AFRL-AFOSR-UK-TR-2014-0045

Motion Planning and Task Assignment for Unmanned Aerial Vehicles Cooperating with Unattended Ground Sensors

Tal Shima
Pantelis Isaiah
Yoav Gottlieb

TECHNION R & D FOUNDATION LTD
SENATE BUILDING, TECHNION CITY
HAIFA ISRAEL

EOARD Grant 12-2116

Report Date: October 2014

Final Report from 10 August 2012 to 9 August 2014

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515, APO AE 09421-4515

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE ***(DD-MM-YYYY)*** 24 October 2014	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From – To)* 10 August 2012 – 09 August 2014

4. TITLE AND SUBTITLE **Motion Planning and Task Assignment for Unmanned Aerial Vehicles Cooperating with Unattended Ground Sensors**	**5a. CONTRACT NUMBER** **FA8655-12-1-2116**
	5b. GRANT NUMBER **Grant 12-2116**
	5c. PROGRAM ELEMENT NUMBER **61102F**
6. AUTHOR(S) Tal Shima Pantelis Isaiah Yoav Gottlieb	**5d. PROJECT NUMBER**
	5d. TASK NUMBER
	5e. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) TECHNION R & D FOUNDATION LTD SENATE BUILDING, TECHNION CITY HAIFA ISRAEL	**8. PERFORMING ORGANIZATION REPORT NUMBER** # 2017129
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD Unit 4515 APO AE 09421-4515	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/AFOSR/IOE (EOARD)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) **AFRL-AFOSR-UK-TR-2014-0045**

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this research we focused on intertwined low-level motion planning and high-level task assignment for a cooperating team of heterogeneous unmanned assets. As a case study we are envisioning the scenario where a team of unmanned aerial vehicles (UAVs) has to query a set of unattended ground sensors (UGSs), positioned along a road network. In such a scenario the UAV team cooperates in searching and prosecuting ground target intruders, identified by the ground sensors. Of particular interest to us was the cooperation and coordination between the unmanned aerial vehicles in devising their assignments and trajectories. The research was conducted in two phases. In the first phase we have concentrated on the motion planning and task assignment aspects of the problem where a single UAV has to decide on the query ordering of the UGSs, given constraints on its motion. This problem was formalized as the Dubins travelling salesman problem (TSP). In the second phase of the research we have paid special attention to the coupled motion planning and task assignment problem for groups of UAVs and UGSs. The specific problem considered includes allocating the group of UAVs to a given set of UGSs while accounting for the vehicles kinematic constraints and avoiding collision with obstacles scattered in the vehicles' environment.

15. SUBJECT TERMS

EOARD, motion planning, cooperative control, UAV, UGS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Kevin Bollino
a. REPORT UNCLAS	**b. ABSTRACT** UNCLAS	**c. THIS PAGE** UNCLAS	SAR	47	**19b. TELEPHONE NUMBER** *(Include area code)* +44 (0)1895 616163

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39-18

Faculty of Aerospace Engineering
Technion – Israel Institute of Technology

Motion Planning and Task Assignment for Unmanned Aerial Vehicles Cooperating with Unattended Ground Sensors

AFOSR Contract No. FA8655-12-1-2116

Final Report
(Covering the period: August 2012 - October 2014)

Principal Investigator: Prof. Tal Shima

Investigators: Dr. Pantelis Isaiah and Yoav Gottlieb

Abstract

In this research we focused on intertwined low-level motion planning and high-level task assignment for a cooperating team of heterogeneous unmanned assets. As a case study we are envisioning the scenario where a team of unmanned aerial vehicles (UAVs) has to query a set of unattended ground sensors (UGSs), positioned along a road network. In such a scenario the UAV team cooperates in searching and prosecuting ground target intruders, identified by the ground sensors. Of particular interest to us was the cooperation and coordination between the unmanned aerial vehicles in devising their assignments and trajectories.

The research was conducted in two phases. In the first phase we have concentrated on the motion planning and task assignment aspects of the problem where a single UAV has to decide on the query ordering of the UGSs, given constraints on its motion. This problem was formalized as the Dubins travelling salesman problem (TSP). The contributions of the study in the first phase can be summarized as follows:

- Two new algorithms for the Dubins TSP were developed that are applicable to both ordered and unordered sets of targets. Instead of being decoupled, the combinatorial and motion planning aspects of the Dubins TSP were treated in an integrated manner and no assumptions were made on the magnitude of the intercity distances. The two algorithms complement each other in terms of their range of applicability with regard to the size of the problem.
- The first algorithm — dubbed "k-step look-ahead algorithm" — stems from the formulation of the Dubins TSP as a minimum-time control problem and is suitable for obtaining short tours when the number of cities is relatively small.
- The second algorithm is an adaptation of the classic 2-Opt algorithm for the TSP and can be applied to large instances of the Dubins TSP.
- Analytical bounds were derived for the lengths of the tours and for the complexity of the problem. Monte Carlo simulations were used to investigate and compare between the performance of the proposed algorithms and existing algorithms from the literature.

In the second phase of the research we have paid special attention to the coupled motion planning and task assignment problem for groups of UAVs and UGSs. The specific problem considered includes allocating the group of UAVs to a given set of UGSs while accounting for the vehicles kinematic constraints and avoiding collision with obstacles scattered in the vehicles' environment. The contributions of the study in the second phase can be summarized as follows:

- To account for the effect of a diminishing return with time for information collected at each UGS, each UGS is assigned a specific time dependent benefit. The benefit represents the UGS importance and priority, possibly assigned by a human operator. The time varying priority issue is addressed by incorporating the vehicles' feasible path length, which represents the vehicles' response time, and the targets' properties as an inherent part of the problem formulation.
- The integrated problem of task assignment and motion planning was posed in the form of a decision tree and two solution algorithms were developed. An exhaustive search algorithm which improves over run time and provides the best solution encoded in the decision tree, and a greedy algorithm that provides a quick feasible solution (also used as an upper bound).

- Although the greedy algorithm provides a sub-optimal solution, it is useful in large scale real time scenarios, where computational running time is important. The exhaustive algorithm can provide an immediate solution that improves over run time for large scale scenarios, or it can be used in off-line scenarios.
- Using simulations the performance of the algorithms was compared and evaluated, and the influence of the time varying targets' priority on the task allocation process was demonstrated and investigated.

The results of the research have appeared in 2 conference publications, 1 submitted journal publications, and 1 journal paper that is currently in preparation stages.

The published conference papers are:

- Pantelis, I. and Shima, T., "A Task and Motion Planning Algorithm for the Dubins Travelling Salesperson Problem", Proceedings of the 19th IFAC World Congress, Cape Town, South Africa, August 2014.
- Gottlieb, Y. and Shima, T., "Task Assignment and Motion Planning in the Presence of Obstacles and Prioritized Targets", Proceedings of the 54th Israel Annual Aerospace Conference, February 2014.

The following paper has been submitted to Automatica. It is currently in its second round of review:

- Pantelis, I. and Shima, T., "Motion Planning Algorithms for the Dubins Travelling Salesperson Problem", 2014.

The following paper is in preparation to be submitted to the journal Sensors as an invited feature paper:

- Gottlieb, Y. and Shima, T., "UAVs Task and Motion Planning in the Presence of Obstacles and Prioritized Targets", 2014.

Contents

List of Figures

Introduction

This research is concerned with intertwined low-level motion planning and high-level task assignment for unmanned aerial vehicles (UAVs). As a case study, the scenario where a team of UAVs has to query a set of unattended ground sensors (UGSs) is envisioned. The problem can be explained using the following example: A network of Unattended Ground Sensors (UGS) and a team of unmanned vehicles are used to prevent intruders' access to a restricted zone (base defense) [45]. The UGS network is deployed at critical road junctions, when a sensor is triggered by an intruder, the location is sent as a target to be visited by the team of unmanned vehicles. In case there are multiple intrusions at different times, the group of vehicles must be allocated according to the vehicles' response time (time to target) and the target's priority, which can be based on the order of the UGS triggering time or on the location of the sensor. The target (sensor) priority is time dependent since as time passes, the intruder may advance to a different location and the target relevance decreases.

1.1 Single Vehicle - Multiple Targets

In the first phase of our research we have concentrated on the motion planning and task assignment aspects of the problem where a single UAV has to decide on the query ordering of the multiple UGSs, given constraints on its motion. This problem was formalized as the Dubins travelling salesman problem (TSP).

The Dubins Travelling Salesperson Problem (DTSP) and its variants [8, 12, 18, 21, 22, 24, 25] are useful abstractions for the study of problems related to motion planning and task assignment for uninhabited vehicles. As in the case of the classic Euclidean Travelling Salesperson Problem (ETSP) in $\mathbb{R}^2$ [19], the sought after solution to the DTSP is a tour of minimum length that passes through every city (target), however, in the case of the DTSP, the tour is required to be a planar C^1 regular curve whose curvature is bounded above by a given constant. The additional requirements on the regularity and the curvature of the tour have a fundamental implication on the very nature of the problem. Specifically, the ETSP belongs to the realm of combinatorial optimisation, whereas the DTSP does not. A precise formulation of the DTSP is given below, however the crux of the matter is that, in the case of the DTSP, even if the order of the targets is given and fixed, the length of the tour depends on the heading of the Dubins vehicle when it passes through each target (in other words, the slope of the tour at each target). Therefore, the solution space for the DTSP has the cardinality of the continuum.[1] A possible remedy is to directly discretise the problem; that is, for each target, to consider h candidate headings that partition the interval $[0, 2\pi)$ uniformly into h subintervals. The resulting discrete DTSP (DDTSP) can be cast as an integer linear program (ILP) that can be solved using existing, optimised solvers and is

[1]Although the solution space consists of closed, planar *curves*, to each n-tuple $(\theta_1, \ldots, \theta_n) \in \mathbb{S}^n$ of headings, where n is the number of cities, corresponds a finite number of tours of the same minimum length.

amenable to analytic tools from the theory of ILP. The discretisation approach has been employed in [7] to prove tight upper bounds for the optimal solution (to the DTSP) that depend only on the number of targets and the norm of the partition of $[0, 2\pi)$. The bounds are tight in the sense that they can be achieved by instances of the DTSP chosen by an adversary. On the other hand, the DDTSP is equivalent to a generalised asymmetric travelling salesperson problem (GATSP) [13] with n clusters and $1 + h(n-1)$ cities[2] and it is, therefore, of interest to keep the value of h small. However, the minimum-time function for the Dubins dynamics is discontinuous (in other words, an infinitesimal change of the value of a heading can cause the overall length of the tour to jump to a higher value) and, hence, a large value of h is necessary if an optimal solution to the DTSP is to be approximated. One way to address this issue is to observe that the DTSP lends itself naturally to the application of a receding horizon principle. Such an approach is taken in [17]. The k-step look-ahead (k-step LAA) algorithm in Section 2.3 also relies on a receding horizon principle; the essential difference between the k-step LAA and the algorithm in [17] is that, in [17], the order of the targets is considered given and fixed, whereas in the present paper it is one of the unknowns of the problem. With regard to the DDTSP, it is shown in Sections 2.3 and 2.5 that the k-step LAA can reduce the search for a feasible DTSP tour to a combinatorial problem that is smaller in size than the DDTSP[3] (depending on the choices of h and k), while achieving a substantial improvement (by 50%, in certain cases) over algorithms that rely on a solution to the ETSP to find a feasible tour for the DTSP [22].

To address the problem of solving instances of the DTSP that are too large for the k-step LAA to be practical, a local improvement algorithm is presented in Section 2.4. The idea behind the algorithm is borrowed from the literature on the ETSP where it is known as the 2-Opt algorithm [15]. Hence, we call the algorithm of Section 2.4 the "2-Opt k-step look-ahead algorithm" (2-Opt k-step LAA) and establish an upper bound on the length of DTSP tours it generates, under a non-restrictive assumption on the *initial* tour. When the order of the targets is fixed and the k-step LAA is only applied to find an admissible DTSP tour, we use the name k-step ETSP-LAA to distinguish such cases from those where the order of the targets is also unknown.

Once an assignment of a heading to each target has been made, several existing algorithms—e.g., approximation algorithms for the Asymmetric Travelling Salesperson Problem (ATSP)—can be directly applied to find a feasible tour for the DTSP. For the purpose of illustration and comparison, the k-step LAA is combined with Dijkstra's algorithm to generate the simulations in Section 2.5. Given the sheer size of the combinatorial problem to be solved and that Dijkstra's algorithm is a greedy algorithm that returns an optimal solution (a shortest path on a graph), this combination limits the applicability of the overall algorithm to cases with a small number of cities (these statements are quantified in later sections) and leaves open for further investigation the fusion of the k-step LAA with other algorithms. Nevertheless, there are application areas where a small number of targets does not represent an unrealistic scenario.

1.2 Multiple Vehicles - Multiple Targets

In the second phase of the research we have paid special attention to the coupled motion planning and task assignment problem for groups of UAVs and UGSs. The specific problem considered includes allocating the group of UAVs to a given set of UGSs while accounting for the vehicles

[2]The city of origin has its heading fixed to the initial heading of the vehicle. To every other city corresponds a cluster with h nodes—as many as the candidate headings.

[3]At the cost of computing the heading at each city, rather than assigning it arbitrarily.

kinematic constraints and avoiding collision with obstacles scattered in the vehicles' environment.

The task assignment problem is usually coupled with the motion planning problem as the assignments allocation process depends on the path length, and the path length depends on the vehicle's assignments. This coupling issue is addressed in the unmanned vehicles cooperative multiple task assignment problem (CMTAP)[26]. The CMTAP includes a scenario in which multiple unmanned vehicles perform multiple tasks on stationary targets. Different approaches based on customized combinatorial optimization methods were employed to solve this problem, including the mixed integer linear programming (MILP) [27, 28], the capacitated transhipment network solver [29, 30], genetic algorithms [31, 32, 26] and tree search methods [33, 34]. The presented works account for the vehicles' constraints but they simplify the problem by assuming that the environment is obstacle free. Most of the studies which take into account obstacles, address only the motion planning subproblem between an initial and final configuration. They include methods such as the RRTs method [35], probabilistic roadmaps [36] and kinodynamic method [37].

One of the main properties of the problem stated above is the assumption that the targets have the same characteristics and differ only in their position. In many scenarios, each target in the targets set has unique attributes which include different importance and priority. The targets' priority may also vary in time depending on the specific scenario. Cases in which targets are assigned with a priority value were studied in [42, 43, 44]. The problems presented in these studies include a task allocation process that maximizes the service provided to each target based on the target's parameters and the vehicles' capabilities (fuel, payload). The targets priority is addressed by using an objective function which includes a constant parameter describing the priority value. In these problems the vehicles' constraints are not taken into account and the environment is assumed to be free of obstacles, which may lead to infeasible vehicles' trajectories.

Single UAV scenarios

In the first phase of our research we have investigated the problem where a single UAV, modeled as a Dubins vehicle, has to decide on the query ordering of the multiple UGSs. This problem is formalized next as the Dubins travelling salesman problem (TSP).

2.1 Problem Formulation

Consider the control-affine system

$$\gamma'(t) = X(\gamma(t)) + u(t)Y(\gamma(t)) \tag{Σ}$$

on $M = \mathbb{R}^2 \times \mathbb{S}^1$, where $X, Y \in \Gamma^\omega TM$ are the real analytic vector fields with coordinate representations[1]

$$X(x, y, \theta) = (\cos\theta, \sin\theta, 0) \text{ and } Y(x, y, \theta) = (0, 0, 1)\,.$$

The admissible controls are the locally integrable maps $u : \mathbb{R} \supset I \ni t \mapsto u(t) \in [-1/\rho, 1/\rho] \subset \mathbb{R}$, where ρ is a fixed, positive, real number, referred to as the **minimum turn radius**. Given a control u, the corresponding locally absolutely continuous trajectory of (Σ) is denoted by γ. The projection on $\mathbb{R}^2$ of a trajectory γ will be called the **path** that corresponds to γ. An **optimal path** is the projection of an optimal trajectory. A trajectory $\gamma : \mathbb{R} \supset [a, b] \to M$ of (Σ) is said to be **closed** if $\gamma(a) = \gamma(b)$ and the path that corresponds to a closed trajectory is called a **tour**. If I is a subinterval of $\mathbb{R}$, the space of locally absolutely continuous curves in M defined on I is denoted by $W^{1,1}_{\text{loc}}(I; M)$.

The control system (Σ) can be viewed as the kinematic model of a point that moves with constant, unit speed, along a planar curve whose curvature is bounded above by $1/\rho$. We shall abide by the common convention of referring to (Σ) as the "Dubins vehicle" and to the following minimum-time control problem as the Dubins Travelling Salesperson Problem (DTSP).

DTSP: *Let n be a positive integer. Given a point $p \in M$ and n submanifolds (targets) of the form $N_i = \{(x_i, y_i)\} \times \mathbb{S}^1 \subset M$, where $(x_i, y_i) \in \mathbb{R}^2$ and $i \in \{1, \ldots, n\}$, minimise the time $T > 0$ over the set $\Gamma^{\text{cl}}_\Sigma(p, N_1, \ldots, N_n)$ of closed trajectories $\gamma \in W^{1,1}_{\text{loc}}([0, T]; M)$ of (Σ) that satisfy $\gamma(0) = \gamma(T) = p$ and $\text{Im}\gamma \cap N_i \neq \emptyset$, for every $i \in \{1, \ldots, n\}$.*

In view of the interpretation of (Σ) as a kinematic model for planar motion, the targets N_i, above, correspond to fixed points in $\mathbb{R}^2$. Hence, to lessen the notational burden in the remainder of the paper, we refer to targets simply as points (x_i, y_i) in the plane and retain the precise notation $\{(x_i, y_i)\} \times \mathbb{S}^1$ whenever mathematical consistency is warranted. In the same vein, the DTSP can

[1]In the chart on TM induced by the chart $(U, \phi) = (\mathbb{R}^2 \times \mathbb{S}^1 \setminus \{(-1, 0)\}, (x, y, v, w) \mapsto (x, y, \theta = \text{atan}(w/v)))$ on M. To cover the entire state space M, a second chart can be chosen in an obvious manner.

be stated less formally as follows. Given an initial condition p and n points in the plane, find the shortest path for a Dubins vehicle that starts from p, passes through every given point, and returns to p. In our formulation of the DTSP, not only the initial position of the Dubins vehicle is assumed given, but also the initial orientation. It is, of course, possible to also consider the initial orientation of the Dubins vehicle as an independent variable of the optimisation problem. In [22], for example, the initial orientation is chosen freely. This minor discrepancy leads to minor only modifications for any given algorithm to be applicable to either one formulation of the DTSP. To avoid trivial complications with the statements and the notation that follow, we always assume that the targets are distinct from each other and the initial condition p does not lie in any target. Lastly, a tour corresponding to a trajectory $\gamma \in \Gamma_{\Sigma}^{\mathrm{cl}}(p, N_1, \ldots, N_n)$ will be called an **admissible tour**.

2.2 Dubins paths

An essential component of the algorithms presented in later sections is the computation of minimum-time paths for the Dubins vehicle. We, therefore, recall the classification by Dubins [11] of such paths.

Let C_ϕ denote a circular arc of ϕ radians and of radius ρ in the plane, and S_d a straight-line segment of length d in the plane. A C^1 concatenation of such arcs and straight-line segments is denoted by juxtaposition of the corresponding symbols. For example, $C_\alpha S_d C_\beta$ denotes a C^1 curve that consists of an arc of α radians, followed by a straight-line segment of length d, followed by an arc of β radians. Consider, now, the Dubins Problem (DP) which is the following minimum-time problem.

DP: *Given two points $p, q \in M$, minimise the time $T > 0$ over the set of trajectories $\gamma \in W^{1,1}_{\mathrm{loc}}([0,T]; M)$ of (Σ) such that $\gamma(0) = p$ and $\gamma(T) = q$.*
The following theorem is proven in [5, 11, 23].

Theorem 1. *A solution to the Dubins Problem exists and an optimal path has to be either of the form $C_\alpha C_\beta C_\delta$ or of the form $C_\alpha S_d C_\beta$, where $0 \leq \alpha, \delta < 2\pi$, $\pi < \beta < 2\pi$, and $d \geq 0$.*

Moreover, it is shown in [23] that, if $C_\alpha C_\beta C_\delta$ is an optimal path, then $\min\{\alpha, \delta\} < \beta - \pi$ and $\max\{\alpha, \delta\} < \beta$. A path that corresponds to a solution to the DP is called a **Dubins path**. Starting from Theorem 1, it can be shown that a solution to the DTSP exists [3, Theorem 3.5.1] [10, Proposition 2.3, Theorem 3.1]. On the other hand, since solutions to the DP are not unique in the sense that, in general, the minimum may be attained by more than one trajectory, solutions to the DTSP are not unique, a fortiori (an example is given in [14]).

The formulation of the DTSP as a minimum-time control problem hints at possible algorithms for the computation of approximate solutions. The description of such an algorithm is the content of the next section.

2.3 The k-step look-ahead algorithm

The motivation behind the k-step LAA is to replace a uniform (in $[0, 2\pi)$) or random assignment of headings to each target with a judicious choice of headings. This is accomplished by employing a receding horizon principle that allows the motion planning to be integrated with the combinatorial aspect of the DTSP. We turn, now, to a concrete description of these ideas. For future reference, let us call the problem of finding a minimum-time trajectory of (Σ) through k targets the "k-step look-ahead Dubins problem" (k-step LADP). A precise formulation is as follows.

k-step LADP: *Let k be a positive integer. Given a point $p \in M$ and an ordered k-tuple $(N_1, \ldots, N_k)$ of distinct submanifolds (targets) of the form $N_i = \{(x_i, y_i)\} \times \mathbb{S}^1 \subset M$, where $(x_i, y_i) \in \mathbb{R}^2$ and $i \in \{1, \ldots, k\}$, minimise the time $T > 0$ over the set of trajectories $\gamma \in W^{1,1}_{\text{loc}}([0, T]; M)$ of (Σ) such that $\gamma(0) = p$, $\gamma(T) \in N_k$, and $\text{Im}\gamma \cap N_i \neq \emptyset$, for every $i \in \{1, \ldots, k-1\}$.*

In words, we seek a minimum-time path that starts from p, passes through every target in the given order, and ends at the last target. A few comments about the k-step LADP are in order. Let $\tilde{\gamma}$ denote a solution to the k-step LADP. Let, also, t_i denote the first time instant when $\tilde{\gamma}(t_i) \in N_i$, where $i = 1, \ldots, k$ and $t_k = T$. First, it should be noted that the restriction $\tilde{\gamma}|_{[t_{k-1}, t_k]}$, that is, the part of the optimal trajectory between the last two targets can be computed much more efficiently if, instead of Theorem 1 that classifies the minimum-time paths between two given *states*, we use the following result of [4] that classifies the Dubins paths between a given initial *state* and a given final *position*. It is an immediate consequence of the transversality condition at N_k [20].

Lemma 1. *A path that corresponds to a solution to the 1-step LADP is either of the form $C_\alpha C_\beta$ or of the form $C_\alpha S_d$, where $0 \leq \alpha < 2\pi$, $\pi < \beta < 2\pi$, and $d \geq 0$.*

According to the terminology used thus far, Lemma 1 classifies the solutions to the 1-step LADP (whose corresponding paths are also known as **relaxed Dubins paths**) and waives the need to check all possible headings at the final target when an optimal solution to the k-step LADP is computed numerically.

Second, given an instance of the DTSP with n targets, a solution to the k-step LADP with $k = n + 1$ and the final condition $\gamma(T) = p$ substituted for the condition $\gamma(T) \in N_k$ is a globally optimal solution to the DTSP. Clearly, globally optimal trajectories are only relevant when one seeks to comprehend the structure of the solutions to the DTSP by considering instances with a few targets only.

Third, the philosophy behind the k-step LADP—and the advantage of the resulting k-step look-ahead algorithm—is that, even if k is taken to be small (e.g., 1 or 2) relative to n (the number of targets in a given instance of the DTSP), satisfactory admissible tours can still be obtained by iteratively solving a, perhaps large, number of computationally tractable problems.

A formal description of the k-step LAA is somewhat involved and the bookkeeping alone obscures the main idea behind the algorithm. We opt, instead, for a description of the case $k = 2$ by means of a representative example that is simple enough to keep the presentation clear, but also contains the essential ideas so that the extension to the general case ($k > 2$) becomes straightforward.

Suppose an instance of the DTSP with $n = 4$ targets is given and the algorithm to be applied is the 2-step LAA. The first step is to construct a rooted tree whose root R represents the initial condition $p \in M \times \mathbb{S}^1$ of the Dubins vehicle and induces an orientation on the tree away from the root. The children of the root represent, temporarily, the four possible targets. This is shown in Figure 2.1. The representation is "temporary" because, eventually, every node of the tree will represent a target together with a heading assigned to it, not simply a target. Consequently, different nodes may correspond to different headings at the same "physical" target. Next, we assign headings to each one of the children/targets. To this end, each child of R is replicated as

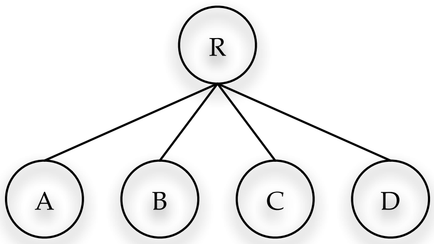


Figure 2.1: The root R of the tree represents the initial condition of the Dubins vehicle and the children of the root represent four targets before headings are assigned to them.

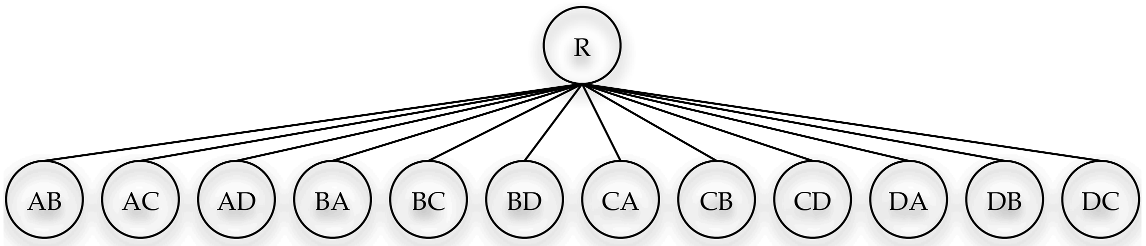


Figure 2.2: Each child of the root in Figure 2.1 is duplicated as many times as the number of possible subsequent targets. The labels on the nodes represent the order in which the targets are visited and each node represents a state of the Dubins vehicle, that is, a location and a heading.

many times as the number of possible subsequent targets. For example, after visiting target A, there are three options: to visit either target B or C or D. Therefore, the root has $\Delta_2^4 = \frac{4!}{2!} = 12$ children.[2] This process leads to the tree in Figure 2.2 and is a consequence of setting $k = 2$ because, now, each node XY, where $X, Y \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$ and $X \neq Y$, can be used to represent the target X with the heading assigned to it by solving the 2-step LADP with initial condition p (represented by the root R), first target X, and second target Y. In the notation of the previous sections, if $\tilde{\gamma}$ is a solution to such a 2-step LADP, $\tau \in]0, \tilde{T}[$ is the first time when $\tilde{\gamma}(\tau) \in X$, and (x, y, θ) are local coordinates, then the child XY is assigned the heading $\theta(\tau)$. The length of the Dubins path that corresponds to $\tilde{\gamma}|_{[0,\tau]}$ is assigned as weight to the edge that connects p to XY. This assignment of headings allows us to view the node XY as the state of the Dubins vehicle that consists of the position of the target X and the heading $\theta(\tau)$. Because the heading at a target X depends on the target Y that is visited next, the grandchildren of the root p are not arbitrary. Rather, a child of a node XY has to be of the form XYZ (i.e., the first two letters remain the same), where $X, Y, Z \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$. Figure 2.3 illustrates this idea which is a design choice: we could allow the children of the node XY to be of the form XWZ with W not necessarily equal to Y, however such a choice would vitiate the anticipative nature of the algorithm. Constraining the children of each node in this way also has the effect of reducing the total number of nodes in the resulting tree.

[2] By definition, $\Delta_k^n = \frac{n!}{(n-k)!}$.

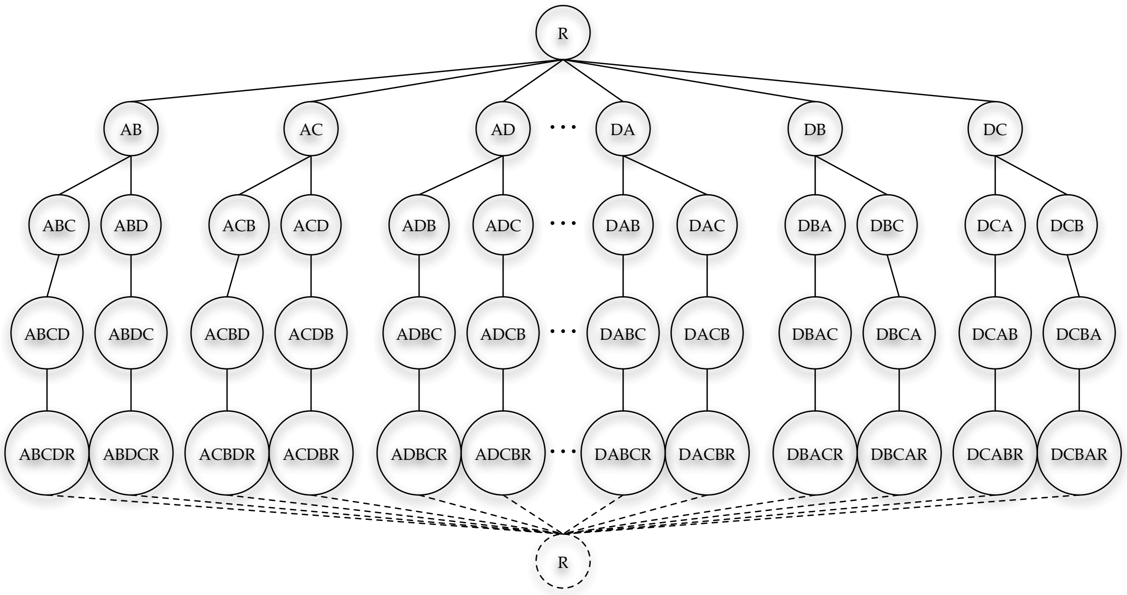


Figure 2.3: The rooted tree used to find a feasible tour for a DTSP with $n = 4$ targets and look-ahead horizon $k = 2$.

Having assigned a heading to every child XY of p, we can proceed in the same manner and compute the weights between the children and the grandchildren of R. The weight of the edge between two nodes XY and XYZ is computed by solving the 2-step LADP with initial condition XY (recall that nodes represent states), first target Y and second target Z. By repeating this procedure, the tree in Figure 2.3 is constructed. To conform with the definition of a rooted tree, the dashed part in Figure 2.3 should not be considered as being formally part of the tree; it is included as a visual aid to the description of the algorithm. Towards the lower end of the tree an off-by-one issue has to be resolved, but this can be done in a straightforward manner. Specifically, the heading of a node (state) with label $XYZWR$, that is, of a node that corresponds to the last target before returning the Dubins vehicle to its initial condition, has to be computed by solving a 2-step LAA with the final condition $\gamma(T) = p$ substituted for the condition $\gamma(T) \in N_k$. Similarly, when $k > 2$, the value of k has to be reduced towards the final stages of the construction of the tree, simply because the look-ahead horizon of the algorithm will exceed the number of targets that are left to be considered (this reduction is, of course, computationally beneficial). Let the tree constructed by the k-step LAA be denoted by $G_{k,n} = (V_{k,n}, E_{k,n})$ (in Figure 2.3, $k = 2$ and $n = 4$). The following proposition gives an upper bound on the length of shortest paths from the root R to the terminal node of $G_{k,n}$.

Proposition 1. *A shortest path in $G_{k,n}$ from the root to the (ficticious) terminal node represents a DTSP tour of length at most*

$$\mathrm{ETSP}(n) + (n+1)\kappa\pi\rho,$$

where n is the number of targets, κ is a constant, and $\mathrm{ETSP}(n)$ *denotes the length of a solution to the corresponding ETSP.*

Proof. Every admissible solution to the DTSP is a concatenation of Dubins paths γ_i, $i = 1, \ldots, n+1$, and the length $\ell(\gamma)$ of a Dubins path γ between two targets satisfies $\ell(\gamma) \le d + \kappa\pi\rho$, where d is the Euclidean distance between the targets and $\kappa \in [2.657, 2.658]$ is a constant [22, Thm 3.4].

Moreover, an instance of the DTSP can also be viewed as an instance of the ETSP and one of the paths from the root R to the terminal node in $G_{k,n}$ corresponds to the optimal order σ_{ETSP} for the ETSP. Therefore, the following bound holds for the length $L^k_\rho(n)$ of a *shortest* path from R to the terminal node

$$\begin{aligned} L^k_\rho(n) &= \textstyle\sum_{i=1}^{n+1} \ell(\gamma_i) \leq \sum_{i=1}^{n+1} (d_i + \kappa\pi\rho) \\ &= \text{ETSP}(n) + (n+1)\kappa\pi\rho, \end{aligned} \tag{2.1}$$

where ETSP(n) denotes the length of the solution σ_{ETSP}. □

Remark. The simulations in Section 2.5 provide strong evidence that the bound (2.1) is not always sharp, especially when the average intercity distance is comparable to the minimum turn radius of the Dubins vehicle. This is to be expected because the derivation of (2.1) does not take into consideration essential features of the k-step LAA such as ordering the targets independently of the solution to the ETSP and the use of a receding horizon principle. On the other hand, it is obvious that

$$\lim_{\rho\to 0} L^k_\rho(n) = \text{ETSP}(n).$$

Once the tree $G_{k,n}$ has been constructed the k-step LAA reduces to a shortest path algorithm. To this end, existing algorithms can be adopted. For the implementation of the 1-step and 2-step LAA used to generate the simulations in Section 2.5, Dijkstra's algorithm was chosen because of its simplicity and because it is guaranteed to return a shortest path in the tree. As far as the resulting time-complexity of the k-step LAA is concerned, a first observation is that

$$|V_{k,n}| \geq 1 + \textstyle\sum_{i=1}^{n} \frac{n!}{(n-i)!}, \tag{2.2}$$

where equality holds for $k = 1$ and the standard convention $0! = 1$ is assumed. In the case $k = 2$ that we use as our running example, we have

$$|V_{2,n}| = 1 + n! + \textstyle\sum_{i=2}^{n} \frac{n!}{(n-i)!} = |V_{1,n}| + n! - n. \tag{2.3}$$

Remark. If we construct a tree $\widetilde{G}_{2,n} = (\widetilde{V}_{2,n}, \widetilde{E}_{2,n})$ that represents all possible DTSP tours in the case where two headings are assigned to each target either randomly or by partitioning $[0, 2\pi)$ into two subintervals, then $|\widetilde{V}_{2,n}|$ increases faster than $|V_{2,n}|$ with respect to n. The following table shows the values of $|\widetilde{V}_{2,n}|$ and $|V_{2,n}|$ as the size n of an instance of the DTSP increases.

n	3	4	5	6	7	8	9	10
$\lvert\widetilde{V}_{2,n}\rvert$	16	65	326	1957	13700	109601	986410	9864101
$\lvert V_{2,n}\rvert$	19	61	201	871	5293	40713	363457	3629611

If the min-priority queue used by Dijkstra's algorithm is implemented as a Fibonacci heap, then the time-complexity of searching in $G_{k,n}$ is [9]

$$O(|V_{k,n}| \log |V_{k,n}| + |E_{k,n}|) = O(|V_{k,n}| \log |V_{k,n}| + |V_{k,n}|), \tag{2.4}$$

where we used the fact that $|E_{k,n}| = |V_{k,n}| - 1$, because $G_{k,n}$ is a tree, and discarded the additive constant. Additionally, the weight of every edge in $E_{k,n}$ is computed by solving a k-step LADP.

Given that Dubins paths can be computed in constant time $O(1)$, if we partition $[0, 2\pi)$ uniformly into h subintervals, then the time-complexity for a numerical approximation of a solution to a k-step LADP is $O(kh^{k-1})$, if we ignore additions and finding minima among lists of numbers. However, a useful observation is that the tree $G_{k,n}$ does not have to be constructed in its entirety and then searched; instead, only the propagating front of Dijkstra's algorithm needs to be stored. This idea, combined with an upper bound for the DTSP tour can reduce the number of computations and amount of memory required. The above discussion is summarised in the following proposition for the case $k = 2$ in which simple explicit formulae like (2.3) can be obtained. Moreover, in practice, the 2-step LAA seems to achieve a good balance between computation time and the size of the resulting DTSP tour.

Proposition 2. *The worst-case time-complexity of the* 2*-step look-ahead algorithm is*

$$O\left(|V_{2,n}| \log |V_{2,n}| + (4h + 1)|V_{2,n}| + n!\right), \tag{2.5}$$

where h is the number of subintervals into which $[0, 2\pi)$ is partitioned when approximating a solution to the 2-step LADP.

Proof. Assuming that the whole tree $G_{2,n}$ has to be searched until a shortest path is found, then, as explained above, the running time for the search procedure is

$$O(|V_{2,n}| \log |V_{2,n}| + |V_{2,n}|).$$

Assigning a weight to every edge of $G_{2,n}$ requires

$$O\left((2h + h + h)|V_{2,n}|\right) = O(4h|V_{2,n}|) \tag{2.6}$$

operations and an additional $O(n!)$ to compute the weights of the dashed edges in Figure 2.3 that correspond to the final Dubins paths that complete the DTSP tours. The term $(2h + h + h)$ in equation (2.6) comes from the fact that we consider h headings in $[0, 2\pi)$ and, for each heading, we have to compute two Dubins paths (one path that connects the initial condition of the Dubins vehicle to the first target and one path that connects the first to the second target; see the formulation of the k-step LADP), sum their lengths, and find the minimum among the h sums of lengths. That is, there are $2h$ computations of Dubins paths, h additions, and finding a minimum among h numbers (which takes $O(h)$ [9]). Combining everything together gives (2.5). $\square$

Similar analyses can be carried out for different values of the look-ahead horizon k, however the calculations need to account for a varying k and, hence, are more involved.

Once headings have been assigned to the targets using the receding horizon principle behind the k-step LAA, another possibility for obtaining an admissible DTSP tour is to view the vertices in V_k as a GATSP with n clusters. Each cluster consisting of the vertices in V_k that represent different headings at the same target. The resulting GATSP can be subsequently transformed to an ATSP and solved by means of an approximation algorithm such as the algorithm in [16, Thm 4.1]. When an approximation algorithm for the ATSP is incorporated as part of the k-step LAA, the solution space is different (larger) than in the case where a shortest path algorithm in $G_{k,n}$ is used, unless the constraint that a child of a node of the form XY be of the form XYZ is imposed, as explained in Section 2.3.

We conclude this section with one final remark on the applicability of the k-step LAA to large instances of the DTSP. One possibility is to create small clusters of targets according to a suitable criterion, apply the k-step LAA to each cluster, and combine the resulting tours into a single tour. This method is facilitated if the targets naturally form spatially separated clusters.

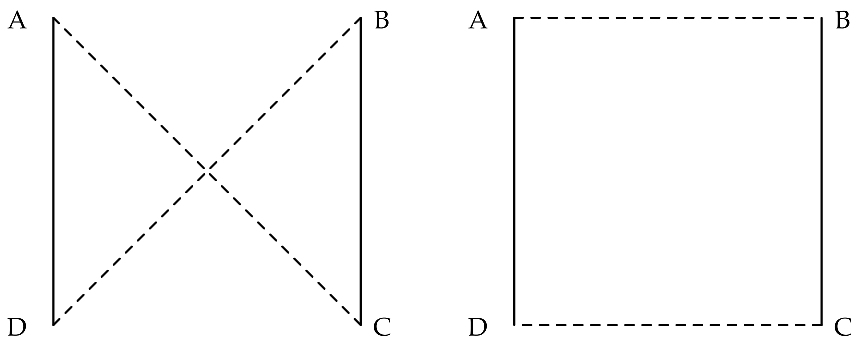


Figure 2.4: A schematic representation of a 2-Opt move.

2.4 A local improvement algorithm

The goal of this section is to describe an algorithm that can be applied to instances of the DTSP that are too large for the k-step LAA to be practical. The motivation behind the algorithm is twofold. First, to leverage one of the many ideas that have been successfully applied to the ETSP and, second, to exploit the fact that, if the order of the targets is fixed, then an admissible DTSP tour can be found quickly. For a thorough description of the ideas that follow, in their original context (the TSP and the ETSP), and related results the reader is referred to [15, Section 3].

A class of algorithms that are particularly effective for the ETSP is that of local improvement algorithms, also known as exchange heuristics. These are algorithms characterised by a set of *exchanges* or *moves* that reorder the targets. A local improvement algorithm attempts to decrease the length of a given tour by performing a sequence of admissible moves. If a move results in a shorter tour, then the new tour is stored as the current optimum, otherwise the new tour is discarded. A sequence of moves is performed until either there there is no improvement or a predetermined maximum number of moves is reached. The most widely known local improvement algorithms are the k-Opt algorithms. The idea behind them is to remove k edges from a given tour, thus breaking the tour into separate paths, and reconnect these paths in different ways. For example, if $ACBDA$ is the initial tour in Figure 2.4, then an example of a 2-Opt move performed by a 2-Opt algorithm is the replacement of edges (A, C) and (B, D) by edges (A, B) and (C, D). In this contrived example, a single move leads to the optimal tour.

Analysing the performance of k-Opt algorithms is a non-trivial matter (e.g., determining after how many moves they halt, their approximation ratio, or if they are guaranteed to improve a given initial tour) and in practice they perform better than what the theoretical analysis suggests [15]. One reason behind the discrepancy between the theoretical bounds and the actual performance of the algorithms is that, in deriving worst-case bounds, it is assumed that an adversary chooses the initial tour or that the adversary also chooses the moves performed by the algorithm. The situation is even more complicated in the case of the DTSP because much less is known analytically about the properties of optimal solutions. For example, an uncrossing move like the one shown in Figure 2.4) cannot increase the length of an ETSP tour and it is known that all crossings can be eliminated from a tour by applying at most n^3 2-Opt moves that remove crossings. Although there are no analogous results for the DTSP, the main idea behind the k-Opt algorithms can be combined with randomisation to obtain an algorithm for the DTSP that performs well in practice. For concreteness and simplicity we consider only the combination of a 2-Opt move with the k-step ETSP-LAA. A 2-Opt move means that, given a DTSP tour, we choose two pairs of consecutive cities (A, B) and (C, D) on the tour and alter the tour so that C is visited after A and D is visited

```
2-Opt k-step look-ahead algorithm
Input:   An instance of the DTSP and a positive integer $I_{\max}$
Output:  An admissible DTSP tour
 1: $I_{\max} \leftarrow$ maximum number of moves
 2: Find an initial order $\sigma_0$ for the targets.
 3: Use $\sigma_0$ to apply the $k$-step ETSP-LAA and obtain
    an initial DTSP tour of length $L_0$ and a vector
    $H_0$ containing the headings at the targets.
 4: for i = 1 to $I_{\max}$
 5:   Apply a randomly generated 2-Opt move to $\sigma_0$
      to find a new order $\sigma$.
 6:   Use $\sigma$ to apply the $k$-step ETSP-LAA and ob-
      tain a DTSP tour of length $L$ and a vector $H$
      containing the headings at the targets.
 7:   if $L < L_0$
 8:     $\sigma_0 \leftarrow \sigma$
 9:     $H_0 \leftarrow H$
10:     $L_0 \leftarrow L$
11:   end if
12: end for
13: return $\sigma_0$, $H_0$, and $L_0$
```

Table 2.1: The 2-Opt k-step look-ahead algorithm

after B. Having no easy (systematic) way to tell a priori which moves will not increase the length of a tour, the 2-Opt moves are generated randomly. The algorithm terminates when a predefined maximum number $I_{\max}$ of moves has been attempted. The corresponding 2-Opt k-step look-ahead algorithm is described in Table 2.1. To obtain an upper bound for the output of the 2-Opt k-step LAA, suppose that an initial order for the targets is found using an approximation algorithm for the ETSP (step 2 in Table 2.1) with approximation ratio α. In principle, such an algorithm could be Christofides's algorithm [6] which has approximation ratio 3/2 or obtained from Arora's polynomial time approximation scheme for the ETSP [2]. Then we have the following.

Proposition 3. *Given an instance of the DTSP with n targets, the 2-Opt k-step look-ahead algorithm returns a DTSP tour of length at most*

$$\alpha \cdot \mathrm{ETSP}(n) + (n+1)\kappa\pi\rho. \tag{2.7}$$

Proof. Given an initial ETSP tour whose length is α times the length of an optimal tour, equation (2.1) implies that the length L_0 of the corresponding DTSP tour (step 3 in Table 2.1) satisfies

$$L_0 \leq \alpha \cdot \mathrm{ETSP}(n) + (n+1)\kappa\pi\rho.$$

In the worst case scenario, no 2-Opt move will result in any improvements of the DTSP tour and σ_0, H_0, and L_0 will be the output of the 2-Opt 2-step look-ahead algorithm. □

Similarly to the case of the k-step LAA, a bound that relies on the order of the targets that corresponds to a solution to the ETSP (possibly approximate solution, in the case of the 2-Opt k-step LAA) is not expected to be sharp, unless the minimum turn radius of the Dubins vehicle is small relative to the average intercity distance. On the other hand, the right-hand side of (2.1) is susceptible to relatively accurate estimations by means of the Held–Karp bound for the ETSP [15, Section 2.3].

2.5 Simulations

2.5.1 The 2-step look-ahead algorithm

The k-step LAA can also be used simply as a receding horizon algorithm on a sequence of n targets $(N_i)_{i=1}^n$ that have been ordered by some other method, e.g., by solving an ETSP. Specifically, suppose that the targets have been reindexed so that N_i is the i-th target. Starting from the initial condition p, a solution $\tilde{\gamma}$ is found to the k-step LADP that corresponds to the first k targets. Then, only the part $\tilde{\gamma}|_{[0,t_1]}$ that connects p to N_1 is kept and the point $\tilde{\gamma}(t_1)$ is considered as a new initial condition from which the k-step LADP for the targets N_2 to N_{k+1} can be solved. This procedure is repeated until an admissible tour is constructed with the horizon k being reduced as necessary when less than k targets are left. Recall that we called this method for finding DTSP tours the "k-step ETSP-LAA".

In this section, the following five algorithms are compared by means of Monte Carlo simulations and the results are shown in Figures 2.5 to 2.7.

1. **1-step ETSP-LAA:** An ETSP is solved to order the targets and the look-ahead horizon is set to $k = 1$.

2. **2-step ETSP-LAA:** An ETSP is solved to order the targets and the look-ahead horizon is set to $k = 2$.

3. **1-step LAA:** The k-step LAA of Section 2.3 for $k = 1$.

4. **2-step LAA:** The k-step LAA of Section 2.3 for $k = 2$.

5. **AA:** (alternating algorithm) This is the algorithm described in [22].

Remark. In [17], algorithms (1) and (2), above, are called "two-point algorithm" and "look-ahead algorithm", respectively. Note, however, that, in [17], these algorithms are applied to sets of targets that are ordered randomly, as opposed to being ordered by first solving the corresponding ETSP.

For each number of targets shown in the x-axes of Figures 2.5, 2.6, and 2.7, 100 instances of the DTSP were randomly generated using the implementation of the Mersenne twister provided by the Boost C++ Libraries [1]. The initial condition was set equal to $p = (0, 0, \pi/2)$ and the targets were contained in $[-2.5, 2.5]^2 \subset \mathbb{R}^2$ with uniform distribution. Next, each algorithm was applied to all randomly generated instances and the length of each DTSP tour was normalised (divided) by the length of the solution to the ETSP for the same set of targets (hence, the normalised length of the solution to the ETSP is always equal to 1). The y-axes correspond to the average of these normalised lengths. The three figures correspond to three different minimum turning radii. As expected, when the minimum turn radius is small (i.e., $\rho = 0.1$ in Figure 2.7) relative to the distance between the targets, the difference between the output of the five algorithms is negligible (note the range of the y-axis). In such a case, execution speed becomes the dominant factor in choosing an algorithm and the 2-step LAA is the most time-consuming among the five algorithms. In all cases, however, the 2-step LAA yields the shortest DTSP tours as the number of targets increases.

Figure 2.5 corresponds to minimum turn radius $\rho = 1$ and can be used to quantify our earlier statement that the bound (2.1) is not always sharp. We observe that, for the generated instances of the DTSP,

$$L_1^2(n) < 1.7 \cdot \mathrm{ETSP}(n), \quad n = 3, \ldots, 9, \tag{2.8}$$

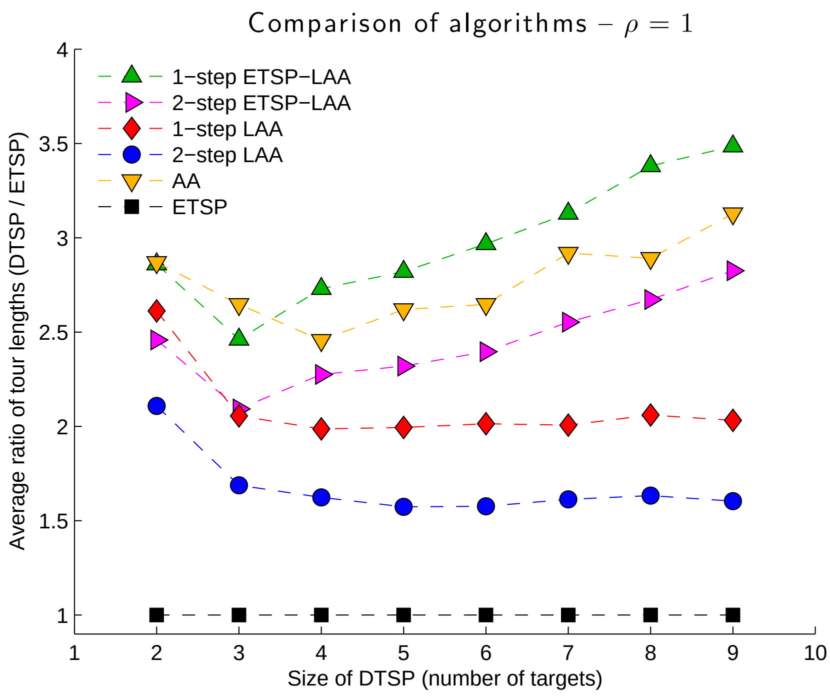


Figure 2.5: Comparison of five algorithms via Monte Carlo simulations. For each number of targets shown in the horizontal axis, the algorithms were applied to 100 randomly generated instances of the DTSP. For a fixed minimum turn radius equal to 1 and a fixed initial condition of the Dubins vehicle $(0, 0, \pi/2)$, the positions of the targets were generated with uniform distribution in the square $[-2.5, 2.5]^2 \subset \mathbb{R}^2$. The length of every DTSP tour was normalised (divided) by the length of the ETSP tour for the same set of targets and the vertical axis corresponds to the average normalised lengths of the DTSP tours.

on the average. For the sake of comparison, if we set $n = 9$ and ETSP(9) equal to the average value $\overline{\text{ETSP}}(9) = 13.387$ of the length of the 100 ETSP tours with 9 targets used to generate Figure 2.5, then the right-hand side of (2.1) becomes 96.891, whereas the right-hand side of (2.8) becomes 22.758.

In Figure 2.8, the five algorithms are compared on an instance of the DTSP where five targets and the initial condition lie on a circle of radius $r = 1.1$ and the minimum turn radius ρ is equal to 1. This example is chosen because it allows a comparison between the output of each algorithm and the optimal DTSP tour which is approximately a circle of radius r (Fig. 2.8e). Moreover, it demonstrates how reliance on the solution to the ETSP can lead to DTSP tours that are far from optimal. Specifically, the optimal ETSP tour is a polygon inscribed in the circle of radius r, however, if the targets are ordered clockwise (the orientation of a tour is immaterial for the ETSP) the k-step ETSP-LAA, for $k = 1, 2$, and the alternating algorithm output the tours shown in Fig. 2.8a, 2.8b, and 2.8c, respectively.

2.5.2 The 2-Opt 2-step look-ahead algorithm

For the simulations in this section, the 2-Opt k-step LAA was implemented for $k = 2$ and a simplified version of the 2-Opt algorithm for the ETSP was used to find an initial order σ_0 for the

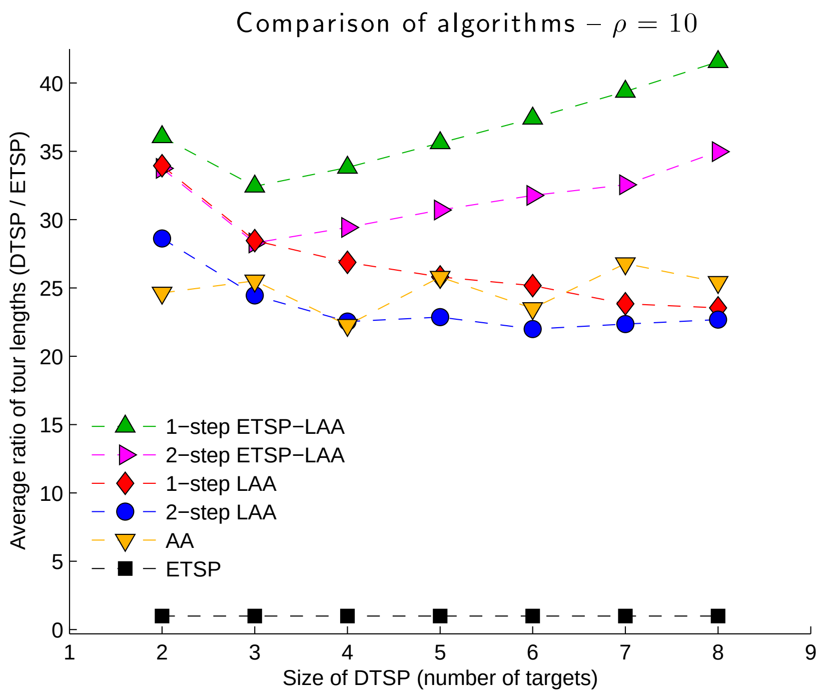


Figure 2.6: Same as Figure 2.5, but with $\rho = 10$.

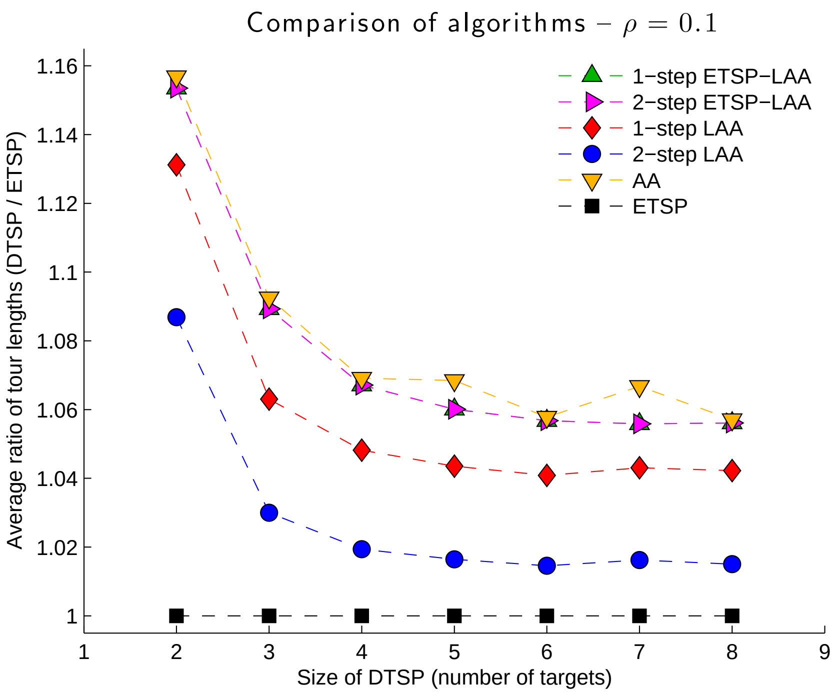
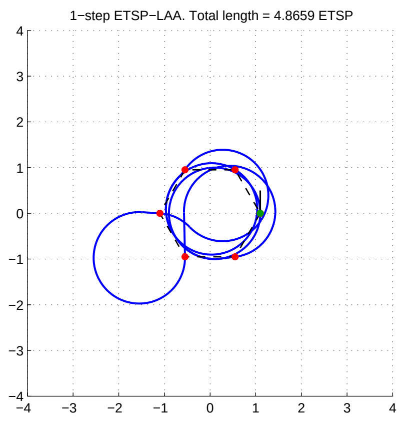
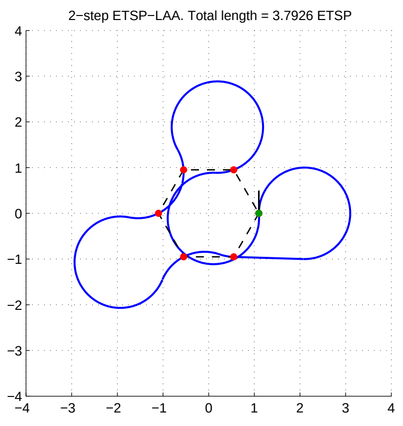


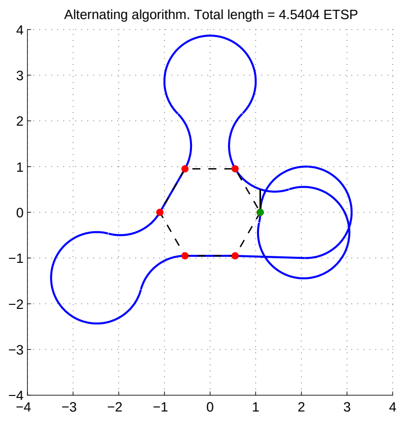
Figure 2.7: Same as Figure 2.5, but with $\rho = 0.1$. The scale of the vetical axis shows that, when the minimum turn radius is small compared to the average distance between the targets, the DTSP tours approximate the solution to the ETSP.



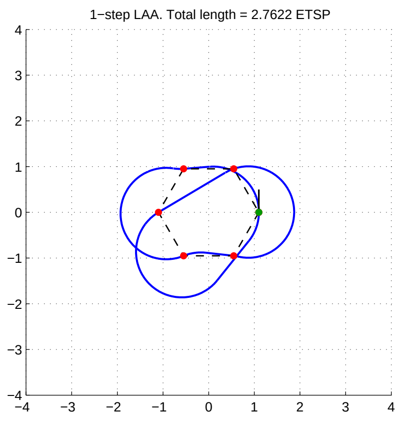
(a)



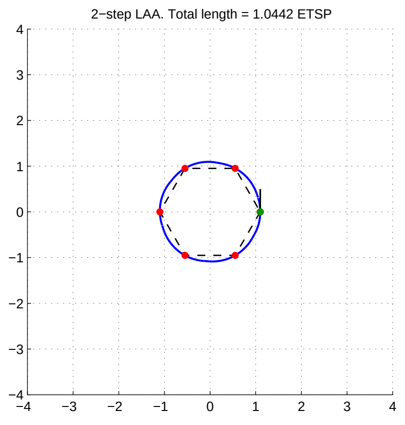
(b)



(c)

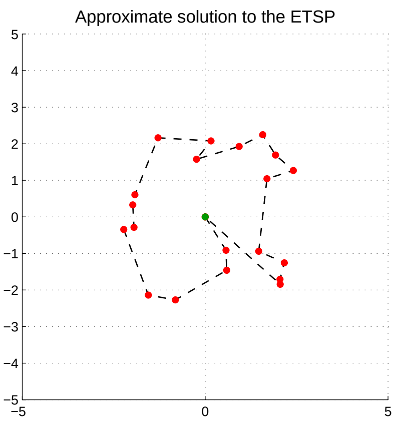


(d)

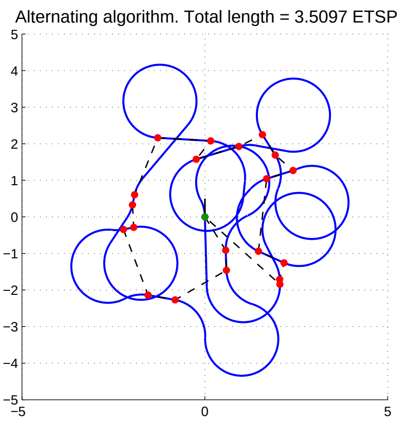


(e)

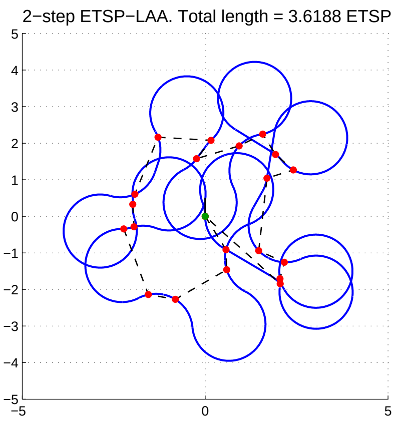
Figure 2.8: Comparison of five algorithms on an instance of the DTSP with the 5 targets and the initial condition lying on the circle $x^2 + y^2 = r^2$, $r = 1.1$. The minimum turn radius is $\rho = 1$ and the initial condition of the Dubins vehicle is $p = (1, 0, \pi/2)$. The DTSP tour generated by the 2-step LAA practically coincides with the optimal solution.



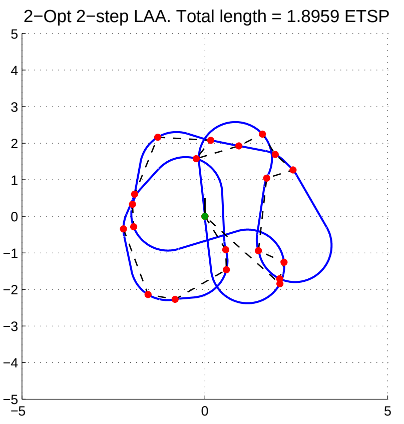
(a)



(b)



(c)



(d)

Figure 2.9: Comparison of the 2-Opt 2-step LAA, the 2-step ETSP-LAA, and the alternating algorithm on a single instance of the DTSP with 20 targets. Similarly to Figure 2.5, the minimum turn radius is $\rho = 1$, the initial condition of the Dubins vehicle is $p = (0, 0, \pi/2)$, and the positions of the targets are randomly and uniformly distributed in $[-2.5, 2.5]^2 \subset \mathbb{R}^2$. The ETSP tour in (a) is found using a simplified version of the 2-Opt algorithm and is not optimal. In (d) the maximum number $I_{\max}$ of 2-Opt moves equals 1000.

targets. The same order σ_0 was provided as input to the 2-step ETSP-LAA and the alternating algorithm. Because the 2-Opt algorithm is not guaranteed to find a global minimum, it should be noted that it is with respect to this approximate solution to the ETSP that the lengths were normalised. Applying the 2-Opt 2-step LAA to an instance of the DTSP with $n = 20$ and $I_{\max} = 1000$ requires less than five minutes. An example is shown in Figure 2.9. Besides experimenting with the value of $I_{\max}$, there are improvements that can be made in the implementation of the 2-Opt k-step LAA to reduce the execution time. First, the algorithm in Table 2.1 is easily parallelisable in a way that multiple threads can be executed almost independently. Second, when a 2-Opt move is applied to σ_0 to obtain a new order σ (step 5), the tours that correspond to σ_0 and σ can have a large part in common and the new tour need not always be computed anew. In general, this is the case if the 2-Opt move alters the latter part of σ_0.

Mulitple UAV scenarios

The problem considered in the second phase of the research includes allocating a group of vehicles to a given set of targets while taking into account the vehicles' kinematic constraints and avoiding collision with obstacles scattered in the environment. Each target is assigned with a time dependent value (referred to as the target benefit) that represents the target's importance and priority. The objective is to maximize a reward function which is the sum of all the benefits gathered by the group of vehicles.

3.1 Problem Formulation

3.1.1 Vehicles

Let $V = \{V_1, V_2, .., V_{N_V}\}$ be a set of unmanned aerial vehicles which need to complete the visit requirements of the given set of targets. The vehicles have a minimum turn radius and can move only forward. The kinematic constraints need to be accounted for when planning the vehicles' trajectory. The equations of motion are presented below:

Vehicle kinematics

$$\begin{aligned} \dot{x} &= v\cos\psi \\ \dot{y} &= v\sin\psi \\ \dot{\psi} &= \omega \end{aligned} \tag{3.1}$$

Motion constraints:

only forward motion is allowed

$$v = U \tag{3.2}$$

Turn rate constraint (Given a minimum turn radius)

$$|\omega| \leqslant U/R_{\min} \tag{3.3}$$

where (x,y) are the vehicle's Cartesian coordinates and ψ is the vehicle's orientation angle, U and ω are the vehicle's constant speed and maximum turn rate respectively.

3.1.2 Targets and Benefits

Let $T = \{T_1, T_2, .., T_{N_T}\}$ be the set of N_T stationary targets designated for the group of vehicles. Let $C = \{C_1, C_2, ..., C_{N_T}\}$ be the set of initial benefits assigned to each target and let $S = \{1, 2, ..., N_T\}$ be the set of stages in which a target is allocated as an assignment to a vehicle. The target's benefit represents the value granted to a vehicle for visiting the target. Since the benefits are time dependent, we propose a mathematical formulation which is referred to as the "benefit function". This formulation represents the reward granted to a vehicle for visiting a target, depending on the target's priority (represented by its initial benefits) and the time it takes the vehicle to get to the target from its initial position.
Let

$$t_{ik}^m = L_{ik}^m / V \tag{3.4}$$

be the time required for vehicle $i \in V$ to travel to target $k \in T$ at stage $m \in S$ from its current position. L_{ik}^m, V are the path length taken by vehicle i to visit target k at stage m and the vehicle constant speed, respectively. Note that L_{ik}^m depends on the assignment history of vehicle i prior to stage m. The assignment history is used to obtain the initial condition (position and orientation) required to calculate the vehicle's path. Let A be a user defined coefficient which defines the benefit function's descent rate. $\sum_{m=1}^{l} \sum_{k=1}^{N_T} t_{ik}^m$ is the total time required for vehicle i to travel to target k from its initial position at the current stage l . We use the descent rate coefficient and the total time passed until the vehicle visits the current target to formulate the benefit function as follows:

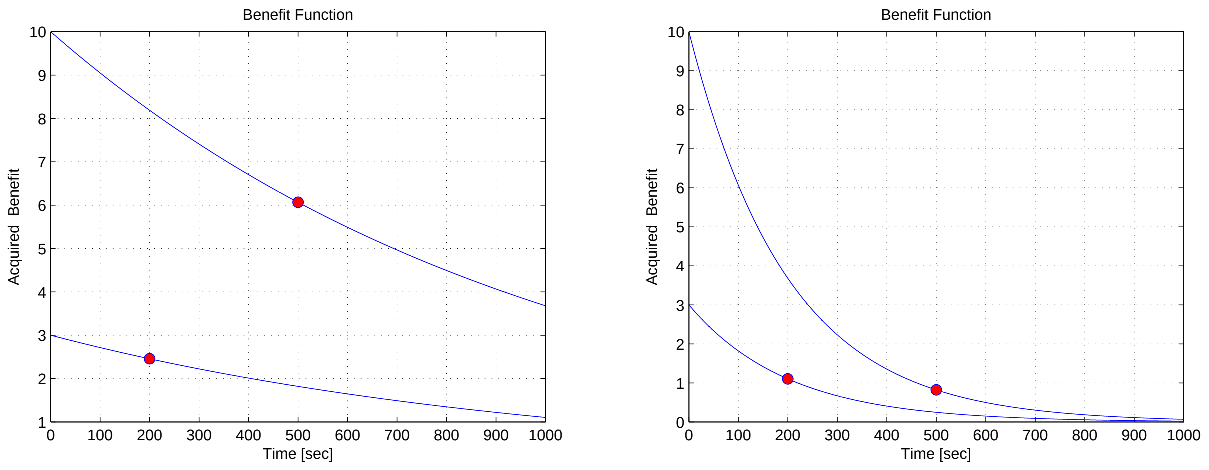
$$C_j e^{-A \sum_{m=1}^{l} \sum_{k=1}^{N_T} t_{ik}^m} \tag{3.5}$$

This formulation helps create a problem in which the vehicle assignments' order depends on the path to each target and not only on the target's initial priority (for example, the highest priority target is not necessarily visited first and the time to arrive at the target's location is also taken into consideration). In addition, the same formulation can be used to describe the example scenario that includes a UGS network and a team of unmanned vehicles used for intruder detection and identification. The vehicles' response time is taken into account by calculating the vehicles' path length, and the targets' different initial priority represents the order of the UGS triggering time. Since the time it takes a vehicle to reach a target depends on the vehicle's path length, the latter will be calculated using a motion planning subroutine, described in section 3.2. Figure 3.1a shows the benefit function's change over time, each curve begins with a different initial value (initial benefits 3 and 10). The benefit function is a monotonically decreasing function and as such, the initial value diminishes as time progresses. When the descent rate coefficient is changed (increased by 5 times), the benefit rapidly diminishes over time, as seen in Figure 3.1b. The increase of the descent rate may also cause a change in the targets assigned to each vehicle or a different order in which the assigned targets need to be visited.

3.1.3 Cost function

The objective is to complete the visit requirement (visiting the given set of targets once) so as to maximize a reward function. The reward function considered is the overall of benefits acquired by the vehicles:

$$J_1 = \sum_{i=1}^{N_V} \sum_{l=1}^{N_T} \sum_{j=1}^{N_T} [C_j e^{-A \sum_{m=1}^{l} \sum_{k=1}^{N_T} t_{ik}^m x_{ik}^m}] x_{ij}^l \tag{3.6}$$



(a) Benefit Function Over Time (b) Benefit Function Over Time - Increased Decent rate

Figure 3.1: Benefit Function

where $x_{ij}^l \in \{0,1\}$ be a binary decision variable that equals 1 if vehicle i visits target j at stage l. Since the benefit function diminishes with time, we formulate a "lost" benefit function which is given by $C_j - C_j e^{-A\sum_{m=1}^{l}\sum_{k=1}^{N_T} t_{ik}^m}$. The "lost" benefit function formulation allow the definition of a cost function which is the equivalent to the reward function defined above, but instead of maximizing the reward function the objective is to minimize the cost function. The cost function mathematical formulation is given by

$$J_2 = \sum_{i=1}^{N_V}\sum_{l=1}^{N_T}\sum_{j=1}^{N_T}[C_j - C_j e^{-A\sum_{m=1}^{l}\sum_{k=1}^{N_T} t_{ik}^m x_{ik}^m}]x_{ij}^l \tag{3.7}$$

The constraints of the problem are given by:

$$\sum_{l=1}^{N_T}\sum_{i=1}^{N_V} x_{ij}^l = 1, \;\; j = 1, \dots, N_T \tag{3.8}$$

$$\sum_{j=1}^{N_V}\sum_{j=1}^{N_T} x_{ij}^l = 1, \;\; \forall\, l = 1, \dots, N_T \tag{3.9}$$

Eq. (3.8) ensures that each target is visited once. Eq. (3.9) ensures that in each stage only one vehicle is assigned to a target.
In [31, 34, 26] somewhat similar problem involving multiple targets and vehicles was solved. The cost function used in the related works is the sum of the path lengths of all the vehicles and can be formulated as:

$$J_3 = \sum_{i=1}^{N_V}\sum_{l=1}^{N_T}\sum_{j=1}^{N_T} L_{ij}^l x_{ij}^l \tag{3.10}$$

In these cases the targets' importance is identical and it is ignored when solving the problem. In the simulation results' section we use this cost function to help compare the performance of the proposed algorithms.

The solution of the proposed problem includes solving two integrated subproblems: task assignment and motion planning problems. In order to minimize the cost function, the task assignment depends on the underlying motion planning for the path length, while the motion planning in turn depends on the task assignment for the order of the vehicle's targets. This makes the problems coupled.

The motion planning problem is presented next. We assume that each vehicle is assigned with a list of an ordered set of targets, made by the task assignment algorithm. The goal of the motion planning is to derive a trajectory for each vehicle to visit all targets on the list, avoid collision with obstacles and respect the vehicle kinematic constraints described in section 3.1.1. This goal should be accomplished in minimum time in order to maximize the benefit acquired from each target.

3.2 Motion planning

In order to represent the motion planning problem in the form of a decision tree it is necessary to generate nodes representing the following: targets position, vehicle's initial configuration and the obstacles' vertices (under the assumption of polygonal obstacles). The vehicle's path will either be a direct path (free of obstacles) connecting the initial configuration and the set of targets, or a path which also passes through some of the obstacles vertices, in case a direct path does not exist. Each branch of the tree represents the described path. The root node (initial configuration) is connected to all of the targets nodes and in case a direct path is not feasible, obstacles nodes are also included. The goal is to find the branch that provides the minimum time path.

The vehicles in this work are modeled as Dubins vehicles. The Dubins paths are concatenation of arcs of minimum radius turn and straight line segments which connect an initial and final configuration (position and orientation). The optimal path can be achieved by checking 6 possible path types for the Dubins vehicle [11]. In case the orientation angle in the final configuration is removed the number of possibilities is significantly reduced. This is known as the relaxed Dubins path that include only 4 [4].

An important benefit obtained by using the relaxed path is explained using the following example: When calculating the optimal path between an initial (node 1), final (node 5) configurations and three additional unordered configurations (for example: obstacle vertices) located between them (nodes 2-4), the following branches of the tree graph are generated: A branch connecting nodes 1-2-3-4-5 and a branch connecting nodes 1-2-4-3-5. In the relaxed case the path connecting nodes 1 and 2 should be calculated only once, as it is independent of the remaining nodes. However, in the non relaxed case, the arrival angle at node 2 depends on the order of the following nodes (node 3 or 4) and the path between node 1 and 2 needs to be calculated separately for each branch. This attribute, where the path between two nodes doesn't depend on the following nodes enables us to pose the problem as tree graph.

In order to find the relaxed optimal path which connects the initial configuration and the targets' set and doesn't intersect with obstacles, it is necessary to search the tree graph presented above. The search process includes calculating the relaxed path connecting the different graph nodes - obstacles' vertices or targets' positions. In this search process the calculation of the relaxed path is repeated multiple times, especially in large scale scenarios. In case real time scenarios are considered, the use of the relaxed path becomes highly beneficial since the computational complexity is significantly reduced, compared to the non relaxed case.

In this work an existing motion planning algorithm is used as a subroutine for the developed task assignment algorithm. The motion planning solution can be achieved by one of two algorithms:

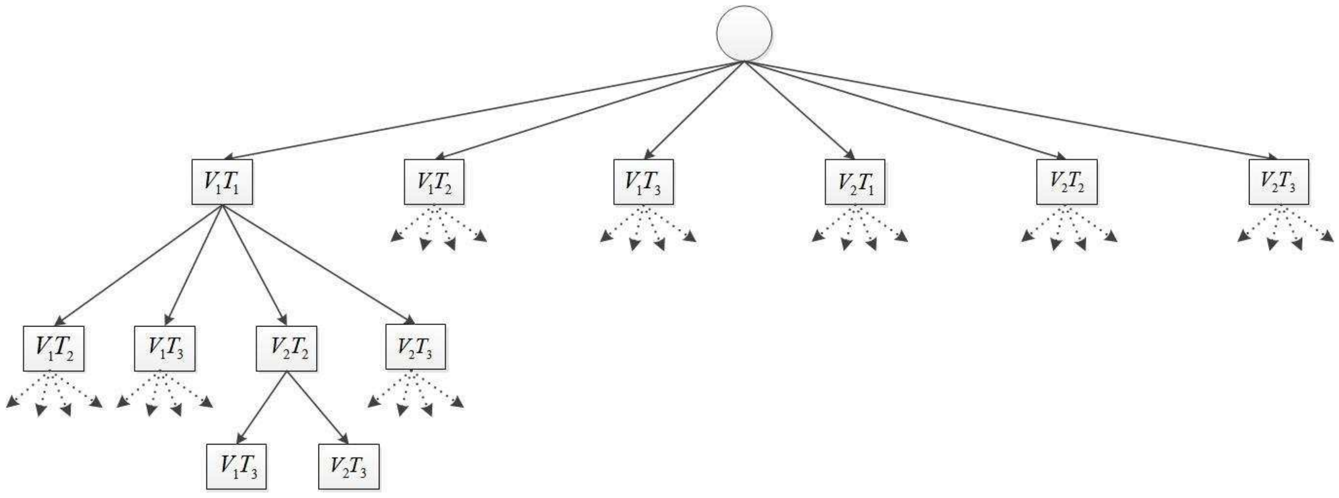


Figure 3.2: A Tree graph for 2 vehicles and 3 targets

1) An Exhaustive algorithm which explores every branch of the search tree and evaluates every possible visit order sequence in order to find the minimum cost one. 2) An A* like heuristic algorithm which uses Euclidean distances as a heuristic estimation and a greedy approach to find a feasible path. The algorithms are detailed in the Appendix. The product of these algorithms is a vehicle's feasible path, connecting an ordered set of waypoints among obstacles.

3.3 Task assignment

The task assignment problem is represented as a tree (as can be seen in Figure 3.2) by generating nodes that describe a combination of a vehicle V_i assigned to target T_j. Nodes are constructed until all combination of vehicles and targets have been taken into account. Each node is associated with a cost. For example: node V_iT_j has a cost that equals the "lost" benefit granted to vehicle i for visiting target j. The "lost" benefit value depends on the time that it takes vehicle i to reach target j from its initial position, which in turn depends on the vehicles' path length. The path is obtained using a motion planning subroutine (described in section 3.2) which guarantees feasible path for the vehicles. Since the motion planning subroutine is used in task assignment process, the problem solution consists of a primary task assignment tree search which in turn depends on a secondary motion planning tree search. Two algorithms that provide solutions to the task assignment problem are proposed, an exhaustive search algorithm and a greedy algorithm. The greedy algorithm provides a computationally fast solution which may not be optimal, and the exhaustive algorithm explores all the assignment possibilities to derive an assignments allocation with minimum cost value.

3.3.1 Exhaustive Task assignment Algorithm

The proposed algorithm which is describe in Algorithm 1, explores every branch of the tree to evaluate all the assignments possibilities. The input to the algorithm is configurations of all the vehicles, obstacles vertices locations, and targets positions. The algorithm is initialized by computing an upper bound (line 1), which is calculated by using the greedy algorithm described in section 3.3.2. The upper bound is used to prevent unnecessary exploration of branches by bounding the branching of the tree. In the first step of the algorithm the first layer of the tree is generated, all possible vehicle target combinations are described as nodes and the associated cost is calculated (lines 4-21). After the initial nodes are generated, the exhaustive search begins. A depth first search is used to expand a branch until a leaf node is generated (line 22-44). When a node is added, the accumulated cost of the branch is calculated and compared with the upper bound (line 28). In case the cost is higher, the branching is bounded. If a leaf node is reached and the upper bound is higher than the accumulated cost, the upper bound is updated and assignments' order for each vehicle is stored (line 34-38). The process is repeated until all branches have been either bounded or completely explored. The algorithm output is a minimum cost ordered set of targets, assigned to each vehicle.

3.3.2 Greedy Task Assignment Algorithm

The proposed algorithm is based on a greedy search method which enables to quickly find an assignment solution, it is described in Algorithm 2. Since the algorithm is greedy by nature, the objective function used is the reward function presented in Equation 3.6. The input to the algorithm is the configurations of all the vehicles, obstacles vertices locations, and targets' positions. In the first step of the algorithm, the benefit value of all possible pairs of vehicle-target are computed (lines 2-7). The vehicle of the pair with the highest value is assigned to the corresponding target (line 8). It is now assumed that the target is already visited and that the vehicle is at the target position (lines 9-12). Next, the benefit value of all the possible pairs is evaluated again (lines 14-19). The values are compared in order to find the pair with the highest one and to relocate the vehicle to the corresponding (now visited) target (line 20-24). This procedure is repeated until the visit requirements are fulfilled (line 13). The output of the algorithm is a target list allocated to each vehicle and the order in which the vehicles need to visit their assigned targets. Unlike the exhaustive algorithm, the assignments' order does not guarantee minimum cost since only a specific (and not necessarily the optimal) branch of the tree has been generated.

3.4 Simulation Results

In this section, sample-runs are used to demonstrate the presented algorithms and to explain the different parameters' (vehicle type, benefit's descent rate...) influence on the obtained solution. In all the different scenarios, the task assignment algorithms (exhaustive or greedy) use the motion planning subroutine (exhaustive or heuristic) based on the relaxed Dubins distances, hence the coupling of the problem is kept. The vehicles' turn radius is set to 60 [m] for illustrative reasons and the targets' initial benefit values can be set between 1,000 to 10,000 (the values are presented in the figures next to each target as numeral between 1 and 10). Each scenario has a summary table presenting the acquired and lost benefit, and the solution running time.

3.4.1 General Scenario

Figure 3.3 presents a scenario in which 2 aerial vehicles need to visit 4 targets with different initial benefits. The scenario solution is obtained using different algorithm setups in each case. Table 3.1 summarizes the results of the different sample runs. The highest benefit (lowest lost benefit) and longest running time were gained using an exhaustive algorithm setup (Figure 3.3a). When the heuristic motion planning is used instead (Figure 3.3c), the cost remains the same but the running time shortens. For the case of a greedy task assignment and heuristic motion planning

```
INPUT: Vehicles Configuration, Target position and and Obstacles vertices positions.
1: Upper bound=(Greedy task assignment algorithm solution)
2: Open list=[ ]    ▷ Initialize the Open list - stores the node to be examined further.
3: Target list=$T_j : j = 1 \dots N_T$    ▷ Initialize a list containing targets to be visited.
4: for ($V_i : i = 1\ to\ N_V$) do    ▷ Generate the first layer of the tree.
5:     for ($T_j : j = 1\ to\ N_T$) do
6:         Current node-Vehicle=$V_i$
7:         Current node-Target=$T_j$
8:         Current node-Path($V_i$)=Path Length($V_i$,$T_j$)    ▷ Calculate the path length between the current vehicle and target.
9:         Current node-Cost=Lost Benefit function(Current node-Path($V_i$))
                ▷ Calculate the Lost benefit value.
10:        Current node-Target list=Target list\$T_j$    ▷ Remove the current target from the target list of the expanded branch.
11:        Current node-vehicle target list($V_i$)= $T_j$    ▷ Store the targets assigned to each vehicle
12:        if  Current node-Target list = $\emptyset$ then    ▷ If a leaf node is reached and its cost is lower than the upper bound: update the upper bound and store the vehicle's assigned targets
13:            if Current node-Cost $\leq$ Upper bound then
14:                Upper bound=Current node-Cost($V_i$)
15:                Vehicle target list($V_i$)= $T_j$
16:            end if
17:        else    ▷ Add the current node to Open list for further exploration.
18:            Open list =Open list $\cup$ Current node
19:        end if
20:    end for
21: end for
```

Algorithm 1: Task assignment Exhaustive search algorithm

```
22: while Open list $\neq \emptyset$ do                                  ▷ Perform a Depth First Search
23:     Parent node=Open list(last inserted node)          ▷ Choose the last inserted node as the current parent node for further exploration
24:     for ($V_i$ : $i$=Parent node-Vehicle $to$ $N_V$) do          ▷ Consider all possible children nodes.
25:         for $T_j$ $\in$Parent node-Target list do
26:             Child node-Path($V_i$)=Path Length($V_i$,$T_j$)
                                  + Parent node-path($V_i$)   ▷ Calculate the overall distance traveled by vehicle-$V_i$ in the explored branch.
27:             Child node-Cost=Parent node-Cost              ▷ Calculate the cost
                          + Lost Benefit function(Child node-path($V_i$))
28:             if Child node-Cost $\leq$ Upper bound then        ▷ If the cost is higher than the upper bound, the branching is bounded
29:                 Child node-vehicle=$V_i$
30:                 Child node-target=$T_j$
31:                 Child node-Target list=Parent node-Target list$\backslash T_j$
32:                 Child node-vehicle target list($V_i$)= [Parent node-vehicle target list($V_i$), $T_j$]
33:                 if Child node-Target list = $\emptyset$ then      ▷ If a leaf node has been reached: update the upper bound and store the vehicle's assigned targets.
34:                     Upper bound=Child node-Cost($V_i$)
35:                     Vehicle target list($V_i$)=Child node-vehicle target list($V_i$)
36:                 else
37:                     Open list =Open list $\cup$ Child node       ▷ Add the child node to Open list for further exploration.
38:                 end if
39:             end if
40:         end for
41:     end for
42:     Open list =Open list $\backslash$ Parent node          ▷ Remove the evaluated Parent node from the Open list
43: end while
    OUTPUT: Vehicle target list - The targets assigned to each vehicle and the required visitation order.
```
Algorithm 1: Task assignment Exhaustive search algorithm (Continue)

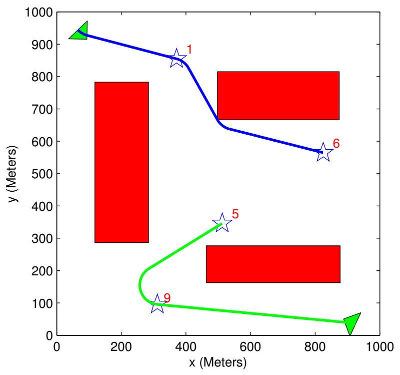
```
INPUT: Vehicles Configuration, Target position and and Obstacles vertices positions.
 1: Target list=$T_j : j = 1 \dots N_T$        ▷ Initialize a list containing targets to be visited.
 2: for ($V_i : i = 1$ to $N_V$) do            ▷ Calculate the benefits of all possible vehicle-target pairs.
 3:     for ($T_j : j = 1$ to $N_T$) do
 4:         Vehicle-target-path($V_i$,$T_j$)=Path Length($V_i$,$T_j$)
 5:         Vehicle-target-benefit($V_i$,$T_j$)=Benefit function(Vehicle-target-path($V_i$,$T_j$))
 6:     end for
 7: end for
 8: [$V_i$,$T_j$] $\leftarrow$ max(Vehicle-target-benefit($V_i$,$T_j$))   ▷ Find the pair with the highest benefit value.
 9: Vehicle target list($V_i$)=[Vehicle target list($V_i$), $T_j$]   ▷ Add the target to the vehicle assigned targets list.
10: Target list=Target list$\setminus T_j$     ▷ Remove the target from the target list.
11: Vehicle position($V_i$)=Target position($T_j$)   ▷ Update the vehicle position.
12: Vehicle path($V_i$)=Vehicle path($V_i$)
                + Vehicle-target-Path($V_i$,$T_j$)   ▷ Calculate the overall distance traveled by the vehicle .
13: Total benefit=max(Vehicle-target-benefit($V_i$,$T_j$))   ▷ Store the accumulated benefit.
```

Algorithm 2: Task assignment Greedy search algorithm

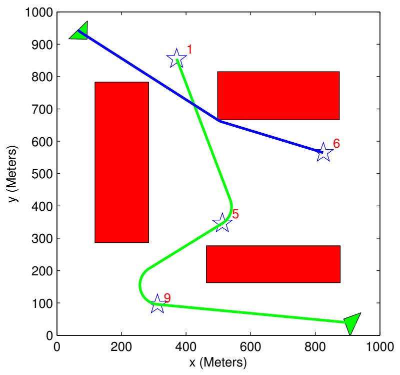
```
14: while Target list $\neq \emptyset$ do                              ▷ Repeat the process until all targets have been visited.
15:     for ($V_i : i = 1 \ to \ N_V$) do                              ▷ Calculate the benefits of all possible vehicle-target pairs.
16:         for ($T_j \in$ Target list) do
17:             Vehicle-target-path($V_i$,$T_j$)=Path Length($V_i$,$T_j$) + Vehicle path($V_i$)
18:             Vehicle-target-benefit($V_i$,$T_j$)=Benefitfunction(Vehicle-target-path($V_i$,$T_j$))
                                         + Total benefit
19:         end for
20:     end for
21:     [$V_i$,$T_j$] $\leftarrow$ max(Vehicle-target-benefit($V_i$,$T_j$))    ▷ Find the pair with the highest benefit value.
22:     Vehicle target list($V_i$)=[Vehicle target list($V_i$), $T_j$]      ▷ Store the vehicle assigned target.
23:     Target list=Target list\$T_j$                                    ▷ Remove the target from the target list.

24:     Vehicle position($V_i$)=Target position($T_j$)                   ▷ Update the vehicle position.
25:     Vehicle path($V_i$)=Vehicle path($V_i$)
                    + Vehicle-target-path($V_i$,$T_j$)               ▷ Calculate the overall distance traveled by the vehicle.
26:     Total benefit=max(Vehicle-target-benefit($V_i$,$T_j$))          ▷ Store the accumulated benefit.
                + Total benefit
27: end while
    OUTPUT: Vehicle target list - The targets assigned to each vehicle and the required visitation order.
```

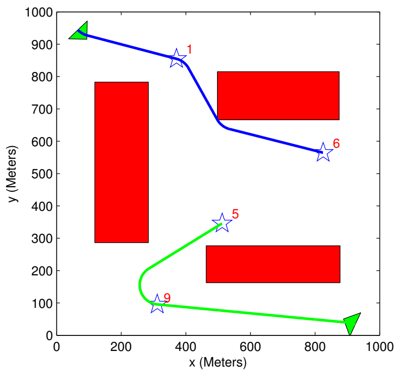
Algorithm 2: Task assignment Greedy search algorithm (Continue)



(a) Exhaustive Task assignment algorithm; Exhaustive Motion planning algorithm



(b) Greedy Task assignment algorithm; Heuristic Motion planning algorithm



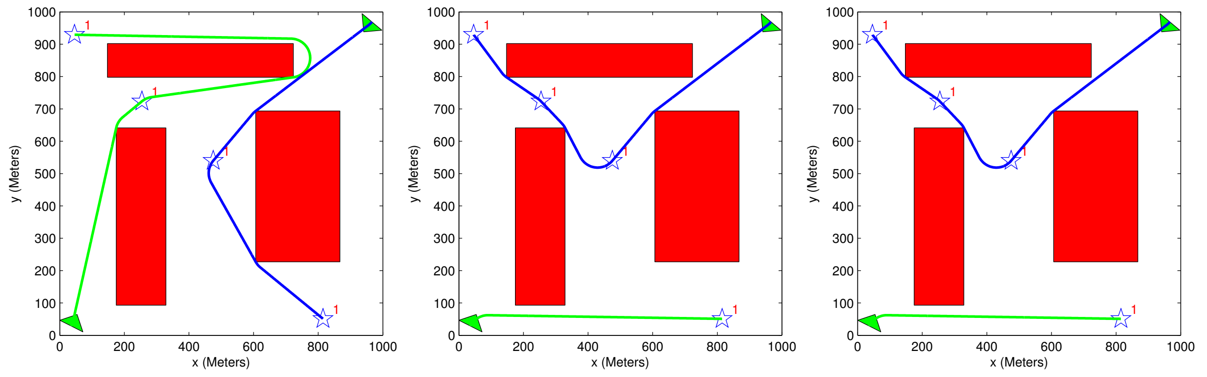
(c) Exhaustive Task assignment algorithm; Heuristic Motion planning algorithm

Figure 3.3: General scenario - 2 Vehicles and 4 Targets among obstacles

Figure #	Algorithms used	Initial Benefit	Acquired Benefit	Lost Benefit	Overall Distance	Solution Time[sec]
3.3a	Exhaustive TA Exhaustive MP	21000	9814	11186	1930	58.1
3.3b	Greedy TA Heuristic MP	21000	9397	11603	2438	1.05
3.3c	Exhaustive TA Heuristic MP	21000	9814	11186	1930	17.3

Table 3.1: Different Initial Benefits Scenario

algorithms (Figure 3.3b), the lowest benefit (highest cost) and shortest running time are attained. The solution presented in Figure 3.3a shows that the vehicles are generally first heading towards targets with high priority while taking into account targets with lower priority. Since the benefit diminishes with time, the vehicle does not head directly towards the high priority targets but also passes through low priority targets which are closer to its location (upper vehicle on Figure 3.3a). In Figure 3.3b the upper vehicles head directly to target 6 (initial benefit value=6) and skip target 1 since in this case the task assignment algorithm is greedy by nature. In the scenarios presented in sections 3.4.2 and 3.4.3 the exhaustive algorithms' setup yields the same results as the exhaustive task assignment algorithm and heuristic motion planning algorithm setup, therefore only latter setup is presented. Even though the results presented in these sections are identical, it can be shown that in certain cases the exhaustive algorithms' setup provides better results however, the solution running time becomes longer.



(a) Greedy Task assignment algorithm; Heuristic Motion planning algorithm

(b) Exhaustive Task assignment algorithm; Heuristic Motion planning algorithm

(c) Exhaustive Task assignment algorithm; Exhaustive Motion planning algorithm; Minimize the sum of the overall distance travelled

Figure 3.4: Equal Benefits scenario - 2 Vehicles and 4 Targets among obstacles

3.4.2 Equal Benefit Scenario

The scenario shown in Figure 3.4 is similar to the scenario shown in Figure 3.3, only the targets' initial benefit is equal. Since the targets' priority is identical, we expect the results to be similar to the case where the cost function objective is to minimize the overall distance travelled by the vehicles (Equation 3.10). In the results summarized in table 3.2, the highest benefit (lowest cost) is obtained by the setup of the exhaustive task assignment algorithm (Figure 3.4b). The overall distance is the same as in the case of using the cost function which minimizes the sum of the distance travelled (Figure 3.4c), as we expected. As in the previous scenario, the solution running time has the same tendency, when using a greedy and heuristic algorithms' combination the shortest running time is gained and with an exhaustive algorithms combination the longest running time is gained. This tendency remains the same through all the presented scenarios.

3.4.3 Comparing Exhaustive and Greedy Task Assignment Algorithms

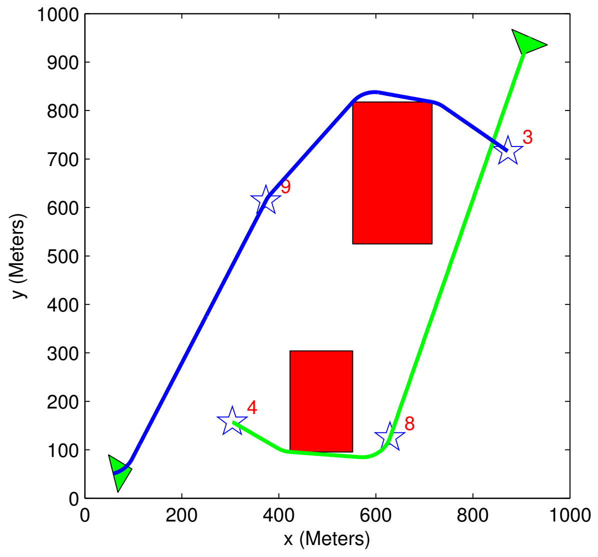
A scenario of 4 targets and 2 vehicles is presented in Figure 3.5. The results of the scenario (table 3.3) support the claim that the exhaustive algorithm always provides better or equal results compared to the greedy algorithm. This can be seen in table 3.1 and table 3.4 as well. The main advantage of the greedy algorithm is its low computational time, which makes it suitable to real time applications. In cases where both the exhaustive task assignment algorithms and the motion planning algorithm are used, the best solution coded in the tree is obtained and the lowest cost assignments allocation and vehicles' paths are provided.

3.4.4 Benefit Time Dependency

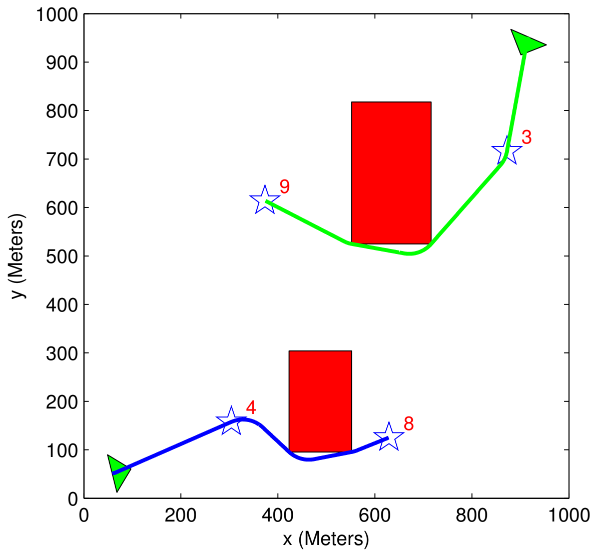
A simple scenario of one vehicle and two targets (initial benefit of 10 and 5) is used in Figure 3.6 and Figure 3.7. The time dependency can be easily explained using this two scenarios, in Figure 3.6 the vehicle's path passes through target 5 even though it causes the vehicle to extend its path toward target 10. This happens because the time it takes to get to target 5 is very short compared to target 10, and in order to minimize the "lost benefit" of the two targets it is better to first pass

Figure #	Algorithms used	Initial Benefit	Acquired Benefit	Lost Benefit	Overall Distance	Solution Time[sec]
3.4a	Greedy TA Heuristic MP	4000	1399	2601	3353	1.3
3.4b	Exhaustive TA Heuristic MP	4000	1623	2377	2075	73.7
3.4c	Exhaustive TA Heuristic MP (Sum of path length cost function-Eq. 3.10)	4000	1623	2377	2075	73.7

Table 3.2: Equal Initial Benefits Scenario



(a) Greedy Task assignment algorithm; Heuristic Motion planning algorithm



(b) Exhaustive Task assignment algorithm; Heuristic Motion planning algorithm

Figure 3.5: Comparison Between Exhaustive and Greedy Task Assignment Algorithms

Figure #	Algorithms used	Initial Benefit	Acquired Benefit	Lost Benefit	Overall Distance	Solution Time[sec]
3.5a	Greedy TA Heuristic MP	24000	10027	13973	2514	0.9
3.5b	Exhaustive TA Heuristic MP	24000	13504	10496	1487	13.6

Table 3.3: Comparing Exhaustive and Greedy Task Assignment Algorithms

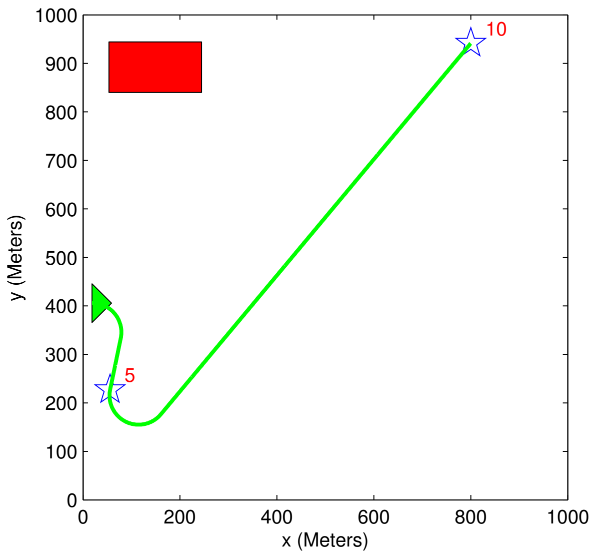


Figure 3.6: Scenario 1 - Exhaustive TA ; Exhaustive MP and Greedy TA ; Heuristic MP

through target 5. In Figure 3.7, however, the time it takes to get to target 5 is still shorter than the time it takes to get to target 10 but since target 10 is now located closer to the vehicle it is better to first pass though target 10. These two scenarios demonstrate how the arrival time of the vehicle to each target influences the task assignment process. In both of these small sized simple cases the greedy algorithm provides an identical result to the exhaustive algorithm's result and the vehicle's path remains the same.

3.4.5 Benefit Descent Rate

The scenario of two targets and one vehicle presented in Figure 3.8 helps illustrate the influence of the descent rate on the obtained results. In this scenario, the time it takes the vehicle to get to target 3 is 200 [sec] from its initial position and the time it takes the vehicle to get to target 10 is 500 [sec] from the same position. By increasing the value of the descent rate, the benefit rapidly diminishes as time progresses. In Figure 3.8b the descent rate is increased by 5 times compared to Figure 3.8a, which causes the change in the vehicle assignments' order. Before the increase of the descent rate, the benefit of target 10 is significantly higher than that of target 3 (upper red bullet compared to lower red bullet in Figure 3.1a) but after the descent rate is increased, the targets have similar benefits (as can be seen by the red bullets' vertical position in Figure 3.1b). Since the

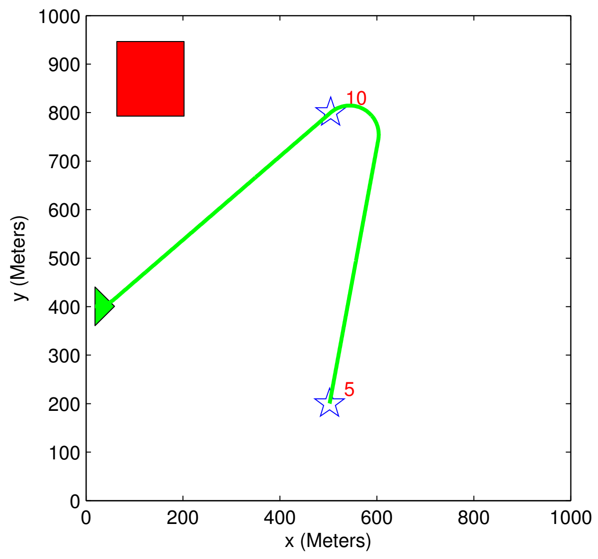
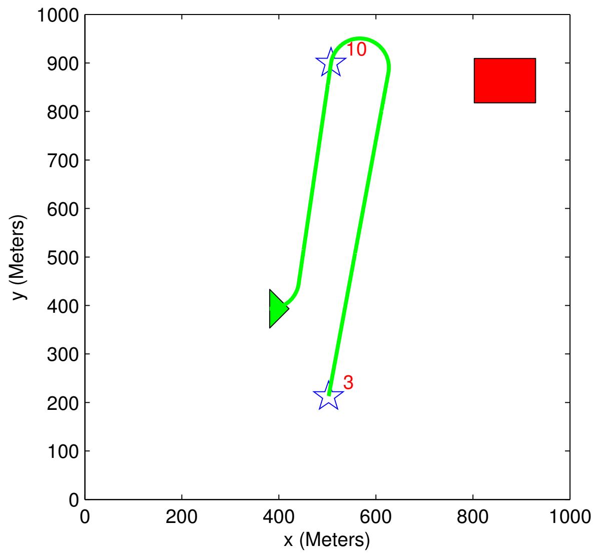


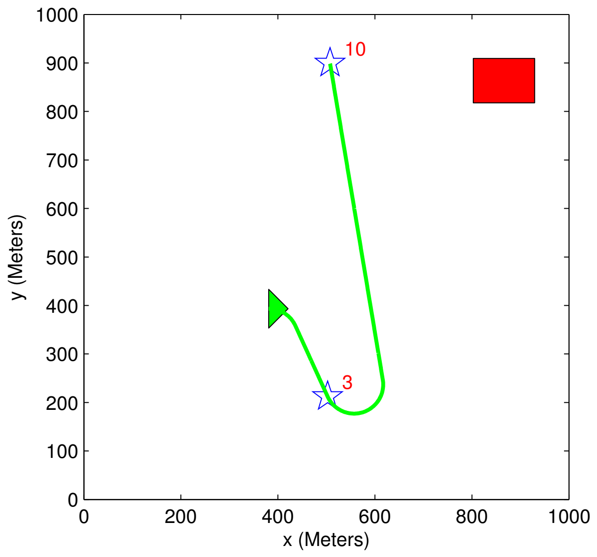
Figure 3.7: Scenario 2 - Exhaustive TA ; Exhaustive MP and Greedy TA ; Heuristic MP

Figure #	Algorithms used	Initial Benefit	Acquired Benefit	Lost Benefit	Overall Distance
3.6	Exhaustive TA Exhaustive MP	15000	6571	8529	1371
3.6	Greedy TA Heuristic MP	15000	6571	8529	1371
3.7	Exhaustive TA Exhaustive MP	15000	6634	8366	1333
3.7	Greedy TA Heuristic MP	15000	6634	8366	1333

Table 3.4: Scenario 1 & Scenario 2



(a) Exhaustive Task assignment algorithm; Exhaustive Motion planning algorithm



(b) Exhaustive Task assignment algorithm; Exhaustive Motion planning algorithm

Figure 3.8: Benefit Decent Rate

Figure #	Algorithms used	Initial Benefit	Acquired Benefit	Lost Benefit	Overall Distance
3.8a	Exhaustive TA Exhaustive MP	13000	6528	6472	1414
3.8b	Exhaustive TA Exhaustive MP	13000	1004	11996	1063

Table 3.5: Benefit Decent Rate

benefit that the vehicle can gather in target 10 is smaller than the one in target 3, it is preferable to change the targets' visitation order as can be seen in Figure 3.8b. The results summarized in table 3.5 emphasize the influence of the descent rate as the benefit acquired in Figure 3.8a is much higher than the one in Figure 3.8b. The benefit decent rate does not only influence the benefit gathered but may also influence the assignments' order as presented above.

Appendix

The task assignment algorithm uses a motion planning subroutine. The motion planning subroutine includes two types of algorithms. These algorithms are based on a search process that explores the tree described in section 3.2. The two algorithms are described next, an exhaustive search algorithm that finds the minimum cost branch, and a heuristic greedy algorithm that finds a feasible solution in a shorter computational time.

Heuristic motion planning algorithm

Given a set of ordered targets, vehicle's initial configuration and the obstacles' vertices position, the algorithm performs an A*-like greedy search based on Euclidean distances heuristics. The algorithm steps are as follows:

1. The vehicle is assumed to be located and oriented according to the initial configuration. The first target is set to be the current target.
2. Calculate an obstacles free relaxed path connecting the vehicle's current configuration and the current target. If such a path exist skip to step 7.
3. Calculate an obstacles free relaxed paths, connecting the current configuration and all of the the obstacles vertices.
4. Find the vertex with the following property: the sum of the relaxed path length (connecting the vertex and the vehicle's current configuration) and the Euclidean distance (connecting the vertex and the current target) is minimum.
5. The vehicle is assumed to be located at the vertex with the described property and the orientation angle becomes the vertex arrival angle.
6. Return to step 2.
7. The vehicle is assumed to be located at the current target and the orientation angle is set to be the arrival angle.
8. The next target to visit on the targets' set becomes the current target.
9. Repeat steps 2 - 8 until the entire targets' set is visited.

The algorithm output is a vehicle trajectory, represented by an ordered set of nodes (including targets and obstacles' vertices) that need to be visited using relaxed paths.

Exhaustive motion planning algorithm

The algorithm explores every branch of the search tree and evaluates every possible visit order sequence in order to find the minimum cost branch. The algorithm's input is a set of ordered targets, vehicle's initial configuration and the obstacles' vertices position. The algorithm steps are as follows:

1. Calculate the initial upper bound, using the heuristic algorithm.

2. An OPEN list is generated to store the nodes that will be examined as the next node to visit.
3. The initial configuration is entered to OPEN as a node.
4. The node with the lowest cost in OPEN (relaxed path connecting the initial configuration and the node) is selected as the current node.
5. The neighbors of the current node that can come after it in the visit order are examined. Their estimated distance is calculated. It is defined as the sum of the following:
   (a) Cost of the selected node.
   (b) Relaxed path connecting the selected node and the neighbor.
   (c) Euclidean distance between the neighbor and the current target to visit.
   (d) The total Euclidean distance which connects the current target and the remaining targets to visit in the targets' set in the required order.
6. The neighbors with an estimated distance which is lower than the current upper bound are added to OPEN as new nodes.
7. All of the new nodes added to OPEN are examined.
   (a) In case a new node is the next target to visit, the target is marked as visited in the current explored branch.
   (b) In case a new node is the last target to visit and the entire targets' set is visited in the required order, a leaf node of the branch is reached and the entire branch is explored.
      i. The upper bound is updated to the relaxed path total length described by the nodes in the branch and the visit order of the nodes is stored.
8. The current node is removed from OPEN since all the neighbors have been evaluated.
9. This process is repeated until the OPEN list is empty.

The algorithm output is identical to the heuristic algorithm output described above.

Bibliography

[1] Boost C++ libraries. `http://www.boost.org`.

[2] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[3] J.-P. Aubin. *Viability Theory.* Birkhäuser, 1991.

[4] J.-D. Boissonnat and X.-N. Bui. Accessibility region for a car that only moves forwards along optimal paths. Rapport de recherche 2181, INRIA Sophia Antipolis, Janvier 1994.

[5] J.-D. Boissonnat, A. Cérézo, and J. Leblond. Shortest paths of bounded curvature in the plane. Robotique, Image et Vision 1503, INRIA Sophia Antipolis, Juillet 1991.

[6] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Management Sciences Research Report 388, Carnegie-Mellon University, February 1976.

[7] I. Cohen, C. Epstein, and T. Shima. On the discretized Dubins traveling salesman problem. Available online at `http://ie.technion.ac.il/Home/Users/icohen/DTSP_140614.pdf`.

[8] M. S. Cons, T. Shima, and C. Domshlak. Integrating task and motion planning for unmanned aerial vehicles. *Unmanned Systems*, 2(1):1–20, 2013.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms.* MIT Press, 2001.

[10] L. Doyen and M. Quincampoix. Multi-target control problems. *Journal of Optimization Theory and Applications*, 93(1):121–139, April 1997.

[11] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, pages 497–516, 1957.

[12] E. Edison and T. Shima. Integrated task assignment and path optimization for cooperating uninhabited aerial vehicles using genetic algorithms. *Computers & Operations Research*, 38(1):340–356, 2011.

[13] A. L. Henry-Labordere. The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem. *Revue Francaise D' Informatique De Recherche Operationnelle*, 3(NB 2):43, 1969.

[14] P. Isaiah and T. Shima. A task and motion planning algorithm for the Dubins travelling salesperson problem. In *Proceedings of the 19th IFAC World Congress*, volume 19, pages 9816–9821. International Federation of Automatic Control, August 2014.

[15] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, pages 215–310, 1997.

[16] H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation algorithms for asymmetric tsp by decomposing directed regular multigraphs. *Journal of the ACM*, 52(4):602–626, July 2005.

[17] X. Ma and D. A. Castañón. Receding horizon planning for dubins travelling salesman problems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 5453–5458, San Diego, CA, USA, December 13-15 2006.

[18] T. G. McGee and J. K. Hedrick. Path planning and control for multiple point surveillance by an unmanned aircraft in wind. In *Proceedings of the 2006 American Control Conference*, pages 4261–4266, June 2006.

[19] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity.* Dover books on mathematics. Dover Publications, 1998.

[20] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mischenko. *The Mathematical Theory of Optimal Processes.* John Wiley & Sons, 1962.

[21] S. Rathinam, R. Sengupta, and S. Darbha. A resource allocation algorithm for multivehicle systems with nonholonomic constraints. *IEEE Transactions on Automation Science and Engineering*, 4(1):98–104, January 2007.

[22] K. Savla, E. Frazzoli, and F. Bullo. Travelling salesperson problems for the dubins vehicle. *IEEE Transactions on Automatic Control*, 53(6):1378–1391, July 2008.

[23] H. Sussmann and G. Tang. Shortest paths for the reeds-shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. *Rutgers Center for Systems and Control Technical Report*, 10:1–71, 1991.

[24] Z. Tang and Ü. Özgüner. Motion planning for multitarget surveillance with mobile sensor agents. *IEEE Transactions on Robotics*, 21(5):898–908, October 2005.

[25] G. Yang and V. Kapila. Optimal path planning for unmanned air vehicles with kinematic and tactical constraints. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 1301–1306, December 2002.

[26] Shima, T., Rasmussen, S., Sparks, A., and Passino, K., "Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms," *Computers & Operations Research*, Vol. 33, No. 11, 2006, pp. 3252–3269.

[27] Richards, A., Bellingham, J., Tillerson, M., and How, J. P., "Coordination and Control of Multiple UAVs," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2002-4588, Washington, DC, 2002.

[28] Schumacher, C., Chandler, P. R., Pachter, M., and Pachter, L. S., "Optimization of Air Vehicles Operations using Mixed-Integer Linear Programming," *Journal of the Operational Research Society*, Vol. 58, 2007, pp. 516–527, DOI:10.1057/palgrave.jors.2602176.

[29] Chandler, P. R., Pachter, M., Rasmussen, S. J., and Schumacher, C., "Multiple Task Assignment for a UAV Team," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2002-4587, Washington, DC, 2002.

[30] Schumacher, C. J., Chandler, P. R., and Rasmussen, S. J., "Task Allocation for Wide Area Search Munitions," *Proceedings of the American Control Conference*, Evanson, IL, 2002, pp. 1917–1922.

[31] Edison, E. and Shima, T., "Integrated Task Assignment and Path Optimization for Cooperating Uninhabited Aerial Vehicles Using Genetic Algorithms," *Computers and Operations Research*, Vol. 38, 2011, pp. 340–356.

[32] Shaferman, V. and Shima, T., "Unmanned aerial vehicles cooperative tracking of moving ground target in urban environments," *Journal of guidance, control, and dynamics*, Vol. 31, No. 5, 2008, pp. 1360–1371.

[33] Rasmussen, S. J. and Shima, T., "Tree Search Algorithm for Assigning Cooperating UAVs to Multiple Tasks," *International Journal of Robust And Nonlinear Control*, Vol. 18, No. 2, 2008, pp. 135–153, DOI:10.1002/rnc.1257.

[34] Shima, T., Rasmussen, S., and Gross, D., "Assigning Micro UAVs to Task Tours in an Urban Terrain," *IEEE Transactions on Control System Technology*, Vol. 15, No. 4, 2007, pp. 601–612, DOI:10.1109/TCST.2007.899154.

[35] Karaman, S. and Frazzoli, E., "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, Vol. 30, No. 7, 2011, pp. 846–894, DOI:10.1177/0278364911406761.

[36] Kavraki, L., Svestka, P., Latombe, J., and Overmars, M., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, Vol. 12, No. 4, 1996, pp. 566–580.

[37] Donald, B., Xavier, P., Canny, J., and Reif, J., "Kinodynamic motion planning," *Journal of the ACM (JACM)*, Vol. 40, No. 5, 1993, pp. 1048–1066.

[38] Agarwal, P. K. and Wang, H., "Approximation Algorithms for Curvature-Constrained Shortest Paths," *SIAM Journal on Computing*, Vol. 30, No. 6, 2001, pp. 1739–1772, DOI:10.1137/S0097539796307790.

[39] Backer, J. and Kirkpatrick, D., "A Complete Approximation Algorithm for Shortest Bounded-Curvature Paths," *Proceedings of the 19th International Symposium on Algorithms and Computation*, Surfers Paradise, Australia, 2008, pp. 628–643.

[40] Jacobs, P. and Canny, J., "Planning smooth paths for mobile robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, USA, 1989.

[41] Laumond, J.-P., Jacobs, P., Taix, M., and Murray, R., "A Motion Planner for Non-holonomic Mobile Robot," *IEEE Transactions on Robotics and Automation*, Vol. 10, No. 5, 1994, pp. 577–593, DOI:10.1109/70.326564.

[42] Delle Fave, F., Rogers, A., Xu, Z., Sukkarieh, S., and Jennings, N., "Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection," *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, 2012, pp. 469–476.

[43] JIANG, L. and ZHANG, R., "An Autonomous Task Allocation for Multi-robot System," *Journal of Computational Information Systems*, Vol. 7, No. 11, 2011, pp. 3747–3753.

[44] Shetty, V., Sudit, M., and Nagi, R., "Priority-based assignment and routing of a fleet of unmanned combat aerial vehicles," *Computers & Operations Research*, Vol. 35, No. 6, 2008, pp. 1813–1828.

[45] Krishnamoorthy, K., Casbeer, D., Chandler, P., Pachter, M., and Darbha, S., "UAV search amp; capture of a moving ground target under delayed information," *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, 2012, pp. 3092–3097.

[46] Soueres, P., Fourquet, J.-Y., and Laumond, J.-P., "Set of reachable positions for a car," *IEEE Transactions on Automatic Control*, Vol. 39, 1994, pp. 1626–1630, DOI:10.1109/9.310037.