEMENT OF TEXT
LOOP1:  MOVB    (R3)+,R2        ;GET A CHAR
        BIC     #177600,R2      ;CLEAR PARITY BIT
        OUTHP   R2              ;TO HP
        SOB     R4,LOOP1
        OUTHP   #146000         ;END OF TEXT
        RTS     PC
        ;
        ;DISPLAY TEXT OF STRING2
        ;CALLING--TEXT(VAR IX,IY,SIZE,NCHAR: INTEGER; VAR TX: STRING2 )
        ;
TEXT2:: TST     (R5)+
        OUTHP   #143000         ;USING PEN CONTROL
        OUTHP   #141000         ;PEN UP
        OUTHP   @(R5)+          ;IX
        OUTHP   @(R5)+          ;IY
        OUTHP   #141001         ;PEN DOWN
        MOV     @(R5)+,R1       ;SIZE
        ADD     #145000,R1      ;CHAR SIZE AND ROTATION
        OUTHP   R1              ;TO HP
        OUTHP   #140000         ;COMMAND FOR "TEXT"
        MOV     @(R5)+,R4       ;NCHAR
        MOV     (R5)+,R3        ;ADDRESS OF FIRST ELEMENT OF TEXT
LOOP2:  MOVB    (R3)+,R2        ;GET A CHAR
        BIC     #177600,R2      ;CLEAR PARITY BIT
        OUTHP   R2              ;TO HP
        SOB     R4,LOOP2
        OUTHP   #146000         ;END OF TEXT
        RTS     PC
        .END
```

```
{***********************************************************************}
{ File: TERDIS.PAS                                                      }
{ This file is for perspective display of the prismatic terrain.        }
{                                                                       }
{***********************************************************************}

Program TERDIS;

%include Global;
%include Mainvar;
%include DISP;
%include Joystick;
%include Libr;

const   OneDegree = 0.017453;    { 1 degree in rad }
        Ywidth = 60.0;           { Y-width of terrain in feet }
        Space = 1.0;          { distance between two lines in terrain display }
        CntrInt = Vector( 20.0,30.0,0.0 );
        R = 200.0;       { distance between EP and Center of interest }

var     FilNum, I, J    : integer;
        Filename        : String;
        EyePt           : Vector;
        Jscale, DT,  Last, Present,
        Xangle, Yangle, HR
                        : real;
        Velocity        : array3;
        F               : text;
        G               : file of OnePlane;

procedure Initz: NonPascal;
procedure NamFil( var FilNum : integer ): NonPascal;
procedure StpNam: NonPascal;
procedure Stover( var FilNum : integer ): NonPascal;
procedure Moveto( var IX,IY : integer ): NonPascal;
procedure Dramto( var IX,IY : integer ): NonPascal;

procedure  ReadTerrain;
{ global modified : NumPolys, Points }

var     I, J,
        V1, V2, V3, V4, NumPts,
        TmpNumpts
                : integer;
        Filename
                : string;
        F, G
                : text;
begin
        Filename := 'Terrain    ';
        reset( F,Filename,'DB1:[357,1]TERRAIN.DAT' );   { open input file }
        Filename := 'tr4.dat    ';         { output filename for display data }
        rewrite( G,Filename,'.DAT' );   { create output file }
        readln( F,NumPts );    TmpNumpts := NumPts;
        NumPts := 2*NumPts;     NumPolys := TmpNumPts - 1;

        { generate a new file for polygon description of terrain }
        writeln( G,NumPts:3,' ',NumPolys:3);
        for I:=1 to TmpNumPts do        { description of points }
                with Points[I] do
```

```
                        begin
                        readln( F,J,X,Z ); Y := 0.0;
                        writeln( G,J:3,' ',X:8:3,' ',Y:8:3,' ',Z:8:3);
                        end;

        for I:=1 to TmpNumPts do        { description of points }
                with Points[I] do
                        begin
                        J := I + TmpNumPts;  Y := Ywidth;
                        writeln( G,J:3,' ',X:8:3,' ',Y:8:3,' ',Z:8:3);
                        end;

        J := 4;         { number of edges }
        for I:=1 to NumPolys do         { vertex descriptions }
                begin
                V1 := I; V2 := I+TmpNumPts; V3 := V2+1; V4 := I+1;
                writeln( G,J:3,' ',V1:3,' ',V2:3,' ',V3:3,' ',V4:3 );
                end;
        Close( F );     { close terrain information file }
        Close( G );     { close terrain data file }
end;    { ReadTerrain }


procedure GetPlanes;
{ global referenced: NumPolys, Polygons, Points, Vertices               }
{         modified: Planes                                     }

var     A1, B1, C1, M
                : real;
        J1, J2, J3, I
                : integer;
        Pt1, Pt2,UnitNormal
                : Vector;

begin
        for I:=1 to NumPolys do
                begin
                with Polygons[I] do
                        begin
                        J1 := Vertices[ Start+1 ];
                        J2 := Vertices[ Start+2 ];
                        J3 := Vertices[ Start+PolyVtces ];
                        end;

                with Planes[I] do       { plane coefficients }
                        begin
                        VectSub( Points[J3],Points[J1],Pt1 );
                        VectSub( Points[J2],Points[J1],Pt2 );
                        CrossProd( Pt1,Pt2,UnitNormal );
                        Dist :=-DotProd( UnitNormal,Points[J1] );

                        { boundaries of a plane }
                        Xmin := Points[J1].X; Xmax := Points[J3].X;
                        if I = NumPolys then Xmax := 1000.0;
                        end;
                end;    { for loop }
end;    { GetPlanes }


procedure Spacing( Mt: Matrix4 );
```

```
{ global referenced: NumPolys, Points, Ywidth, Space    }

var     TmpPt, NewPt
                : Vector;
        XD, ZD, HD, Slope, CosX,
        Xdis, Zdis
                : real;
        I, J, IX, IY
                : integer;
        Flag1, Flag2
                : boolean;

begin
        for I:=1 to NumPolys do
            begin
            XD := Points[I+1].X - Points[I].X;
            ZD := Points[I+1].Z - Points[I].Z;
            HD := sqrt( XD*XD + ZD*ZD );
            if XD <> 0.0
                then begin
                     Slope := ZD/XD; CosX := XD/HD;
                     Xdis := CosX*Space; Zdis := Slope*Xdis;
                     end
                else begin
                     Xdis := 0.0;
                     if ZD > 0.0 then Zdis := Space
                                 else Zdis :=-Space;
                     end;

            with Points[I] do
                begin
                TmpPt.X := X+Xdis; TmpPt.Y := 0.0; TmpPt.Z := Z+Zdis;
                end;

            repeat
                Vect4Transform( Mt,TmpPt,NewPt );
                MakeDisplayable( NewPt );
                with NewPt do
                     begin
                     IX := trunc(X); IY := trunc(Y); Moveto(IX,IY);
                     end;
                TmpPt.Y := Ywidth;
                Vect4Transform( Mt,TmpPt,NewPt );
                MakeDisplayable( NewPt );
                with NewPt do
                     begin
                     IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
                     end;
                with TmpPt do
                     begin
                     X := X+Xdis; Y := 0.0; Z := Z+Zdis;
                     end;

                Flag1 := false; Flag2 := false;
                if ZD > 0.0
                        then begin
                              if TmpPt.Z > Points[I+1].Z
                                 then Flag1 := true
                                 else Flag1 := false;
                               end
```

```
                    else begin
                         if (TmpPt.Z < Points[I+1].Z)
                            or (TmpPt.X > Points[I+1].X)
                            then Flag2 := true
                            else Flag2 := false;
                         end

     until  Flag1 or Flag2;

end;     { for loop }

{ draw base of terrain }
Vect4Transform( Mt,Points[1],NewPt );
MakeDisplayable( NewPt );
with NewPt do
        begin
        IX := trunc(X); IY := trunc(Y); Moveto(IX,IY);
        end;
with TmpPt do
        begin
        X := 0.0; Y := 0.0; Z := -10.0;
        end;
Vect4Transform( Mt,TmpPt,NewPt );
MakeDisplayable( NewPt );
with NewPt do
        begin
        IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
        end;
with TmpPt do X := Points[NumPolys+1].X;
Vect4Transform( Mt,TmpPt,NewPt );
MakeDisplayable( NewPt );
with NewPt do
        begin
        IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
        end;
Vect4Transform( Mt,Points[NumPolys+1],NewPt );
MakeDisplayable( NewPt );
with NewPt do
        begin
        IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
        end;
for I := 1 to 2 do
begin
with TmpPt do
        begin
        X := 0.0; Y := 0.0; Z := -10.0;
        if I = 2 then X := Points[NumPolys+1].X;
        end;
Vect4Transform( Mt,TmpPt,NewPt );
MakeDisplayable( NewPt );
with NewPt do
        begin
        IX := trunc(X); IY := trunc(Y); Moveto(IX,IY);
        end;
with TmpPt do Y := Ywidth;
Vect4Transform( Mt,TmpPt,NewPt );
MakeDisplayable( NewPt );
with NewPt do
        begin
        IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
```

```
                end;

        with TmpPt do
                begin
                Z := 0.0; if I = 2 then Z := Points[NumPolys+1].Z;
                end;
        Vect4Transform( Mt,TmpPt,NewPt );
        MakeDisplayable( NewPt );
        with NewPt do
                begin
                IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
                end;
        end; { for i:= 1 to 2 do }

end;    { Spacing }

begin   { TERDIS }
        Initz; FilNum := 1; NamFil(FilNum);
        ReadTerrain;{ read in terrain information and make terrain data file }
        Filename := 'tr4.dat    ';
        ReadObject( Filename ); { read in terrain data file }
        GetPlanes;

        { get eyespace by using joystick }
        Xangle := -90.0; Yangle := 10.0;        { initialize eye point }
        Jscale := 5.0; Present := Time;
        repeat
             Joystick( Velocity, Jscale );
             Last:= Present;
             Present := Time;
             DT := 3600.0 * ( Present - Last );
             Xangle := Xangle + DT * Velocity[0];
             Yangle := Yangle + DT * Velocity[1];
             with EyePt do
                begin
                HR := R * Cos( OneDegree * Yangle );
                X := HR * Cos( OneDegree * Xangle ) - CntrInt.X;
                Y := HR * Sin( OneDegree * Xangle ) - CntrInt.Y;
                Z := R * Sin( OneDegree * Yangle )  - CntrInt.Z;
                end;
             GetEyeSpace( EyePt,CntrInt );
             Stover( FilNum );
             DisplayObject( EyeSpace,false );   { display terrain }
             Spacing( EyeSpace );               { display spacings }
             StpNam;
        until ( Velocity[2] < -2.5 );

        Filename := 'EYESPACE  ';               { EYESPACE.DAT;1 }
        rewrite( F,Filename,'.DAT;1' );         { create file for EyeSpace }
        for I := 1 to 4 do                      { store data in file }
                begin
                for J := 1 to 4 do
                     write( F,EyeSpace[I,J] );
                writeln( F );
                end;
        Close( F );

        Filename := 'PLANES    ';               { PLANES.DAT;1 }
        rewrite( G,Filename,'.DAT;1' );         { create file for coefficients}
        for I := 1 to MaxPolys do               { of planes of terrain        }
                begin
                G^ := Planes[I]; put( G );
                end;
        Close( G );

end.    { TERDIS }
```

```
(************************************************************************)
( File: TERPRF.PAS                                                      }
( This file is for terrain profile generation.                          }
{                                                                       }
(************************************************************************)

program TERPRF;

%include Global;
%include Mainvar;
%include DISP;
%include Joystick;
%include Libr;


type    String1 = packed array[1..11] of char;
        String2 = packed array[1..2] of char;

var     IX, IY
                : integer;

procedure Initz; NonPascal;
procedure NamFil( var FilNum : integer ); NonPascal;
procedure StpNam; NonPascal;
procedure Stover( var FilNum : integer ); NonPascal;
procedure Moveto( var IX,IY : integer ); NonPascal;
procedure Drawto( var IX,IY : integer ); NonPascal;
procedure Erase( var FilNum : integer ); NonPascal;
procedure FindL( var Locate : integer ); NonPascal;
procedure Text1( var IX,IY,CharSize,Nchar: integer; var TX: String1 );
        NonPascal;
procedure Text2( var IX,IY,CharSize,Nchar: integer; var TX: String2 );
        NonPascal;


procedure Cursor( var IX,IY: integer );
( draw arrow cursor on screen )

const   L = 20;

var     KX,KY,FileNum : integer;

begin
        FileNum := 4; NamFil( FileNum );
        KX := IX - L; KY := IY - L; Moveto( KX,KY );
        Drawto( IX,IY );
        KX := IX + L; Drawto( KX,KY );
        StpNam;
end;    ( Cursor )

procedure MoveCursor( var IX,IY : integer );
( move cursor on screen )

var     Present, Last, Scale, DT : real;
        FileNum : integer;
        Velocity : array3;
begin
        FileNum := 4;
        Scale   := 200.0;       ( scale for joystick )
        Present := Time;
        repeat                    { move cursor without drawing vectors }
            Joystick( Velocity,Scale );
            Last := Present;
            Present := Time;
            DT := 3600.0 * ( Present - Last );
            IX := trunc( DT*Velocity[0] ) + IX;
            IY := trunc( DT*Velocity[1] ) + IY;
            Stover( FileNum );
            Cursor( IX,IY )
        until ( Velocity[2] < -100.0 ); { until z axis is twisted clockwise }
        while Velocity[2] < -70.0 do    { wait until joystick returns    }
             Joystick( Velocity,Scale );{ to netural position  }
end;    { MoveCursor }


procedure DrawMenu;

var     FileNum, IX, IY,
        CS, NC  : integer;
        TX : String1;
begin
        FileNum := 1; NamFil( FileNum );
        IX := 0; IY := 800;
        Moveto( IX,IY );
        IY := 1000; Drawto( IX,IY );
        IX := 400;  Drawto( IX,IY );
        IY := 800;  Drawto( IX,IY );
        IX := 0;    Drawto( IX,IY );
        IX := 200; IY := 1000; Moveto( IX,IY );
        IY := 800; Drawto( IX,IY );
        NC := 11; CS := 0;       { set character size }
        IX := 34; IY := 910;  TX := '   Draw    ';
        Text1( IX,IY,CS,NC,TX );
        IX := 34; IY := 860;  TX := '  Terrain  ';
        Text1( IX,IY,CS,NC,TX );
        IX := 234; IY := 910; TX := 'Perspective';
        Text1( IX,IY,CS,NC,TX );
        IX := 234; IY := 860; Tx := '   View    ';
        Text1( IX,IY,CS,NC,TX );
        StpNam;
end;    { DrawMenu }


procedure DrawXZ;
{ draw X and Z axis to draw cross section of terrain }

const   Xmin = 50;      Xmax = 900;
        Ymin = 350;     Ymax = 600;
var     I, IX, IY, FileNum,
        J, JX, JY, CS, NC
                : integer;
        TX      : String2;

begin
        FileNum := 2; NamFil( FileNum );
        NC := 2;          { number of character }
        CS := 1;          { character size }
        for I := 1 to 2 do       { draw XZ axis }
            begin
            IX := Xmin; IY := Ymin; Moveto( IX,IY );
```

```
              if I = 1 then begin
                        IX := Xmax; JX := IX+7; JY := IY-12; TX :='X ';
                        end
                    else begin
                        IY := Ymax; JX := IX-8; JY := IY+7; TX :='Z ';
                        end;
              Drawto( IX,IY );
              Text2( JX,JY,CS,NC,TX );
              end;
           CS := 0;
           JX := Xmin; JY := Ymin-25; TX :='0 '; Text2( JX,JY,CS,NC,TX );
           for I := 1 to 8 do      { draw scale every 10 feet on X axis }
              begin                { put the numbers on them }
              IX := Xmin + I*100; IY := Ymin + 5; Moveto( IX,IY );
              IY := Ymin - 5;  Drawto( IX,IY );
              JX := IX - 10; JY := IY - 20;
              case I of
                   1: TX:='10'; 2: TX:='20'; 3: TX:='30'; 4: TX:='40';
                   5: TX:='50'; 6: TX:='60'; 7: TX:='70'; 8: TX:='80';
              end;         { case end }
              Text2( JX,JY,CS,NC,TX );
              if I = 8 then begin
                        JX := JX + 35; TX:='ft'; Text2( JX,JY,CS,NC,TX );
                        end;
              end;

           for I := 1 to 4 do      { draw scale every 5 feet on Z axis }
              begin                { and put numbers on them           }
              IY := Ymin + I*50; IX := Xmin + 5 ; Moveto( IX,IY );
              IX := Xmin - 5;  Drawto( IX,IY );
              JX := IX - 30; JY := IY - 6;
              case I of
                1: TX:=' 5'; 2: TX:='10'; 3: TX:='15'; 4: TX:='20';
              end;
              Text2( JX,JY,CS,NC,TX );
              if I = 4 then begin
                        JY := JY+25; TX:='ft'; Text2( JX,JY,CS,NC,TX );
                        end;
              for J := 1 to 32 do
                   begin
                   IX := IX + 20;  Moveto( IX,IY );
                   IX := IX + 5;   Drawto( IX,IY );
                   end;
              end;
           StpNam;
end;       { DrawXZ }

procedure DrawTerrain;
{ draw lines for terrain interactively using joystick }
{ and generate an output file for terrain information }

label   10;

const   Scale = 0.1; Xmin = 50; Ymin = 350;

type    String = packed array[1..7] of char;

var     IX, IY, I, J, FileNum, L : integer;
        Filename : string;
        F : text;
        TmpPts : array[1..30] of Vector;

begin
10:     FileNum := 3; NamFil( FileNum );
        L := 500;        { initialize write pointer for terrain display }
        IX := Xmin; IY := Ymin; I := 1;
        with TmpPts[I] do
             begin
             X := 0.0; Z := 0.0;
             end;
        FindL(L);        Moveto( IX,IY ); L := L + 1;
        MoveCursor( IX,IY );
        while (IX>400) or (IY<800) do
             begin
             I := I + 1;
             with TmpPts[I] do
                  begin
                  X := Scale * ( IX - Xmin );
                  Z := Scale * ( IY - Ymin );
                  end;
             FindL(L);    Drawto( IX,IY ); L := L + 1;
             MoveCursor( IX,IY );
             end;
        StpNam;

        if IX < 200
             then begin
                  Initz;
                  DrawMenu; Cursor( IX,IY );DrawXZ;      { erase previous data }
                  goto 10;        { draw again }
                  end;
        Filename := 'terrain';
        rewrite( F,Filename,'.DAT');     { create output file }
        writeln( F,I:3);          { write number of points }
        for J := 1 to I do
             with TmpPts[J] do
                  writeln( F,J:3,' ',X:8:3,' ',Z:8:3 ); { write X and Z values }
        close( F );
end;    { DrawTerrain }


begin   { TRPRF body }
        Initz; DrawMenu;
        IX := 500; IY := 500;
        Cursor( IX,IY );
        while ( IY<800 ) or ( IX>200 ) do
                  MoveCursor( IX,IY );
        DrawXZ;
        DrawTerrain;

end.    { TRPRF }
```

```
(****************************************************************************)
( File: DISP.PAS                                                             )
( This file is for graphics routines                                         )
(                                                                            )
(****************************************************************************)

procedure ReadObject( var Filename: String );

var     I, J , NumVtces, NumPts : integer;
        F : text;

begin
reset(F, Filename, '.CAT');     ( open input file )
readln(F,NumPts, NumPolys); ( read in points )
for I := 1 to NumPts do
        with Points[I] do
                readln(F, J, X, Y, Z);
NumVtces := 0;                  ( initialize size of vertex array )
for I := 1 to NumPolys do       ( read in polygon descriptions )
        with Polygons[I] do
                begin
                start := NumVtces;      ( start point in vertex array )
                read(F, PolyVtces );    ( number of vertices )
                for J := 1 to PolyVtces do      ( read vertex pointers )
                        read(F, Vertices[ NumVtces + J ]);
                readln(F);              ( go to next line of input )
                NumVtces := NumVtces + PolyVtces;
                end;
Close(F);       ( close the file )
end;    ( ReadObject )


procedure GetEyeSpace( EyePt, CntrInt: Vector );

var     Mtx : Matrix4;
        C1,C2 : Vector;
        Hypotenuse,CosA,SinA : real;

begin
with EyePt do
        TransMat( -X,-Y,-Z,EyeSpace );
Vect4Transform( EyeSpace,CntrInt,C1 );    ( translate center of interest )

with C1 do Hypotenuse := sqrt( X*X + Y*Y );
if Hypotenuse <> 0.0 then
        begin
        CosA := C1.Y / Hypotenuse; SinA := C1.X / Hypotenuse;
        RotateMat( 'Z',CosA,SinA,Mtx );         ( rotate about z axis )
        Mat4Mult( Mtx,EyeSpace,EyeSpace );
        end;
Vect4Transform( EyeSpace,CntrInt,C2 );  ( rotate ctr. of interest )

with C2 do Hypotenuse := sqrt( Y*Y + Z*Z );
if Hypotenuse <> 0.0 then
        begin
        CosA := C2.Y / Hypotenuse; SinA := -C2.Z / Hypotenuse;
        RotateMat( 'X',CosA,SinA,Mtx );         ( rotate about x axis )
        Mat4Mult( Mtx,EyeSpace,EyeSpace );
        end;
Ident( Mtx );
```

```
                { switch between Y and Z axes }
Mtx[2,2] := 0.0; Mtx[2,3] := -1.0;
Mtx[3,2] := 1.0; Mtx[3,3] := 0.0;
Mat4Mult( Mtx,EyeSpace,EyeSpace );
end;    { GetEyeSpace }


procedure MakeDisplayable( var Pt : Vector );

begin           { perspective projection }
Pt.X := Scale * Pt.X + Middle.X;
Pt.Y := Scale * Pt.Y + Middle.Y;
end;


procedure DisplayObject( var Mt : Matrix4);

var     TmpPt   : Vector;
        I,J,K,
        IX,IY   : integer;

begin
for I := 1 to NumPolys do
        with Polygons[I] do
                begin
                Vect4Transform( Mt, Points[Vertices[Start+PolyVtces]],TmpPt);
                MakeDisplayable( TmpPt );
                with TmpPt do
                        begin
                        IX := trunc(X); IY := trunc(Y); Moveto(IX,IY);
                        end;
                for J := 1 to PolyVtces do
                        begin
                        Vect4Transform( Mt,Points[Vertices[Start+J]],TmpPt );
                        MakeDisplayable( TmpPt );
                        with TmpPt do
                                begin
                                IX := trunc(X); IY := trunc(Y); Drawto(IX,IY);
                                end;
                        end;
                end;
end;    { DisplayObject }


procedure DisplayFile( var Mt : Matrix4);

var     TmpPt   : Vector;
        I,J,K,
        IX,IY   : integer;
        F       : text;

begin
rewrite(F,'ASVHP.dat');
for I := 1 to NumPolys do
        with Polygons[I] do
                begin
                Vect4Transform( Mt, Points[Vertices[Start+PolyVtces]],TmpPt);
                MakeDisplayable( TmpPt );
                with TmpPt do
                        writeln(F,'3',' ',X,' ',Y);
                for J := 1 to PolyVtces do
                        begin
                        Vect4Transform( Mt,Points[Vertices[Start+J]],TmpPt );
                        MakeDisplayable( TmpPt );
                        with TmpPt do
                            writeln(F,'2',' ',X,' ',Y);
                        end;
                end;
close( F );
end;    { DisplayFile }
```

EN

FILM

12-84

DTIC