AN EFFICIENT WAY FOR PATH PLANNING OF COOPERATIVE AUTONOMOUSLY SOARING GLIDERS

BY

MUHAMMAD ANEEQ UZ ZAMAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Naira Hovakimyan

ABSTRACT

This thesis attempts to solve the problem of planning paths for a group of gliding UAVs performing a task. These gliders have start and goal configurations (positions and orientations) in 2-dimensional space and also a starting altitude. This can be thought of as the starting energy of the glider. The task, given to the gliders, is to visit a set of *interest points* in 2-dimensions. Since the gliders start with a limited energy and are constantly losing it, the paths should be planned such that the energy lost while traveling over the paths is minimized. Moreover, exploitation of free energy present in the environment, called *Autonomous Soaring*, can also be used to maximize the range of the aircraft, potentially allowing the gliders to visit even more interest points.

The task of planning paths for the gliders is decoupled into two parts (i) planning the best sequence of waypoint visitation (for each glider) and, (ii) planning paths over these sequences. This decoupled approach results in increased computational efficiency of the framework.

The first section of the thesis deals with assignment and sequencing of waypoints for each glider, such that the cumulative energy lost by the team of gliders is minimized. This section uses an estimate of the actual energy, spent by the gliders going from point to point. The second section deals with planning paths over this sequence of waypoints, such that the dynamic constraints of the gliders are respected and the energy lost by each glider, over the course of its mission, is minimized.

Each section starts with a review of the literature relevant to that topic. The problem is formulated in a rigorous way and is followed by the proposed solution. Any theoretical guarantees which follow from the proposed solution are stated and proved. After which simulation results are presented.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my adviser, Professor Naira Hovakimyan, for encouraging me to take on challenging problems and ideas, creating opportunities for me that help me grow and providing me with guidance when it was most needed. I would also like to thank her for allowing me into her research lab which is full of wonderful people.

I would also like to thank my lab mates for being tremendously helpful and fun to work with. I would specifically like to thank Syed Bilal Mehdi for introducing me firstly, to this research group and secondly, to the idea of Cooperative Autonomous Soaring which took hold of my imagination and later lead to this research. I am thankful to Enric Xargay for his thoughtful suggestion about including interest points in the problem formulation. I am grateful to Ronald Choe for the great pains he took to peruse my research and help me make it better. I am also indebted to him for lending me his codes and his expertise of Pythagorean Hodograph Beziér curves. I would also like to thank Professor Cedric Langbort for being so approachable and for the many thought provoking discussions we had on this research.

Finally, I would like to thank my family for standing by me through thick and thin. I thank them for their advice in times of need and their unconditional love and support throughout my stay in the US.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

DTSP	Dynamic Travelling Salesman Problem
VSP	Visitation Sequence Planning
EPP	Energy-preserving Path Planning
ETOP	Euclidean Team Orienteering Problem
MTGS	Multi-Tier Graph Search
BLGS	Bi-Level Graph Search
LPA*	Lifelong Planning A*
PH	Pythagorean Hodograph

CHAPTER 1

INTRODUCTION

In 1994 NASA started the Environmental Research Aircraft and Sensory Technology (ERAST) Program with the objective of developing high endurance unmanned aircraft for high altitude missions. The principle technology being used to increase endurance was solar power. To maximize the flight endurance, the weight of the prototype was optimized to the limit. This compromised the structural integrity of the prototype and it broke up in mid-flight near the Hawaiian Island of Kauai in 2003 [1]. This leads us to the realization that to increase range and endurance of glider flight, other forms of renewable energy must be explored, which do not impose strict limitations on the structure of the gliders and are also dependable.

Autonomous Soaring is the exploration and exploitation of free energy present in the environment (like convective air flow and wind shear), by a glider-like UAV to increase its flight range endurance. Manned Soaring was first investigated in 1958 in [2], [3] but research on the problem of autonomous soaring [4], [5], [6] and thermal centering [7] has recently gained momentum. Cooperative autonomous soaring defines a group of aircraft performing a task cooperatively while soaring and has been researched by authors of [8], [9]. The ultimate vision for Cooperative Autonomous Soaring is to have groups of gliders roaming worldwide for years on end without any need for human intervention.

There are several applications for these high endurance missions like

- Surface Monitoring
- Earth Systems Monitoring
- Extended Communications Coverage
- Forest Fire Detection

- Oceanographic Research
- Disaster Recovery

The aim of the research presented in this thesis is to plan trajectories for a group of gliders, which are performing a task, while soaring autonomously. The task is to visit a number of points. These points are called *interest points*. The gliders have a starting energy which is lost as the gliders travel. The ability of these gliders to visit all the interest points might be restricted by their starting energies. As we know there are certain phenomena available in the environment that can provide energy and hence increase the range to the gliders. These phenomena are called *thermals* and their locations are known a priori. The problem now is to plan a path over the interest points and thermals that minimizes the energy lost while maximizing the number of interest points visited. Please note that interest points and thermals have different kinds of utility for us. The objective of the group of gliders is to visit as many interest points as possible, while the thermals can be used by gliders to increase the energy of the glider and help it visit even more interest points.

A problem very similar to ours, called Dynamic Traveling Salesman Problem (DTSP), is addressed in [10], [11]. Here the robots are considered to be Dubins vehicles. Dubins vehicles can only travel on paths where the curvature of the path is constrained. In [12] Dubins was able to characterize the optimal paths for these kinds of vehicles. These paths are appropriately called Dubins paths. Dubins paths have the useful property that for a given start and end configuration (position and orientation) of the vehicle, the best path is already known. But Dubin's vehicles have an undesirable characteristic. The curvature of a Dubin's path, although bounded, can be discontinuous. A glider can not follow a path which has a discontinuous curvature because it would mean an instantaneous change in the bank angle of the aircraft which is impossible. Hence, we cannot use this approach and our problem requires a little more work. The approach to solving our problem is outlined below.

The task of planning paths for the gliders is decoupled into two parts, (i) planning sequences of waypoint visitation for each glider and, (ii) planning paths over this sequence. Chapter 2 of the thesis deals with how to assign waypoints to each robot and to calculate the best sequence over the assigned waypoints. This is called *Visitation Sequence Planning*

(VSP). Section 3 tackles the problem of planning paths over these sequences of points which preserve energy. This is called *Energy-preserving Path Planning* (EPP).

1.1 Visitation Sequence Planning

Visitation Sequence Planning is the problem of planning a sequence of waypoint visitation for each robot. This sequence should minimize the cumulative energy lost by the gliders. The solution to VSP involves evaluating the cost of a sequence, which means the energy lost by a glider while traveling over the sequence. This cost is compared with the costs of other sequences to pick the best sequence. It is made sure that not every sequence of waypoint visitation is evaluated since that would be prohibitively computationally expensive.

But there is a problem in this decoupled approach. The cost of a sequence can only be determined if the exact path of the glider over the sequence is known, since the loss of energy of glider depends on the path it takes. Because we have a decoupled solution, the VSP part does not know what exact path is going to be planned for a sequence. Hence the exact cost of each sequence is unknown. To tackle this problem an estimate of the actual cost is used instead.

A problem similar to our task of assigning and sequencing waypoints (for a group of robots) has been investigated in Operations Research. It is called the Euclidean Team Orienteering Problem (ETOP). The ETOP is a Multiple Euclidean Travelling Salesmen Problem with the constraints of a Knapsack Problem. Meaning that in ETOP, a team of agents have to find a path, from start positions to goal, while visiting a given set of waypoints, and also respecting the energy constraint of each robot. Our problem called the Visitation Sequence Planning (VSP) is a variation of the ETOP. Several solutions have been proposed for ETOP including techniques like Column Generation [13], Branch-and-Price scheme [14], Tabu Search [15], [16], Variable Neighborhood Search [17] etc.

The solution presented in [18] guarantees a solution to the ETOP. It employs a multi-tier graph search based technique to find the optimal path for each agent which visits a maximum number of waypoints. But this certainty comes at a price of increased time complexity. The authors justify this by pointing out that the algorithm can be run on-the-fly to provide

intermediate solutions while computing and converging to the optimal one.

The first part of the thesis builds on the work by [18] to solve the ETOP and adapts it to help solve the VSP. There are some fundamental differences between VSP and ETOP that need to be addressed before a solution can be found. In ETOP all waypoints are equally valuable. In VSP interest points and thermals have different utility for the gliders. Furthermore, ETOP only allows for a problem where the energy of the robots can only decrease, whereas, we need a solution that caters for increasing and decreasing energy of the gliders, since they lose energy while traveling and gain energy when visiting thermals.

Chapter 2 deals with solving the VSP. The proposed approach solves these problems by augmenting the afore-mentioned multi-tier graph search based approach. Furthermore, a major improvement to the graph search algorithm is proposed, which is inspired by the LPA* search algorithm. LPA* is a sound and complete graph search algorithm, which is guaranteed to have better time complexity than A* search.

Lastly theoretical guarantees for the feasibility and optimality of the solution obtained from the algorithms are presented in form of theorems along with their proofs. These algorithms are then tested by different scenarios, for different number of robots, interest points and thermals and for different waypoint placements. Simulation results are presented to illustrate the performance of the algorithm and validate the theoretical findings.

1.2 Energy-preserving Path Planning

Once the best sequence of waypoint visitation for each glider has been determined, the path the glider needs to take (over the waypoints) has to be computed. Chapter 3 deals with the problem of planning feasible paths over these waypoint sequences. Since the gliders start with a finite energy, the paths need to be planned such that the energy lost while traversing the paths is optimized, hence the term *energy-preserving*. Furthermore, gliders can only fly on paths that respect the dynamic constraints of the gliders like, maximum and minimum speed, maximum acceleration, maximum turning rate etc.

This chapter starts with studying the flight dynamics of a glider, to come up with a trajectory which minimizes the energy lost by the glider. Since, the optimization problem is

nonlinear and non-convex, the solutions found are sub-optimal.

Trajectory generation as a problem has been widely studied. References [19], [20] proposed optimal trajectory planning by using multiple shooting methods. Pseudospectral optimal control methods were proposed by [21], [22] and nonlinear trajectory generation methods were proposed by [23], [24]. Randomized trajectory generation methods have recently gained momentum with examples such as Rapidly-exploring Random Trees (RRT) [25], Probabilistic Roadmaps [26] and direct method for rapid trajectory prototyping [27].

The method proposed by [28] uses Pythagorean Hodograph Beziér curves to plan optimal paths. The solution consists of a Pythagorean Hodograph Beziér curve over 2-dimensional space and a corresponding timing law which defines how the robot moves over the curves.

The hodograph of a curve is the locus defined by the first parametric derivative of the curve. Pythagorean Hodograph (PH) curves are particularly interesting because they have the nice property that the parametric speed at any given point is an analytic function of the parameter. The arc length of the PH curve can be obtained in closed form.

Beziér curves give us some other nice properties like expressions for parametric speed, acceleration, direction cosines, spatial separation etc can be obtained analytically. Using these two together the speed and acceleration, turning rates and temporal separation between two gliders, can also be expressed as analytical expressions. These expressions can then be used to check the feasibility of the paths.

Chapter 3 formulates the problem of energy-preserving path planning and uses the techniques used in [28] to solve it. The PH Beziér curve discussed earlier is formulated using control points in 2-dimensional space. This curve is defined by a dimension-less parameter ζ . The timing law converts the parametrized curve (defined by the parameter) to a curve w.r.t. time t. Expressions for various parameters like turning rate are calculated to check whether the constraints are being met. The control points of the PH Beziér curves can be used to optimize the cost function (which is the energy lost), while also respecting the equality and inequality constraints. We use the in-built MATLAB function fmincon which is part of the Optimization Toolbox.

These algorithms are then tested for different scenarios like, different number of robots and waypoints and for different waypoint placements. Simulation results are presented to

illustrate the performance of the algorithm and validate the theoretical findings.

CHAPTER 2

VISITATION SEQUENCE PLANNING

Recalling the problem at hand, we are given a number of robots, each with start and end configurations (locations and orientations) and a number of *interest points* to be visited. The robots each have an initial energy, which they use up as they travel through space. There are certain phenomena available in space, called *thermals*, which can be used by the robots to gain some of the lost energy. Hence visiting a thermal will increase a robot's energy and enable it to travel farther and even visit more interest points. Paths have to be planned for each robot such that the energy lost is minimized, while as many interest points are visited, as possible. These paths have to satisfy constraints like the robot should never run out of energy, should always remain inside their dynamic envelope and other mission specific constraints.

The way the path planning problem for multiple robots is solved is by partitioning the problem into two sub problems; waypoint visitation planning and energy-preserving path planning. The reason for decoupling sequence visitation planning and path planning is to be able to come up with a solution in a time-efficient manner. The problem addressed in this section is of waypoint assignment planning, which means planning the best sequence of waypoint visitation for each glider.

At the time of waypoint assignment the exact path that is going to be planned for a given sequence is not known. This means that the actual cost of the sequence (the energy lost) is not known. To tackle this problem an estimate of the actual cost is used. Chapter 3 proves that minimizing the energy lost by the glider corresponds to minimizing the arc length of the path of the glider. We will use the terms energy lost by a glider and distance traveled by a glider interchangeably in Chapter 2. Typically the interest points and thermals are spread out over a large area and as the distances between them become larger, the paths the gliders

take (going from waypoint to waypoint) can be approximated as straight lines. Hence, the estimated cost of a sequence is the sum of straight line distances of traveling from one point to another in the sequence.

Now we move on to a review of the relevant literature related to our problem.

Note: In this chapter the terms robot and glider are used interchangeably.

2.1 Literature Review

A similar problem to our task of assigning and sequencing waypoints has been investigated in Operations Research. It is called the Euclidean Team Orienteering Problem (ETOP) which is a variant of Team Orienteering Problem. In the ETOP, a team of agents has to find an optimal path from start positions to goal, while visiting a given set of waypoints and respecting the time constraint of each robot. The cost to go from one point to another is the Euclidean distance between the two points. The ETOP is NP-hard, hence our problem which can be considered a special case of ETOP is also NP-hard.

Although our specific problem has not been studied in the literature, several solutions have been proposed for solving the ETOP. In 1999 an exact approach to solve the ETOP was proposed in [13] using column generation. Another exact algorithm was proposed in [14], which formulated a branch-and-price scheme by using column generation coupled with Branch-and-Bound technique. A Multi-Tier Graph Search based method is presented in [18], which is provably optimal. This method is discussed in detail in the following section. There has been a lot of work in heuristic methods to solve the ETOP. The heuristic approach was pioneered by [29], proposing a five-step heuristic, by augmenting, the five-step heuristic used to solve the Orienteering Problem. Similarly [15] used a Tabu Search Heuristic embedded in an Adaptive Memory Procedure (AMP) to address the problem. The AMP is similar to genetic algorithms in design. Solutions generated using AMP are stored and further refined using Tabu Search.

A Slow and a Fast Variable Neighborhood Search method was developed in [16]. An Ant Colony Optimization (ACO) approach to solve the ETOP problem was proposed in [30]. In ACO, a feasible solution is created and then updated via a local search. The authors of [31]

focus on getting good ETOP solutions in a small amount of time. The approach involves Guided Local Search and a Skewed Variable Neighborhood Search. In [32], the ETOP is solved using two variants of Greedy Randomized Adaptive Search Procedure; one is a slow version, which gets solutions closer to the optimal, whereas the fast version gives results fast.

While ETOP is similar to our problem in several ways like having multiple robots and multiple waypoints, there are certain noteworthy differences too. Firstly, in our problem we have two types of waypoints; interest points and thermals. For us the utility of visiting both these waypoints is distinct. Visiting interest points is the objective, so it has direct utility. Visiting thermals increases the total energy of the system possibly enabling it to reach new interest points, which were unreachable beforehand. But traveling to and from the thermals, also uses up energy so there is a trade-off. The solution presented in the following sections endeavors to answer this question.

2.2 Introduction to Graph Search

A graph is a mathematical representation of a search problem. Any problem that has a discretized state space and a finite number of actions (or inputs) can be realized as a graph. A graph is a collection of *nodes*, connected by links called *edges*. Each node represents a state in the discretized state space. Each edge represents an action being performed on the node. Whenever an action is performed on a node, a new node is created, also called a *successor* or *child* node. Conversely, the old node can also be called *predecessor* or the *parent* of the new node. Because there are a finite number of actions, a node has a finite number of successors. When all possible successors of a node have been created, the node is said to be *expanded*. Each action performed at any node incurs a cost, which is called an *edge cost*.

The first node in a graph is called *root* node. This can be thought of as the starting state of a system. The node that satisfies a terminal condition, is called a *goal* node. This can be thought of as a goal state of the system. Each node in the graph has a *value* associated with it. This is usually the summation of a sequence of edge costs starting from the root node to that particular node. When there are multiple sequences from the root node to that

particular node, the sequence with the smallest cumulative cost is considered. The objective of graph search is to find the goal node which has the least value.

There are several graph search techniques that can get us the correct solution (meaning the goal node with the least value over all the goal nodes). We will primarily deal with two types of graph search methods, (i) Uniform Cost Graph Search and (ii) A* Graph Search.

Uniform cost graph search works in the following way. It maintains an ordered set of unexpanded nodes, called the *OPEN* set. It starts with just the root node in the OPEN set. It takes the node from the OPEN set, which has the least value and expands it. This node is then discarded from the OPEN set but its successors are put back in the OPEN set in order. Uniform cost search terminates, when a goal node is expanded. This goal node is guaranteed to be the correct solution, if the edge costs are guaranteed to be positive [33].

A* search is similar to the uniform cost search in many ways. It too maintains an ordered set of unexpanded nodes, called the OPEN set. It starts with just the root node in the OPEN set. The order of expansion of nodes is different from the uniform cost search. Here the order of expansion is determined by an *augmented value*, which is the sum of value of the node and a *heuristic*. The heuristic is an estimate of the cost-to-go to the goal node. A* search terminates, when a goal node is expanded. This goal node is guaranteed to be the correct solution, if the edge costs can only be positive and the heuristic is admissible. For A* search to be optimal (meaning it finds the best solution), the heuristic has to be admissible or optimistic [33].

2.3 Multi-Tier Graph Search

This section outlines the work done in [18], which proposes a Multi-Tier Graph Search (MTGS) based solution to solve the ETOP. MTGS involves an iterative graph search over three layers of graphs. This approach will be modified for use in solving the VSP. The reasons for using this approach above others is that i) it is sound and complete, and ii) due to its iterative nature it lends itself for easy adaptation to solving the VSP. The graph layers involved in the search are; Top Level Waypoint Assignment Graph G_T, Middle Level Single Robot Opportunistic Graph G_{M_i} and Low Level Cost-to-Go Graph G_L. A pictorial

depiction of the graphs is given in Figure 2.1 followed by detailed explanation.

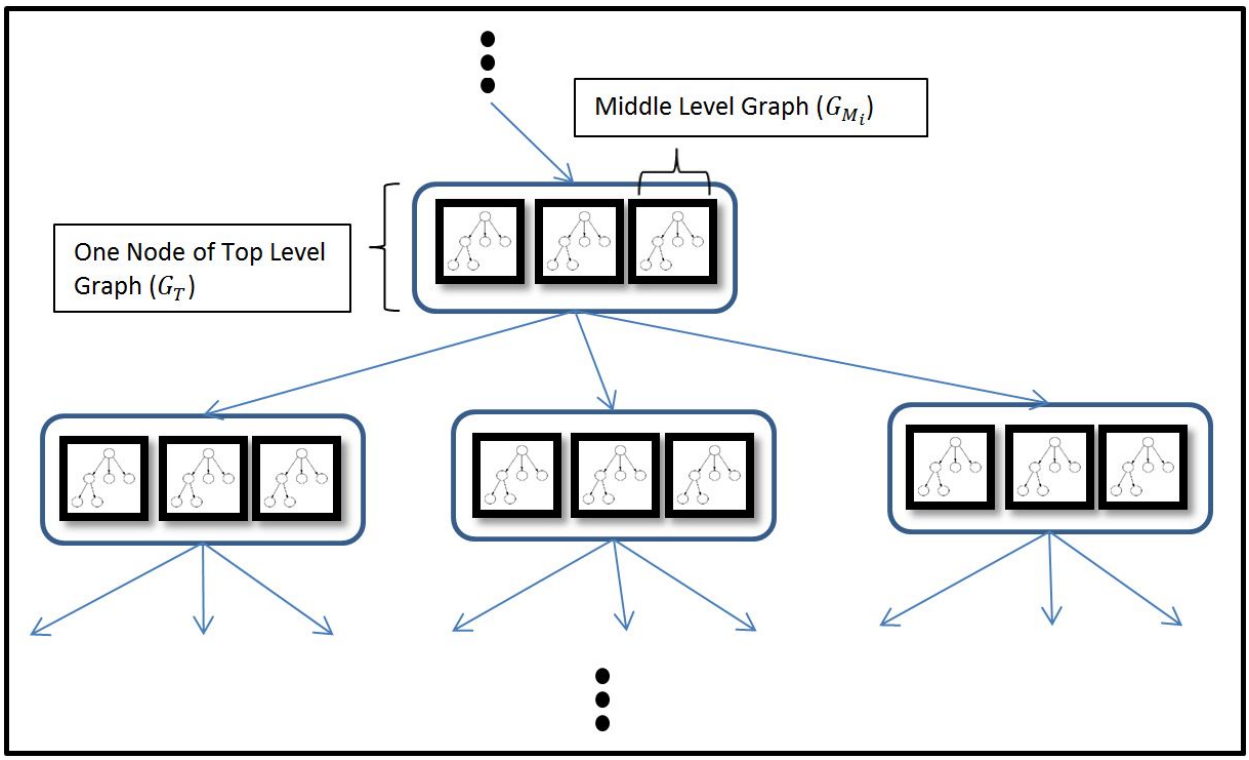

Figure 2.1: Pictorial Depiction of Multi-Tier Graphs

Top Level Waypoint Assignment Graph G_T is the graph, in which waypoints, interest points and thermals are assigned to different robots. Whenever one node of the Top Level Graph is grown, one whole corresponding Middle Level Single Robot Opportunistic Graph G_{M_i} is constructed. Hence, there is a one-to-one correspondence between edges in G_T and middle level graphs G_{M_i}. The edge cost for each edge in the G_T is obtained from the corresponding G_{M_i}. Similarly, for each edge in the Middle Level graph a whole new corresponding Low Level Cost-to-go Graph G_L is constructed, and likewise the edge costs in the middle level graph are obtained from the corresponding low level graph.

2.3.1 Top Layer Waypoint Assignment Graph G_T

The search over Top Layer Graph determines the best waypoint assignment for each robot. The MTGS involves one graph search procedure through this graph. Each node in G_T is a vector of M elements (where M is the number of waypoints); it represents which waypoint

has been assigned to which robot. Each element in the node corresponds to a waypoint and represents the robot, to which the waypoint has been assigned to. Lets take the node $q = \{q_i\}$, if $q_j = l$. This means that the jth waypoint has been assigned to the lth robot. If l is zero, it means the waypoint has not been assigned to any robot.

A node can be a successor of another node, only if one waypoint which was unassigned in the parent node, is assigned in the child node, and all the other waypoint assignments in the parent node are transferred to the child node. A middle level graph G_{M_i} is constructed corresponding to the new edge, which joins the parent and child nodes. The edge cost of this edge is obtained from G_{M_i}. G_{M_i} also provides the number of waypoints not visited by that robot. The value of a node in G_T is the summation of all the edge costs starting from the root node and ending at that particular node. If there are multiple routes from the root node, then the one with the smallest accumulated edge cost is taken. The value of a node q represents the energy spent by all the robots, to travel over the waypoints as assigned in q. If the child node is also a goal node, then there is an extra penalty cost in addition to the edge cost. It is called P_T and the value is:

$$P_T = \sum_{i=1}^{N} T_{max_i} + 1.$$

T_{max_i} is the maximum time the robot i can travel. This penalty term makes the goal nodes with more unvisited waypoints undesirable. The root of the graph is q_{start}, which is a vector of 0s, since no waypoint has been assigned to any robot in the beginning. Similarly q_{goal} is the goal node, which corresponds to all the waypoints being assigned. Notice that there can be many different combinations of waypoint assignment to robots; hence there are many vectors q_{goal}. Consequently the aim now is to find the q_{goal}, which has the minimum value.

The algorithm for computing the best q_{goal}, which is a search through G_T, is presented below.

```
1: procedure MultiRobotInterleave()
2: OPEN = {q_start}
3: g(q_start) = 0
4: g(q) = ∞, ∀q ∈ V(G_T) and q ≠ q_start
5: while q_goal not expanded do
6:    remove q with smallest f(q) from OPEN
7:    for each successor q' of q do
8:       c(q,q') = SingleRobotInterleave(waypoints for
         robot i according to state q') - c*_q(n_Si, n_Gi)
9:       if q' == q_goal then
10:         c(q,q') += (# waypoints not visited across all
            agents in q') × P_T
11:      if g(q') > g(q) + c(q,q') then
12:         g(q') = g(q) + c(q,q')
13:         insert q' in OPEN with f(q') = g(q');
```

The search through the graph is similar to uniform cost search, except for the additional penalty term added to the cost, when the child node is also a goal. The solution to the graph search is the first goal node to be expanded from among all the goal nodes. The value of the goal node is

$$Value = \sum_{i=1}^{N} c(\pi_i) + k' P_T,$$

where $c(\pi_i)$ is the cost of the path for the ith robot obtained from G_{M_i}, k' is the total number of unvisited waypoints, and P_T is the terminal penalty term.

2.3.2 Middle Level Single Robot Opportunistic Graph

A search over Middle Level Graph determines the best possible route through the assigned waypoints for a particular robot. This Middle Level Graph is constructed every time a node is expanded in the top level graph. Lets say a waypoint has been assigned to the robot l in the top level graph. Top level graph provides a set of waypoints assigned to l. Observe that all of these assigned waypoints might not be visited by l, since it might not be feasible for l (meaning it might not be possible for l to visit all the waypoints). The edge between a parent and its child has a cost; this edge cost is obtained from the low level graph G_L.

Whenever each node in G_{M_i} is expanded into a successor, a waypoint is appended to the end of the sequence. For each successor node a low-level graph is constructed. The cost of

transition from parent to child is obtained from the low level graph. Every node has a value, which is the summation of all the edge costs starting form the root node of the graph and ending at the particular node. The value of a node n represents the energy spent by l to travel over the assigned waypoints. If the child node happens to be a goal node too, then there is an extra penalty cost in addition to the edge cost. It is called P_M and the value is

$$P_M = T_{max_i} + 1.$$

T_{max_i} is the maximum time the robot i can travel. Each node n in the G_{M_l} is a sequence, representing which of the assigned waypoints have been visited and in what order. The root of the graph is n_{start}, which corresponds to the situation when no waypoint has been visited. Similarly, n_{goal} is the goal node, achieved when the last waypoint in the sequence is the goal point of l. Notice that there can be many different combinations of waypoint visitation ending in the goal; hence there are many n_{goal} vectors. Consequently, the objective now is to find the n_{goal} which has the smallest value. Note that a path containing just the start and goal points (going from start to goal) might be the shortest path, but will have the highest penalty and hence the highest value.

The algorithm for computing the best q_{goal} is presented below

1: **procedure SingleRobotInterleave(waypoints)**
2: $OPEN = \{n_{start}\}$
3: $g(n_{start}) = 0$
4: $g(n) = \infty$, $\forall n \in V(G_M)$ and $n \neq n_{start}$
5: **while** n_{goal} not expanded **do**
6: remove n with smallest $f(n)$ from $OPEN$
7: **for** each successor n' of n **do**
8: $c(n,n') = d^*(n,n')$
9: **if** dubin's path goes through obstacle **then**
10: $c(n,n') = c^*(s_n, s_{n'})$ //from low-level search
11: $C(n,n') = c(n,n')$
12: **if** $n' == n_{goal}$ **then**
13: $C(n,n')$ += (# waypoints not visited in n) $\times$ P_M
14: **if** $g(n)+c(n,n') < T_{MAX}$ and $g(n') > g(n)+C(n,n')$ **then**
15: $g(n') = g(n)+C(n,n')$
16: insert n' in $OPEN$ with $f(n') = g(n')$;
17: **return** $g(n_{goal})$ - (# waypoints not visited) $\times$ P_M

The search through the graph is similar to uniform cost search, except for the additional penalty term added to the cost when the child node is also a goal. At each expansion, if the cost of the path exceeds the T_{max_i}, it is considered invalid and deleted from the openset. This means that the path is not feasible for travel. The solution (or the best sequence of waypoint visitation) is the first goal node to be expanded from among all the goal nodes. The cost of this sequence is,

$$Cost = \sum_{i=1}^{\hat{k}+1} c(n_{i-1}, c_i) + (k - \hat{k})P_M.$$

Here $c(n_{i-1}, n_i)$ is the cost to go from one waypoint to the other, k is the number of assigned waypoints, $\hat{k}$ is the number of visited waypoints, and P_M is the terminal penalty term.

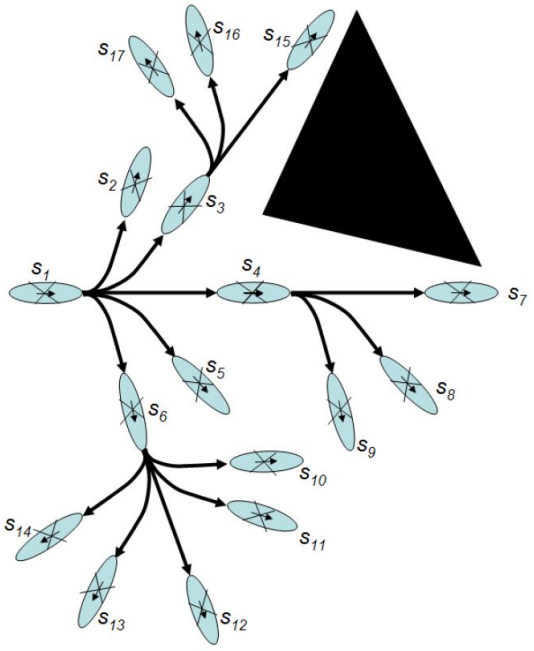

Figure 2.2: Low Level Cost-to-Go Graph

2.3.3 Low Level Cost-to-Go Graph

Given a start and an end waypoint, the Low Level Graph determines the best path to go from one waypoint to the other. This Low Level Level Graph is constructed every time a

node is expanded in the Middle Level graph, which also provides the above mentioned start and the end waypoints.

The graph is either a Dubins path from start to goal [12] or is an A* search over discretized action space. For the A* search the action space is discretized in such a way that under the application of actions the state space is a finite set. This is represented in Figure 2.2.

The A* search is considered, if there is an obstacle in the Dubins path. This is depicted in Fig. 2.2. The cost of the graph is the max over the two costs, one obtained from the Dubins Path and the other from the A* Search.

2.3.4 Theorems for MTGS

Theorem 1: *Given a set of motion primitives, a path found through the graph G_M visits as many waypoints as possible within the alloted time T_{MAX} while minimizing cost of the path.*

Let $\bar{G}_M \subset G_M$ be the mid level graph that contains only the states through which the goal can be reached within the time bound T_{MAX}. Let $\hat{\pi}_{\bar{G}_M}$ be the solution obtained by the planner with the cost $\sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i)$ where $\hat{k}$ is the number of waypoints covered in this solution.

We define,
$c(\hat{\pi}_{\bar{G}_M}) = \sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i) + (M - \hat{k}) \times P_M$, where $n \in V(G_M)$, $n_0 = n_{start}, n_{\hat{k}+1} = n_{goal}$, M = Total number of waypoints and $P_M = T_{MAX} + 1$.

Lemma 1: *All edges $\in E(\bar{G}_M)$ are optimal w.r.t the discretization of state space and action space since they are obtained from the low-level optimal graph search except for the edges connecting into goal states which have costs equal to the cost of least-cost path plus the penalty.*

Lemma 2: *As we run an optimal $A^\star$ search on the pruned graph $\bar{G}_M$, $\hat{\pi}_{\bar{G}_M}$ is the least cost path.*

We need to Prove:

1) $\hat{k}$ is the maximum number of waypoints that can be covered.
2) $\forall \pi_{\bar{G}_M}$ with fixed number of waypoints $\hat{k}$, $\sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i)$ is the minimum cost.

The reason for choosing this algorithm was that it is sound and complete, meaning it is guaranteed to provide the best solution. Theorems 1 and 2, taken from [18], show that the algorithm is provably optimal. The Proofs are given in the Appendix.

Theorem 1 states that, given that the Low Level Graph Search provides the optimal solution, the solution obtained through Middle Level Graph Search will have a) maximum number of waypoints visited b) in the most efficient way possible (with the least possible energy loss).

Theorem 2: *The path found through G_T has the minimum total number of unvisited waypoints and total path costs across all robots.*

Let $\bar{G}_T \subset G_T$ be the top level graph that contains only the states through which the goal can be reached within the time bound T_{MAX_i} for each robot. Let $\hat{\pi}_{\bar{G}_T}$ be the solution obtained by the planner and $\hat{k}$ be the number of unvisited waypoints in this solution.

We define,
$c(\hat{\pi}_{\bar{G}_T}) = \sum_{i=1}^{N} c(\pi_i) + \hat{k} \times P_T$, where $P_T = \sum_{i=1}^{N} T_{MAX_i} + 1$.

Lemma 1: *All edges $\in E(\bar{G}_T)$ are optimal since they are obtained from the mid-level optimal graph search except the edges connecting into goal states which have costs equal to the cost of least-cost path plus the penalty. This is proved in Theorem 1.*

Lemma 2: *As we run an optimal $A^\star$ search on the pruned graph $\bar{G}_T$, $\hat{\pi}_{\bar{G}_T}$ is the least cost path*

We need to Prove:

1) $\hat{k}$ is the minimum number of unvisited waypoints.
2) $\forall \pi_{\bar{G}_T}$ with fixed number of unvisited waypoints $\hat{k}$, $\sum_{i=1}^{N} c(\bar{\pi}_i)$ is the minimum cost, subject to $c(\bar{\pi}_i) \leq T_{MAX_i} \forall \pi_i$.

Similarly. Theorem 2 states that, given that the Middle Level Graph Search provides the optimal solution, the solution obtained through Top Level Graph Search will have a) maximum number of waypoints visited over all the gliders and, b) the least possible energy loss over all the gliders.

2.4 Bi Level Graph Search (BLGS) for solving the VSP

Bi-Level Graph Search (BLGS) is the modified version of MTGS, to solve the Visitation Sequence Planning. The MTGS approach to solving ETOP, lends itself particularly well to solving our problem of VSP. But there are some differences between the two problems. We start with formulating the problem in a mathematically rigorous way. Then the differences between ETOP and VSP are discussed.

2.4.1 Problem Formulation

Firstly, we define the notation used in solving the VSP. Consider a set of gliders $\mathcal{I}_v = \{1, 2, ..., n_v\}$, with starting positions $\mathcal{B}_x = \{x_i \in \mathbb{R}^2 | i \in \mathcal{I}_v\}$, starting orientations $\mathcal{B}_\theta = \{\theta_i \in [-\pi, \pi] | i \in \mathcal{I}_v\}$, goal positions $\mathcal{B}_g = \{g_i \in \mathbb{R}^2 | i \in \mathcal{I}_v\}$ and starting energies $\mathcal{B}_\epsilon = \{\epsilon_i \in \mathbb{R} | i \in \mathcal{I}_v\}$. The quantity *starting energy* ϵ_i, used in this thesis, is the distance a glider can travel before hitting the ground. The set of interest points $\mathcal{I}_{ip} = \{1, 2, ..., n_{ip}\}$ are located at positions $\mathcal{B}_{ip} = \{p_i \in \mathbb{R}^2 | i \in \mathcal{I}_{ip}\}$. All interest points have equal importance for the gliders. The set of thermals $\mathcal{I}_t = \{1, 2, ..., n_t\}$ are located at positions $\mathcal{B}_\gamma = \{\gamma_i \in \mathbb{R}^2 | i \in \mathcal{I}_t\}$ with utilizable energy $\mathcal{B}_\Delta = \{\Delta_i \in \mathbb{R} | i \in \mathcal{I}_t\}$. Here again the utilizable energy of the thermal Δ_i, is the additional distance a glider can travel when it acquires the thermal. The set of waypoints is $\mathcal{I}_{wp} = \{1, ..., n_{ip}, n_{ip} + 1, ..., n_{wp}\}$, where $n_{wp} = n_{ip} + n_t$. The corresponding positions of the waypoints are $\mathcal{B}_{wp} = \{w_i | i \in \mathcal{I}_{wp}, \textit{If } i \leq n_{ip}, w_i = p_i \textit{ otherwise } w_i = \gamma_{i-n_{ip}}\}$. Note that interest points and thermals have different kinds of importance. Interest points need to be visited by the gliders, and thermals enable the gliders to reach more interest points.

We assume that the starting energy ϵ_i, of any glider should be larger than the Euclidean distance between *any* two waypoints. This is required for the VSP problem to be solvable, since without it the glider is unable to reach any waypoint. We later use this assumption to formulate a heuristic in Section 2.4.3.

Our robot starts with a starting energy called $\epsilon_i, i \in \mathcal{I}_v$ and gradually loses energy as it travels through space. The first difference between ETOP and VSP is that now we have two

kinds of waypoints, interest points and thermals. The objective is to maximize the collection or visitation of interest points, while making sure that none of the robots run out of energy. We also have another waypoint type called the thermal, which is like a refueling point for the robot. The robot can gain energy, as it visits the thermal waypoints by extending its range. Consequently the thermals have a different type of indirect utility for us. They can allow the gliders to visit far-off waypoints, which were impossible to visit beforehand. The energy a thermal imparts to a robot $\Delta_i, i \in \mathcal{I}_t$ and initial energy of the robot ϵ_i are considered known.

The second difference between the two problems is that now the energy of the robot is changing. It gains energy as it visits thermals and loses it as it travels.

2.4.2 Approach to Solution

To cater for these differences the penalty costs discussed in Sections 2.3.1 and 2.3.2 are modified, and a structure is put into place which removes clearly infeasible solutions. The reasons for using this approach above others is that it is sound and complete. This new approach is called the Bi-Level Graph Search (BLGS).

BLGS has a two-layered graph structure. The graph layers involved in the search are upper level graph Γ_U and lower level graph Γ_L. A search over Γ_L computes the best sequence of waypoint visitation for one robot, given the waypoints assigned to that robot. A search over Γ_U finds the best possible assignment of waypoints (interest points and thermals) for each robot in a group of robots. Each node in Γ_U represents a particular assignment of waypoints to robots. The objective of BLGS is to find the node in Γ_U, where all the waypoints have been assigned to the robots, the accumulated cost (the energy lost by the gliders while traveling over the waypoints) is minimized and the number of unvisited interest points is also minimized. This accumulated cost is calculated using Γ_L. This is discussed in detail in the following sections.

It should be noted here that a waypoint *assigned* to a robot in Γ_U does not ensure that the waypoint would be *visited* by that robot in Γ_L. It is possible that when search over Γ_U assigns a waypoint to a robot, the search over Γ_L might decide that visiting the waypoint is

not feasible, meaning the waypoint isn't reachable by the robot.

2.4.3 Lower Level Graph Γ_L

The lower level graph determines, the best possible route through the *assigned* waypoints, for one particular robot l. This lower level graph is constructed every time a new node is made in the upper level graph. Upper level graph provides a set of assigned waypoints to be visited by that robot, Ξ. Ξ is defined in section 2.4.4. Ξ_{ip} is the set of interest points assigned to the robot and Ξ_t is the set of thermals assigned to the robot l. Observe that all of these waypoints might not be visited by the robot, since visiting some of the waypoints might be unprofitable or infeasible.

Each node $\boldsymbol{\lambda}$ in Γ_L is a sequence $\boldsymbol{\lambda} = \{\lambda_i | \lambda_i \in \Xi \ \cup \{g_l\}\}$ of waypoints that are visited after the starting position of the robot $x_l \in \mathcal{B}_x$. The expression given above signifies that the robot can visit either one of the assigned waypoints or $g_l \in \mathcal{B}_g$, which is the goal location for robot l. The root of the graph $\boldsymbol{\lambda}_{start} = \{\}$ is an empty sequence, meaning no assigned waypoint has been visited. When the last waypoint in a node is the goal position $g_l \in \mathcal{B}_g$, the node is called the goal node. Notice that there can be many different waypoint sequences ending with g_l. Hence, $\boldsymbol{\Lambda}_{goal}$ is the set of all goal nodes. Consequently the aim now is to find the best goal node $\boldsymbol{\lambda}^*$, which is not infeasible, minimizes the number of unvisited interest points and plans a sequence with the least expended energy for robot l.

Whenever each node in Γ_L is expanded into a successor, a waypoint is appended to the end of the sequence. Consider two nodes in Γ_L, $\boldsymbol{\lambda'}$ and $\boldsymbol{\lambda}$. $\boldsymbol{\lambda}$ has j elements and $\boldsymbol{\lambda'}$ has $j+1$ elements

$$\boldsymbol{\lambda'} \in Succ(\boldsymbol{\lambda}) \text{ only if}, \tag{2.4.1}$$

$$\lambda'_i = \begin{cases} \lambda_i & \text{if } i \leq j \\ \mu \in \{\Xi \setminus \boldsymbol{\lambda}\} & \text{if } i = j+1\,, \end{cases} \tag{2.4.2}$$

where the set $Succ(\boldsymbol{\lambda})$ is the set of successors of $\boldsymbol{\lambda}$.

Each node $\boldsymbol{\lambda}$ has a value associated with it called the transition cost. It can be described as the estimate of energy spent by the glider to travel over the sequence $\boldsymbol{\lambda}$. The transition

cost of a node $\boldsymbol{\lambda}$ with j elements, is defined as

$$T_L(\boldsymbol{\lambda}) = \begin{cases} \sum_{i=1}^{Card(\boldsymbol{\lambda})-1} ||w_{\lambda_{i+1}} - w_{\lambda_i}||_2 + ||w_{\lambda_1} - x_l||_2, \text{ if } \lambda_j \neq g_l\,, \\ \sum_{i=1}^{Card(\boldsymbol{\lambda})-2} ||w_{\lambda_{i+1}} - w_{\lambda_i}||_2 + ||w_{\lambda_1} - x_l||_2 + ||g_l - w_{\lambda_{j-1}}||_2, \text{ if } \lambda_j = g_l\,, \end{cases} \tag{2.4.3}$$

where $w_{\lambda_i} \in \mathcal{B}_{wp}$ are positions for the waypoints in $\boldsymbol{\lambda}$, $Card(.)$ is the cardinality operator, $T_L(\boldsymbol{\lambda})$ is an estimate of the energy required by the glider to travel over the sequence $\boldsymbol{\lambda}$.

Note that we use an estimate of the energy spent by the glider and not the actual energy spent by the glider. As discussed in the beginning of Chapter 2, we do not know the actual energy that the glider is going to spend while traveling over the sequence. But we do know that if the waypoints get farther apart, the energy spent by the glider approaches the Euclidean distance between the waypoints. Another thing that is true is that the Euclidean distance between two waypoints is also the lower limit of the energy spent by a glider, while traveling over the waypoints. This knowledge can be later used to root out the sequences which are definitely infeasible.

Another value associated with a node $\boldsymbol{\lambda}$ is $V_L(\boldsymbol{\lambda})$. It is the total cost of the node. A node is selected for expansion in graph search, according to this cost. It is defined as

$$V_L(\boldsymbol{\lambda}) = \begin{cases} T_L(\boldsymbol{\lambda}) & \text{if } \boldsymbol{\lambda} \notin \boldsymbol{\Lambda}_{goal} \\ T_L(\boldsymbol{\lambda}) + k' P_L & \text{if } \boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{goal}\,, \end{cases} \tag{2.4.4}$$

$$\text{where } P_L = \epsilon_l + \sum_{i \in \Xi_t} \Delta_i + 1\,, \tag{2.4.5}$$

where k' is the number of unvisited interest points in node $\boldsymbol{\lambda}$. For obtaining the correct solution of the graph search over Γ_L, P_L is required to be greater than the maximum possible energy of the glider, given by $\epsilon_l + \sum_{i \in \Xi_t} \Delta_i$. This expression is the summation of the starting energy of the glider ϵ_l and the sum of the energies of all the thermals $\sum_{i \in \Xi_t} \Delta_i$ that the glider can possibly acquire. Hence, a glider cannot possibly have energy higher than P_L.

The algorithm for computing the goal node $\hat{\boldsymbol{\lambda}}$ is an A* Search over the lower level graph Γ_L. The pseudo code for the search is presented in Algorithm 1. The heuristic $\zeta(\boldsymbol{\lambda})$ for the

A* Search, which is an estimate for the cost to go from $\boldsymbol{\lambda}$ to $\hat{\boldsymbol{\lambda}}$, is defined below:

$$\zeta(\boldsymbol{\lambda}) = \begin{cases} E_{min}(Card(\Xi_{ip} \setminus \lambda) + 1) & \text{if } \boldsymbol{\lambda} \notin \boldsymbol{\Lambda}_{goal} \\ \zeta(\boldsymbol{\lambda}) = 0 & \text{if } \boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{goal}\,, \end{cases} \tag{2.4.6}$$

$$\text{where } E_{min} = \min_{w_i, w_j \in \Xi \cup \{x_l, g_l\}, i \neq j} ||w_i - w_j\,||_2\,.$$

Lemma 2.4.1 shows that $\zeta(\boldsymbol{\lambda})$ will always be admissible.

Lemma 2.4.1 *The heuristic $\zeta(\boldsymbol{\lambda})$ of cost-to-go from $\boldsymbol{\lambda}$ to $\boldsymbol{\lambda}_{goal}$ is an admissible heuristic, meaning it will always be less than the real cost-to-go to any node in the set $\boldsymbol{\Lambda}_{goal}$.*

Proof For $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{goal}$ the heuristic is automatically admissible. Consider the node $\boldsymbol{\lambda} \notin \boldsymbol{\Lambda}_{goal}$. The number of assigned interest points is $Card(\Xi_{ip})$. The number of interest points already visited in the node is $Card(\boldsymbol{\lambda} \cap \Xi_{ip})$. Assume that the robot decides to visit a sequence of waypoints $\boldsymbol{\pi}$ (culminating in g_l) after the sequence $\boldsymbol{\lambda}$, where $0 \leq Card(\boldsymbol{\pi}) \leq Card(\Xi \setminus \boldsymbol{\lambda})$. The cost of this sequence is

$$V_L(\boldsymbol{\pi}) = \sum_{i=1}^{Card(\boldsymbol{\pi})+1} ||w_{\lambda_{i+1}} - w_{\lambda_i}||_2 + Card(\{\Xi_{ip} \setminus \boldsymbol{\lambda}\} \setminus \boldsymbol{\pi}) P_L$$

$$\geq (Card(\boldsymbol{\pi} \cap \Xi_{ip}) + 1) E_{min} + Card(\{\Xi_{ip} \setminus \boldsymbol{\lambda}\} \setminus \boldsymbol{\pi}) P_L\,,$$

$$\geq (Card(\boldsymbol{\pi} \cap \Xi_{ip}) + 1) E_{min} + Card(\{\Xi_{ip} \setminus \boldsymbol{\lambda}\} \setminus \boldsymbol{\pi}) E_{min}\,,$$

$$\geq E_{min}.(Card(\Xi_{ip} \setminus \boldsymbol{\lambda}) + 1)\,,$$

$$= \zeta(\boldsymbol{\lambda})\,.$$

The second step in the proof above, follows from the fact $\epsilon_l < P_L$, and we know from our assumption that $E_{min} < \epsilon_l$ (which is needed for the problem to be solvable), hence $E_{min} < P_L$. Hence it has been proven that the heuristic $\zeta(\boldsymbol{\lambda})$ is an admissible heuristic. □

Algorithm 1 presents the A* search over graph G_L. Line 13 in the algorithm roots out nodes in Γ_L, which are guaranteed to be infeasible. It does this by comparing the lower limit of energy spent by the glider $T_L(\boldsymbol{\lambda'})$ with the sum of starting energy, and the energy gained

Algorithm 1 Lower Level Graph Search

Require: $l, x_l, \epsilon_l, g_l, \Xi$
1: $\Omega = \{\boldsymbol{\lambda}_{start}\}$
2: $V_L(\boldsymbol{\lambda}_{start}) = 0$
3: $V_L(\boldsymbol{\lambda}) = \infty \quad \forall \boldsymbol{\lambda} \in Vertices(\Gamma_L)$ and $\boldsymbol{\lambda} \neq \boldsymbol{\lambda}_{start}$
4: **while** $\boldsymbol{\lambda} \notin \boldsymbol{\Lambda}_{goal}$ **do**
5: remove $\boldsymbol{\lambda}$ from Ω with the smallest $V_L(\boldsymbol{\lambda}) + \zeta(\boldsymbol{\lambda})$
6: **for all** $\boldsymbol{\lambda'} \in Succ(\boldsymbol{\lambda})$ **do**
7: $T_L(\boldsymbol{\lambda'}) =$ as calculated in (2.4.3)
8: **if** $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{goal}$ **then**
9: $V_L(\boldsymbol{\lambda'}) = T_L(\boldsymbol{\lambda'}) + k' P_L$
10: **else**
11: $V_L(\boldsymbol{\lambda'}) = T_L(\boldsymbol{\lambda'})$
12: **end if**
13: **if** $T_L(\boldsymbol{\lambda'}) \leq \epsilon_l + \sum_{i \in (\Xi_t \cap \boldsymbol{\lambda})} \Delta_i$ **then**
14: Insert $\boldsymbol{\lambda'}$ into Ω with cost $V_L(\boldsymbol{\lambda'})$
15: **end if**
16: **end for**
17: **end while**
18: $\hat{\boldsymbol{\lambda}} = \boldsymbol{\lambda}$
19: **return** $\hat{\boldsymbol{\lambda}}, T_L(\hat{\boldsymbol{\lambda}}), \hat{k}$

by visiting the thermals $\epsilon_l + \sum_{i \in (\Xi_t \cap \boldsymbol{\lambda})} \Delta_i$. If the lower limit of energy spent is larger than the sum of the starting energy and the energy gained by visiting the thermals, that node is definitely not feasible. A critical point to note here is that as we are using the lower limit of energy spent, there may be some other possibly infeasible nodes that we cannot root out.

The objective of the algorithm is to find $\boldsymbol{\lambda}^*$, which is the node that makes the robot visit as many interest points as possible, with the least possible transition cost. Since the heuristic $\zeta()$ is admissible, it will return the node $\hat{\boldsymbol{\lambda}} \in \boldsymbol{\Lambda}_{goal}$ such that $V_L(\hat{\boldsymbol{\lambda}}) \leq V_L(\boldsymbol{\lambda}), \forall \boldsymbol{\lambda} \in \boldsymbol{\Lambda}_{goal}$. Lemma 2.4.2 presents the soundness guarantees for the algorithm, meaning $\hat{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^*$.

In the algorithm, l is the robot for which Γ_L is being constructed, Ω denotes the Open set (the set of nodes yet to be expanded), $Vertices(\Gamma_L)$ refers to the set of all possible nodes in Γ_L, $k' = Card(\Xi_{ip} \setminus \boldsymbol{\lambda})$ is the number of unvisited interest points. Once the A* search is done $\hat{\boldsymbol{\lambda}}, T_L(\hat{\boldsymbol{\lambda}}), \hat{k}$ are returned to the upper level graph.

Lemma 2.4.2 *An A* search on the graph Γ_L, as described in Algorithm 1, returns the path $\hat{\boldsymbol{\lambda}}$, with $\hat{k}$ number of unvisited interest points. It is guaranteed that $\hat{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^*$, meaning that*

(a) $\hat{k}$ has the minimum number of unvisited interest points, (b) for all nodes in $\mathbf{\Lambda}_{goal}$ with number of unvisited interest points $= \hat{k}$, $T_L(\hat{\boldsymbol{\lambda}})$ is the least transition cost.

Proof (a) We prove by contradiction. Suppose, $\tilde{\boldsymbol{\lambda}}$, which is a feasible path and also a goal node, has less unvisited interest points than $\hat{\boldsymbol{\lambda}}$, $\tilde{k} < \hat{k}$. Since Algorithm 1 is an A* search, and Lemma 2.4.1 ensures that the heuristic $\zeta(\boldsymbol{\lambda})$ is admissible, $\hat{\boldsymbol{\lambda}}$ will have least total cost

$$V_L(\hat{\boldsymbol{\lambda}}) \leq V_L(\tilde{\boldsymbol{\lambda}}) \,,$$

$$T_L(\hat{\boldsymbol{\lambda}}) + \hat{k} * P_L \leq T_L(\tilde{\boldsymbol{\lambda}}) + \tilde{k} P_L \,,$$

$$(\hat{k} - \tilde{k}) P_L \leq T_L(\tilde{\boldsymbol{\lambda}}) - T_L(\hat{\boldsymbol{\lambda}}) \,,$$

$$(\hat{k} - \tilde{k}) P_L \leq \epsilon_l + \sum_{i \in \Xi_t} \Delta_i \,.$$

The last step was made possible by the fact that the transition cost $T_L(\boldsymbol{\lambda})$ for any feasible node $\boldsymbol{\lambda}$ should be less the maximum possible energy of the glider. Using the definition of P_L and the fact that $\tilde{k} - \hat{k} \geq 1$, we arrive at a contradiction.

(b) Assume another node $\tilde{\boldsymbol{\lambda}} \in \mathbf{\Lambda}_{goal}$, such that it has number of unvisited interest points $\tilde{k}$ and $\tilde{k} = \hat{k}$. Using the logic used in part (a) $\hat{\boldsymbol{\lambda}}$ will have least total cost

$$V_L(\hat{\boldsymbol{\lambda}}) \leq V_L(\tilde{\boldsymbol{\lambda}}) \,,$$

$$T_L(\hat{\boldsymbol{\lambda}}) + \hat{k} P_L \leq T_L(\tilde{\boldsymbol{\lambda}}) + \tilde{k} P_L \,,$$

$$T_L(\hat{\boldsymbol{\lambda}}) \leq T_L(\tilde{\boldsymbol{\lambda}}) \,.$$

Hence $\hat{\boldsymbol{\lambda}}$ has the is the minimum transition cost for all goal nodes with number of unvisited interest points equal to $\hat{k}$. □

This lemma shows that Algorithm 1 finds the goal node with the least transition cost. Since transition cost is a summation of the *estimated* cost, this node may not be the best node if you consider the *real* cost. Future work must include a better estimate of the cost so that more substantial claims can be made.

Graph search over Γ_L returns the sequence $\hat{\boldsymbol{\lambda}}$, with the number of unvisited interest points $\hat{k}$ and the transition cost $T_L(\hat{\boldsymbol{\lambda}})$ to the upper level graph Γ_U.

2.4.4 Upper Level Graph Γ_U

Upper level graph is the graph, in which waypoints (interest points and thermals) are assigned to different robots. A search through this upper level graph determines the best waypoint assignment for all the robots. The correct solution to the search is the waypoint assignment which makes the robots visit as many interest points as possible, with the least possible energy lost over all robots. Notice for VSP an estimate of lost energy is used, as discussed in the beginning of Chapter 2.

In Γ_U, each node $\boldsymbol{\tau} = [\tau_1, ..., \tau_{n_{wp}}]^T$ with $\tau_i \in \mathcal{I}_v$, is a vector of n_{wp} elements (where n_{wp} is the number of waypoints). Each element in the vector represents which robot has been assigned to each waypoint. So, for example if $\tau_j = l$, it means that the jth waypoint has been assigned to lth robot. If $l = 0$, then the waypoint has not been assigned to any robot. The first node in the graph is when no robots have been assigned to any waypoint, hence the root of the graph $\boldsymbol{\tau}_{start}$ is an all 0 vector. The node where all the waypoints have been assigned is called a goal node. There can be many possible goal nodes, since there are many possible combinations of waypoint assignments. The set of goal nodes is called $\boldsymbol{T}_{goal}$. Consequently the aim now is to find the best goal node $\boldsymbol{\tau}^*$, which makes the robots visit as many interest points as possible with the least amount of energy expended.

Whenever one node of the upper level graph Γ_U is grown, it corresponds to one waypoint being assigned to one robot. The set of possible successors of $\boldsymbol{\tau}$ is called $Succ(\boldsymbol{\tau})$. Suppose $\boldsymbol{\tau'} \in Succ(\boldsymbol{\tau})$, where the jth waypoint has been assigned to lth robot. Consider $\boldsymbol{\tau} = [\tau_1, ..., \tau_{n_{wp}}], \boldsymbol{\tau'} = [\tau'_1, ..., \tau'_{n_{wp}}]$. Then the set $Succ()$ is defined as:

$$\boldsymbol{\tau'} \in Succ(\boldsymbol{\tau}) \text{ only if}, \tag{2.4.7}$$

$$\tau'_i = \begin{cases} \tau_i & \text{if } i \neq j \\ l & \text{if } i = j, \tau_j = 0\,. \end{cases}$$

The statement implies that only an unassigned waypoint can be assigned to a new robot and the assignments for all the other waypoints are carried over from the parent node $\boldsymbol{\tau}$ to the child node $\boldsymbol{\tau'}$. When a node $\boldsymbol{\tau'}$ is grown from a parent node $\boldsymbol{\tau}$, a new waypoint j is assigned to l and a new Γ_L is created for the robot l. The set of assigned waypoints Ξ that is to be passed to Γ_L, is defined as $\Xi = \{\xi_i = j | \tau_j = l\}$. This means that Ξ contains all the waypoints assigned to robot l. Γ_L returns the best sequence $\hat{\boldsymbol{\lambda}}$ through the assigned waypoints, the number of unvisited interest points in this sequence $\hat{k}$ and the transition cost for this sequence $T_L(\hat{\boldsymbol{\lambda}})$.

Each node in the Γ_U has a vector of costs $C(\boldsymbol{\tau}) = [c_1, c_2, ..., c_{n_v}]^T, c_i \in \mathbb{R}$ associated with it. Each c_l corresponds to robot l and is obtained from the lower level graph Γ_L, grown for robot l and node $\boldsymbol{\tau}$. Γ_L computes the sequence $\hat{\boldsymbol{\lambda}}$ of visitation over the assigned waypoints as is described in Section 2.4.3. Using the case where the jth waypoint has been assigned to the lth robot and letting $C(\boldsymbol{\tau})$ and $C(\boldsymbol{\tau}')$ be the vectors of costs of parent $\boldsymbol{\tau}$ and the child node $\boldsymbol{\tau'}$ respectively, one has

$$c'_i = c_i \quad \forall i \neq l \text{ and } c_l = T_L(\hat{\boldsymbol{\lambda}})\,. \tag{2.4.8}$$

This means that the costs c'_i corresponding to all the robots other than l are copied form the vector $C(\boldsymbol{\tau})$ of the parent node $\boldsymbol{\tau}$ and just the cost corresponding to robot l is taken from the lower level graph constructed for the new node. The accumulated transition cost $T_U(\boldsymbol{\tau})$ of a node $\boldsymbol{\tau}$ is defined as

$$T_U(\boldsymbol{\tau}) = \sum_{i=1}^{\mathcal{I}_v} c_i\,, \tag{2.4.9}$$

where c_i are the elements of $C(\boldsymbol{\tau})$. $T_U(\boldsymbol{\tau})$ is an estimate of the energy expended by the group of robots for node $\boldsymbol{\tau}$.

Another cost associated with a node is the total cost. $V_U(\boldsymbol{\tau})$ is the total cost of the node $\boldsymbol{\tau}$. A node is selected for expansion in uniform cost graph search, according to this cost. It is defined as

$$V_U(\boldsymbol{\tau}) = \begin{cases} T_U(\boldsymbol{\tau}) & \text{if } \boldsymbol{\tau} \neq \boldsymbol{\tau}_{goal} \\ T_U(\boldsymbol{\tau}) + K' P_U & \text{if } \boldsymbol{\tau} = \boldsymbol{\tau}_{goal}\,, \end{cases} \tag{2.4.10}$$

$$\text{where } P_U = \sum_{i=1}^{\mathcal{I}_v} \epsilon_i + \sum_{j=1}^{\mathcal{I}_t} \Delta_j + 1\,, \tag{2.4.11}$$

and K' is the total number of unvisited interest points over all robots in the node $\boldsymbol{\tau}$. For obtaining the correct solution, P_U is required to be greater than the maximum possible energy of the system of gliders, given by $\sum_{i=1}^{\mathcal{I}_v} \epsilon_i + \sum_{j=1}^{\mathcal{I}_t} \Delta_j$. Here $\sum_{i=1}^{\mathcal{I}_v} \epsilon_i$ is the sum of the starting energies of the robots and $\sum_{j=1}^{\mathcal{I}_t} \Delta_j$ is the sum of energies of all the thermals. As is obvious, this is the maximum energy attainable by the group of gliders.

The aim now is to find the best goal $\boldsymbol{\tau}^*$, which makes the robots visit as many interest points as possible with the least amount of energy expended over all the gliders. Algorithm 2 is a uniform cost search over Γ_U. It returns the goal node $\hat{\boldsymbol{\tau}} \in \boldsymbol{T}_{goal}$, where $V_U(\hat{\boldsymbol{\tau}}) \leq V_U(\boldsymbol{\tau}), \forall \boldsymbol{\tau} \in \boldsymbol{T}_{goal}$, because it is a uniform cost search. Lemma 2.4.3 presents the guarantees for the algorithm being sound, meaning $\hat{\boldsymbol{\tau}} = \boldsymbol{\tau}^*$. This lemma assumes that the lower level graph search also returns the correct solution, meaning that for each robot it returns a solution with least number of unvisited interest points and the best possible sequence through them. The guarantee of correctness of graph search over low level graph is presented by Lemma 2.4.2.

Algorithm 2 Upper Level Graph Search

Require: $\mathcal{B}_x, \mathcal{B}_\epsilon, \mathcal{B}_g, \mathcal{B}_\theta, \mathcal{B}_{ip}, \mathcal{B}_\gamma, \mathcal{B}_\Delta$

$\Psi = \{\boldsymbol{\tau}_{start}\}$
$V_U(\boldsymbol{\tau}_{start}) = 0$
$V_U(\boldsymbol{\tau}) = \infty \quad \forall \boldsymbol{\tau} \in Vertices(\Gamma_U)$ and $\boldsymbol{\tau} \neq \boldsymbol{\tau}_{start}$
while $\boldsymbol{\tau} \neq \boldsymbol{\tau}_{goal}$ **do**
  remove $\boldsymbol{\tau}$ from Ψ with the smallest $V_U(\boldsymbol{\tau})$
  for all $\boldsymbol{\tau'} \in Succ(\boldsymbol{\tau})$ **and** $V_U(\boldsymbol{\tau'}) = \infty$ **do**
    $T_U(\boldsymbol{\tau'}) =$ as calculated in (2.4.9)
    if $\boldsymbol{\tau'} = \boldsymbol{\tau}_{goal}$ **then**
      $V_U(\boldsymbol{\tau'}) = T_U(\boldsymbol{\tau'}) + K'.P_U$
    else
      $V_U(\boldsymbol{\tau'}) = T_U(\boldsymbol{\tau'})$
    end if
    Insert $\boldsymbol{\tau'}$ in Ψ with cost $V_U(\boldsymbol{\tau'})$
  end for
end while
$\hat{\boldsymbol{\tau}} = \boldsymbol{\tau}$
return $\hat{\boldsymbol{\tau}}$

In the pseudo code for Algorithm 2, Ψ is the Open Set (the set of nodes not yet expanded), and $Vertices(\Gamma_U)$ refers to the set of all possible nodes in Γ_U. Notice that once a node cost has been evaluated, it is not reevaluated. This is unnecessary, because it will have the same waypoint assignments. We know that for the same waypoint assignments and for the same robot the lower level graph built will be exactly the same and will return the same answers. Hence it will have the same T_U.

Lemma 2.4.3 *A search on the graph Γ_U, as described in Algorithm 2, returns the path $\hat{\boldsymbol{\tau}} = \boldsymbol{\tau}^*$, meaning that (a) $\hat{\boldsymbol{\tau}}$ has the minimum number of unvisited interest points $\hat{K}$ and (b) for all $\boldsymbol{\tau} \in \boldsymbol{T}_{goal}$ with $\hat{K}$ unvisited interest points, $T_U(\hat{\boldsymbol{\tau}})$ is the least accumulated transition cost.*

Proof (a) We prove by contradiction. Suppose $\tilde{\boldsymbol{\tau}} \in \boldsymbol{T}_{goal}, \tilde{\boldsymbol{\tau}} \neq \hat{\boldsymbol{\tau}}$, has fewer unvisited interest points $\tilde{K} < \hat{K}$, where $\tilde{K}$ is the number of unvisited interest points in $\tilde{\boldsymbol{\tau}}$. $\tilde{\boldsymbol{\tau}}$ and $\hat{\boldsymbol{\tau}}$ have cost vectors $C(\tilde{\boldsymbol{\tau}})$ and $C(\hat{\boldsymbol{\tau}})$ respectively. Uniform cost search ensures

$$V_U(\hat{\boldsymbol{\tau}}) \leq V_U(\tilde{\boldsymbol{\tau}}) \,,$$

$$\sum_{i=1}^{\mathcal{I}_v} \hat{c}_i + \hat{K} P_U \leq \sum_{i=1}^{\mathcal{I}_v} \tilde{c}_i + \tilde{K} P_U \,,$$

$$(\hat{K} - \tilde{K}) P_U \leq \sum_{i=1}^{\mathcal{I}_v} \tilde{c}_i - \sum_{i=1}^{\mathcal{I}_v} \hat{c}_i \,,$$

$$(\hat{K} - \tilde{K}) P_U \leq \sum_{i=1}^{\mathcal{I}_v} \epsilon_i + \sum_{j=1}^{\mathcal{I}_t} \Delta_j \,.$$

The last step was made possible by the fact that the summation of costs over all the gliders should be less than the maximum possible energy of the system, otherwise the solution is not feasible. Using the definition of P_U and the fact that $\hat{K} - \tilde{K} \geq 1$, we get a contradiction.

(b) Assume there is another node $\tilde{\boldsymbol{\tau}} \in \boldsymbol{T}_{goal}, \tilde{\boldsymbol{\tau}} \neq \hat{\boldsymbol{\tau}}$, with the number of unvisited interest points $\tilde{K}$ and $\tilde{K} = \hat{K}$, where $\hat{\boldsymbol{\tau}}$ has unvisited interest points $\hat{K}$..Uniform cost search ensures

$$V_U(\hat{\boldsymbol{\tau}}) \leq V_U(\tilde{\boldsymbol{\tau}}) \,,$$

$$\sum_{i=1}^{\mathcal{I}_v} \hat{c}_i + \hat{K} P_U \leq \sum_{i=1}^{\mathcal{I}_v} \tilde{c}_i + \tilde{K} P_U \,,$$

$$\sum_{i=1}^{\mathcal{I}_v} \hat{c}_i \leq \sum_{i=1}^{\mathcal{I}_v} \tilde{c}_i \,.$$

Hence node $\hat{\boldsymbol{\tau}}$ has the minimum accumulated transition cost for equal number of unvisited interest points. □

Theorem 2.4.1 *The solution found by BLGS has minimum total number of unvisited interest points $\hat{K}$ and the minimum accumulated transition cost $T_U(\hat{\boldsymbol{\tau}})$ across all robots.*

Proof Lemma 2.4.3 guarantees that the upper level graph search returns a solution with the least number of unvisited interest points $\hat{K}$ and the least accumulated transition cost $T_U(\hat{\boldsymbol{\tau}})$. But this upper level graph search assumes that the lower level graph search returns a feasible path with the least number of unvisited interest points and the best possible sequence through them. This is guaranteed by Lemma 2.4.2; hence the proposed BLGS returns the correct solution. □

2.5 LPA*-Modification

As stated in the earlier sections, in the upper level graph when a new node $\boldsymbol{\tau'}$ is grown from a node $\boldsymbol{\tau}$, a whole new lower level graph Γ_L is grown from scratch. However, only some of the edge costs in the new lower level graph Γ_L would be different from the same graph grown for $\boldsymbol{\tau}$. Hence, intuitively, the search procedure shouldn't be very different from the search done previously. This is the main motivation behind formulating a dynamic search approach to update the lower level graph rather than building it from scratch.

There has been some work done on incremental search procedures [34]. Dynamic SWSFP-FP uses information from previous searches to find shortest paths for a series of path planning problems. This algorithm can potentially do better than solving each path planning problem from scratch.

Lifelong Planning A* is an incremental planning version of A* search algorithm proposed in [35] by Sven Koenig, Maxim Likhachev and David Furcy. It builds on the functionality

of A* and enables dynamic searches with changing cost function and even a changing tree structure. This provides us with a tremendous benefit of not having to generate trees from scratch, if some of the edge costs are changed. Moreover LPA* guarantees less operations than a standard A* search. The following sections detail how BLGS was modified with LPA* in mind.

2.5.1 LPA*-modified Search Procedure

A natural solution to this problem is by using incremental techniques like LPA*. We try modifying the lower level graph for the parent node $\boldsymbol{\tau}$ to obtain the lower level graph for the child node $\boldsymbol{\tau'}$. In principle this should be much more efficient than making a new graph from scratch. Notation used in this section is borrowed from [35].

LPA* deals with finite graphs whose edge cost varies over time. We use LPA* for making the lower level graph search faster. The way this is achieved is by modifying the edge costs for the lower level graph. Now rather than considering only the assigned waypoints, all waypoints are considered while expanding the nodes. This means that when a node is expanded in the modified lower level graph, now it will have successors which have all the waypoints and not just the assigned waypoints. However, the nodes with unassigned waypoints will have infinity cost. It is equivalent to claiming that the graph search can go there, but with infinite edge cost, which is akin to restricting the graph search from going there. The other nodes with assigned waypoints only will be assigned cost, as defined in section 2.4.3.

In this modified version of graph search, when $\boldsymbol{\tau}$ is expanded into $\boldsymbol{\tau'}$, and a new waypoint j is assigned to a robot l, we begin by copying all the lower level graphs corresponding to $\boldsymbol{\tau}$ over to the lower level graphs of $\boldsymbol{\tau'}$. Now only the lower level graph corresponding to $\boldsymbol{\tau'}$ and l is modified, since only l has gained a new waypoint j. Now all the edge costs corresponding to waypoint j are changed from infinity to their actual values. On this graph with changed edge costs, the LPA* algorithm performs computations until this new graph is locally consistent. Local consistency ensures that LPA* search procedure gives us the same guarantees of soundness as the uniform cost and A* search. For more details, the paper

about LPA* search [35] can be consulted. The details of the algorithm are given in Figure 2.3.

LPA* uses a non-negative and consistent heuristic similar to A* search. Consistency of heuristic implies that it satisfies the triangle inequality. The costs for each node are computed in a similar way to A*, but they are kept and carried forward for new iterations. LPA* uses a one-step look ahead value called rhs or right-hand side value for each node, and it always satisfies the following condition.

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{start} \\ \min_{s' \in pred(s)}(g(s') + c(s', s)) & \text{otherwise.} \end{cases}$$

In the algorithm for LPA* given below, s_{start} is the start node of a graph, $pred(s)$ means the set of all predecessors of node s, $g(s)$ is the cost to go from s_{start} to s, and $c(s', s)$ is the cost to go from s' to s. The algorithm for LPA* search is given in Figure 2.3.

A vertex is called locally consistent, if its g-value is equal to its rhs-value. If all nodes are consistent, their g-values satisfy

$$g^*(s) = \begin{cases} 0 & \text{if } s = s_{start} \\ \min_{s' \in pred(s)}(g^*(s') + c(s', s)) & \text{otherwise.} \end{cases}$$

When all nodes are consistent, the search procedure is finished, but LPA* does not make all nodes consistent. It focuses on the nodes that might help in getting to the best path.

LPA* maintains an OPEN set just like A*. The nodes in the OPEN set are ordered by their key values. The key k of a node is defined as

$$k(s) = [k_1(s), k_2(s)] \text{where} k_1(s) = min(g(s), rhs(s)) + h(s) and k_2(s) = min(g(s), rhs(s)).$$

The vertex chosen form the OPEN set is according to lexicographical ordering of the k-values.

The LPA* search procedure is outlined below. The $Main()$ function calls the $Initialize()$ function first. $Initialize()$ sets the initial g-values and rhs-values for all the vertices. After initialization $ComputeShortestPath()$ is called and it works exactly like A* search. Now

procedure CalculateKey(s)
{01} return $[\min(g(s), rhs(s)) + h(s); \min(g(s), rhs(s))]$;
procedure Initialize()
{02} $U = \emptyset$;
{03} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{04} $rhs(s_{start}) = 0$;
{05} U.Insert$(s_{start}, [h(s_{start}); 0])$;
procedure UpdateVertex(u)
{06} if $(u \neq s_{start})$ $rhs(u) = \min_{s' \in pred(u)}(g(s') + c(s', u))$;
{07} if $(u \in U)$ U.Remove(u);
{08} if $(g(u) \neq rhs(u))$ U.Insert$(u,$ CalculateKey$(u))$;
procedure ComputeShortestPath()
{09} while (U.TopKey()$\dot{<}$CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal}))$
{10} $u =$ U.Pop();
{11} if $(g(u) > rhs(u))$
{12} $g(u) = rhs(u)$;
{13} for all $s \in succ(u)$ UpdateVertex(s);
{14} else
{15} $g(u) = \infty$;
{16} for all $s \in succ(u) \cup \{u\}$ UpdateVertex(s);
procedure Main()
{17} Initialize();
{18} forever
{19} ComputeShortestPath();
{20} Wait for changes in edge costs;
{21} for all directed edges (u, v) with changed edge costs
{22} Update the edge cost $c(u, v)$;
{23} UpdateVertex(v);

Figure 2.3: LPA* Search Pseudo code

we change the edge costs. $UpdateVertex()$ is called to update the keys and rhs-values of all the nodes affected by the changes. It also rearranges them in the $OPEN$ set according to lexicographic ordering. Finally, the $ComputeShortestPath()$ is called to recompute the shortest path.

2.5.2 Evaluation of LPA*-modification

LPA* and the previous algorithm were tested simultaneously on the same datasets. They are matched against exhaustive search for performance.

Although LPA* is guaranteed to perform less operations than A* search, it lags behind in

Algorithm type	Exhaustive (sec)	MTGS (sec)	MTGS with LPA* (sec)
Test Cases			
4 interest points, 4 thermals, 2 robots	5684	2369	4084
4 interest points, 3 thermals, 2 robots	227	106	201
3 interest points, 3 thermals, 2 robots	6	23	25

converting this into actual benefit. The reason being that in Bi-level Graph Search a lower level graph has to be stored for each node in the top level graph. This results in a lot of save-retrieve operations for the computer and hence worsens the efficiency. Moreover, the amount of stored data grows exponentially. One positive thing about LPA* is that it starts to catch up as the number of interest points and thermals increase.

2.6 Simulations

The algorithms described in the section have been rigorously tested and evaluated for accuracy and efficiency. An exhaustive algorithm is also implemented for checking the correctness of the proposed algorithm. The proposed algorithm is first implemented in MATLAB on an Intel Core 2 Quad CPU running at 2.66 GHz. These algorithms are then compiled using MATLABs Code Compiler functionality to test the speed of the algorithm. The compiled version of an algorithm is naturally much faster than the uncompiled version, and since MATLAB is an interpreted language, compiling the code can give us an increase in efficiency by orders of magnitude. The MATLAB code for the algorithms are given in the Appendix. Below are some of the test cases that were performed. Robot start points are represented as circles, goals as crosses, thermals as diamonds and interest points as squares.

2.6.1 Test Cases

Below some test cases are presented to showcase that the algorithm works and provides the optimal path. The first case shows that the algorithm chooses the best ordering of waypoints.

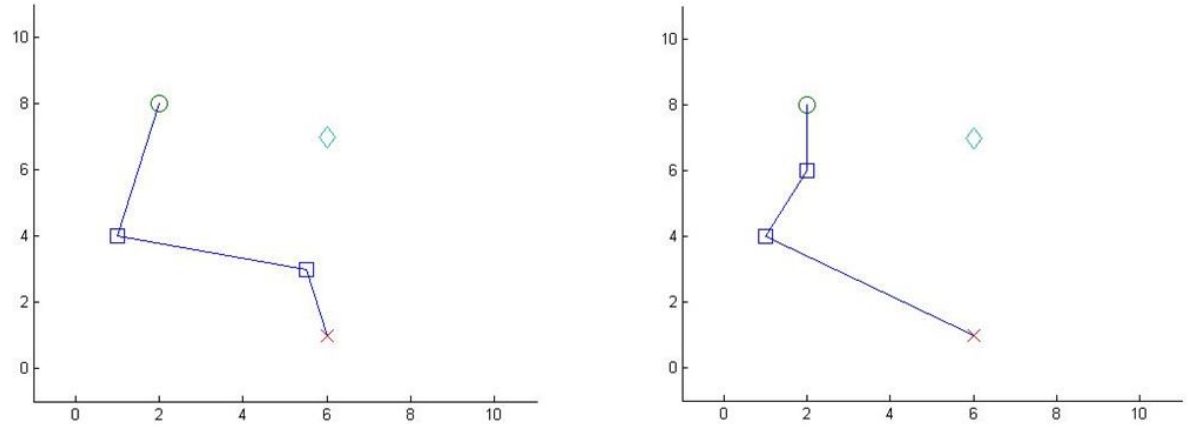

Figure 2.4: Interest points change order

Figure 2.4 depicts a robot which has to visit two interest points and is also provided with a thermal. As can be seen the interest point that is closer to the goal of the robot is shifted towards the starting point such that changing the ordering of the two interest points in the

path becomes a better option for the robot. Notice that the thermal is not utilized since the robot has enough energy to traverse the path and going to the thermal will cost extra energy.

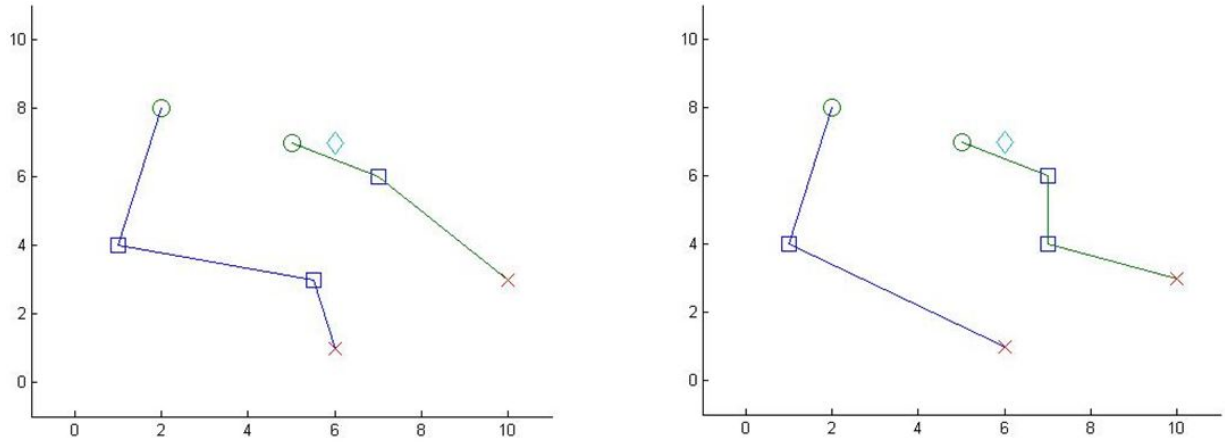

Figure 2.5: Interest points swap routes

Figure 2.5 shows two robots which have to visit 3 interest points and are provided with a thermal. One interest point, which is in the path of one robot, is shifted such that it comes closer to the path of the other robot. At this point it becomes cheaper for the other robot to visit the interest point and hence it swaps routes. The thermal is again left untouched, since it is not needed.

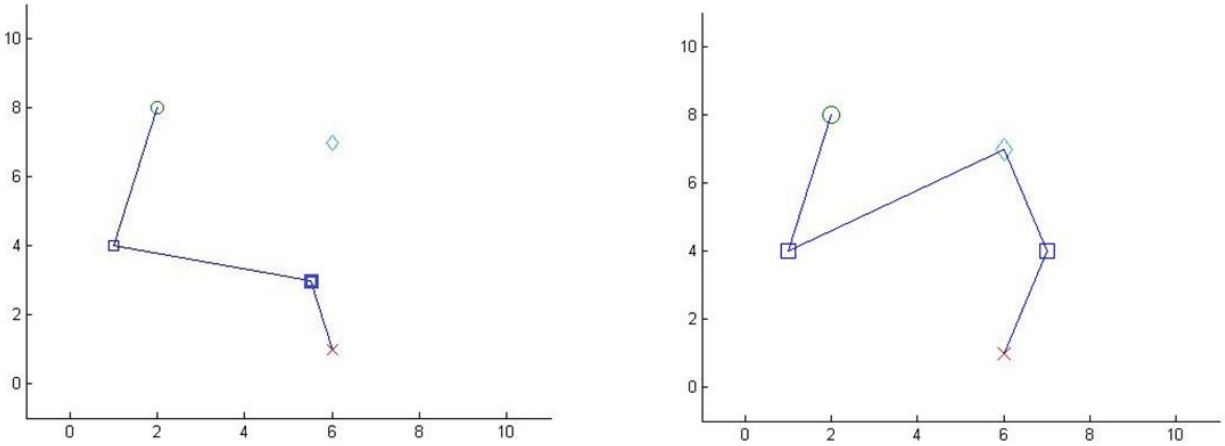

Figure 2.6: Robot starts using thermal when path becomes infeasible

Figure 2.6 depicts a robot, which has to visit two interest points and is also provided with a thermal. The interest point closer to the goal is moved away from the path such that

the starting energy of the robot is insufficient to visit both interest points and reach the goal. Hence the thermal is acquired by the robot to gain the extra energy and complete the mission.

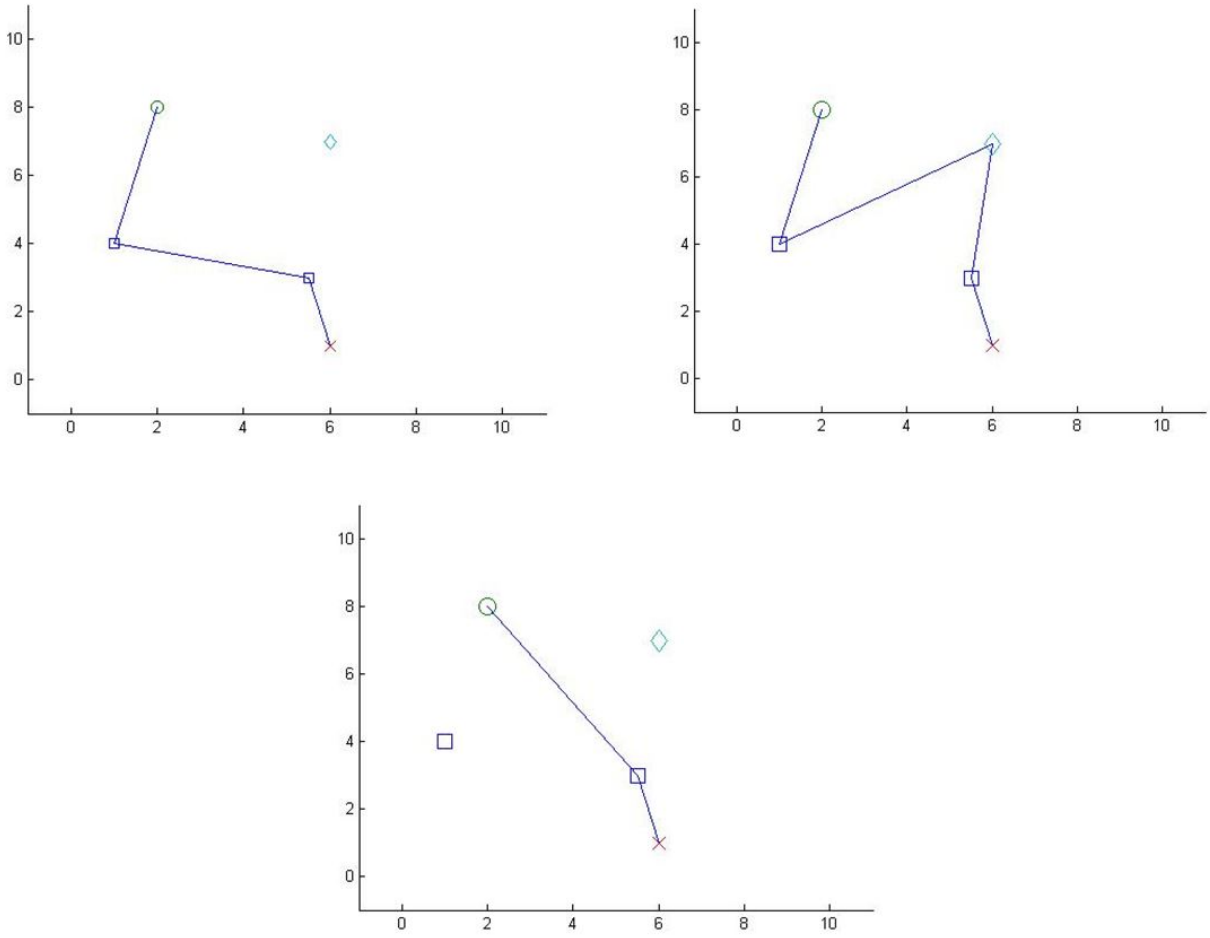

Figure 2.7: Robot modifies path due to different initial energy

Figure 2.7 depicts the optimal path computed by the algorithm as the starting energy of the robot is gradually decreased. In the first case the starting energy is enough to get it to all the Interest Points without using the Thermal. In the second case the starting energy has been decreased, so now it has to utilize the thermal to visit both interest points. In the third case the starting energy has been decreased further; now if it goes to the thermal it can still get one interest point. So it saves its energy that would have been spent to go to the thermal and visits just one interest point.

2.6.2 Two robots, three thermals and four interest points

The following test cases present the performance of the algorithm on various test cases with two robots, three thermals and four interest points. The robots' starting points are represented as circles, goals as crosses, thermals as diamonds and interest points as squares.

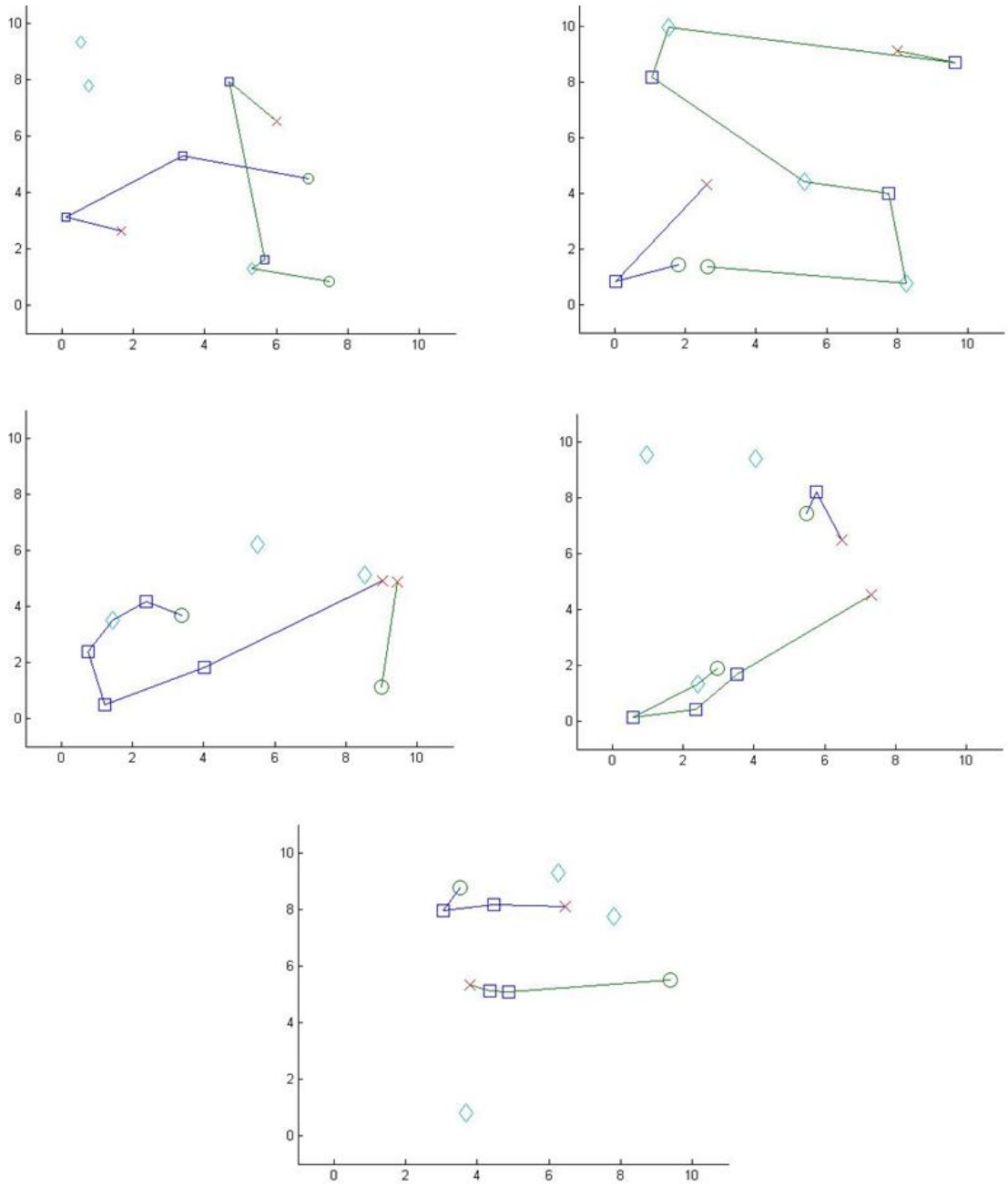

Figure 2.8: Test cases with 2 robots, 3 Thermals and 4 Interest Points

Furthermore the algorithm was tested several times on different combinations of interest

points and thermals. The mean it took for the algorithm to compute a solution is tabulated in the table below.

Algorithm type	Exhaustive (sec)	MTGS (sec)	MTGS_mex (sec)
Test Cases			
4 interest points, 4 thermals, 2 robots	5684	2369	24
4 interest points, 3 thermals, 2 robots	227	106	2
3 interest points, 3 thermals, 2 robots	6	23	1

When compiled on average the algorithm takes 100 times less, which is an unprecedented decrease in execution time.

CHAPTER 3

ENERGY-PRESERVING PATH PLANNING

Now that the best way of sequencing the interest points and thermals have been computed, the next step is to plan close-to-optimal, feasible paths for the gliders in a time efficient manner. The paths being feasible means that they have to be achievable by the robot. This means that the paths respect the dynamics of the robot. This thesis only plans paths but doesn't deal with controlling the gliders to follow those paths. The task of designing a trajectory tracking controller has been explored in the literature; see [36], [37] and [38].

Furthermore the path has to be as close to optimal as possible provided that it is computed in a timely fashion. For the purposes of this thesis an optimal path is defined as a path achievable by the glider, which goes from one point to another with the least loss-of-energy of the glider during the commute. Since the problem is non-convex, a suboptimal solution is found. Hence, the method proposed in this thesis plans a close-to-optimal path for the gliders, but in doing so assures a timely execution of the algorithm. For this reason we call the approach Energy-preserving Path Planning.

As explained in the succeeding sections, the flight dynamics of a glider can only follow paths that fulfill certain constraints. Because of the non-availability of thrust, pure gliding flight cannot achieve trajectories planned for regular aircraft; for example, if at time t_0 the glider has attained a height h_0 at a certain speed s_0, it cannot achieve height h_0 with speed s_0 for time $t > t_0$ without any external source of energy. Hence the glider is losing its mechanical energy at a steady speed. In conditions like these, it becomes imperative for the path planning to be as close-to-optimal as possible.

In gliding flight, loss of energy manifests itself as loss of height, since the velocity of the glider doesn't deviate too much. As is discussed in section 3.2.1, loss of height primarily depends on the path length and also on the turn rate of the glider. Hence, optimizing energy

loss translates to optimizing path length and turn rate. For the purposes of optimizing energy loss, a fast way to compute these expressions is needed. Pythagorean Hodograph (PH) curves proposed in [39] solve these problems by forcing the parametric speed of the curves to have a particular structure. This thesis uses PH curves for the purpose of planning paths for the gliders.

In the following sections we start with a review of literature relevant to the topic of path planning. The next section discusses the dynamics of gliding flight and how choosing an optimal velocity for the glider reduces the 3 dimensional optimization problem to a 2 dimensional optimization problem. This is followed by an introduction to the Pythagorean Hodograph Bézier curves. Then the solution to EPP is proposed. In this section we discuss how PH Bézier curves are patched together to form a path that optimizes energy loss of the glider, while respecting the dynamic constraints of the glider. The last section presents simulation results that validate our theoretical findings.

3.1 Literature Review

Trajectory Generation as a problem has been widely studied. Optimal trajectory planning by using multiple shooting methods is proposed by [19], [20]. Multiple shooting methods are numerical methods for solving boundary value ordinary differential equations.

Pseudospectral optimal control methods proposed by [21], [22] have recently gained momentum. Pseudospectral optimal control is a combination of pseudospectral theory and optimal control theory. Pseudospectral methods are numerical methods for solving partial differential equations. Pseudospectral optimal control is an approximate method which solves the optimal control problem by approximating continuous functions by quadrature nodes. Using this approximation the optimal control problem is solved very efficiently. PS optimal control has been very successful in solving important problems. In March 2007, the International Space Station executed a Zero Propellant Move planned using a pseudospectral optimal control method. As for cooperative trajectory generation, few publications [21], [22] have proposed a PS optimal control methodology that ensures temporal deconfliction of trajectories, but spatial deconfliction cannot be guaranteed.

Nonlinear trajectory generation methods proposed by [23], [24] use mapping of higher dimensional state variables to lower dimensional output space variables to plan optimal paths. Randomized trajectory generation methods have recently gained momentum with examples such as Rapidly-exploring Random Trees (RRT) [25], Probabilistic Roadmaps (PRM) [26] and direct method for rapid trajectory prototyping [27]. But these randomized trajectory generation methods are ill-suited for optimal path planning in general.

Pythagorean Hodograph (PH) Bézier curves [39] provide the solution to our problems. We use PH Bézier curves for the problem of path planning, since they provide closed-form analytical expressions for the arc length, velocity, turn rate and other useful variables that need to be constrained and also optimized over. PH Bézier curves are discussed in great detail in the succeeding sections.

3.2 Gliding Flight and Soaring

Gliding flight is heavier-than-air flight without any means of propulsion. In nature, gliding flight has been used for millennia by animals like gliding squirrels, hylid frogs and even snakes. The first human to successfully design and build a flyable airplane was Sir George Cayley , an Englishman, in early 19th Century. He was the first one to understand the importance of streamlining and a need for cambered aerofoil for efficient lifting power. In 1804, he introduced the stabilizing tailplane and fin mounted behind the wing, based on his work with model gliders. It proved so successful that it has been a standard followed by aircraft designers ever since.

Thermal Soaring is the exploration and exploitation of *free energy* present in the environment (like convective air flow and wind shear) by a glider-like aircraft to increase its flight endurance. Soaring was first performed by the Wright brothers by using the up current winds created by the wind blowing over a ridge of hills. Using this procedure, they were able to make an extraordinary nine minutes and 45 seconds long flight. Their other contributions to flight include, among many things, also introduction of control surfaces as an active flight control mechanism. Gliding as a sport emerged after the first World War amongst Europeans, especially Germans, since it was not banned for them under the Treaty of Versailles.

The ridges hills effect was very well known, but another phenomenon was soon discovered that was even more dependable and powerful. This was the phenomena of columns of rising hot air also called thermals.

In the following sections we describe the flight dynamics of a glider in symmetric and turning flight. We are primarily concerned with minimizing the loss of height of the glider as it travels over a path. For given flight conditions, like angle of attack of aircraft and bank angle, the loss of height of the glider is predetermined. As it turns out, for each glider there is an optimal angle of attack, which minimizes its loss of height. If we use this angle of attack throughout the path the loss of height will vary with the curvature of the path alone. So in effect we have converted a 3-dimensional path planning problem into a 2-dimensional one.

3.2.1 Dynamics of a Glider in symmetric flight

The equations that govern the longitudinal flight dynamics of a single glider traveling in symmetric flight, as taken from [40], are

$$v = \frac{2W}{S\rho C_L}\cos(\gamma_d)\,, \tag{3.2.1}$$

$$\tan(\gamma_d) = \frac{C_D}{C_L}\,, \tag{3.2.2}$$

$$RD = v\sin(\gamma_d) = \sqrt{\frac{2WC_D^2}{S\rho C_L^3}\cos^3(\gamma_d)}\,, \tag{3.2.3}$$

where v is velocity of the glider, W is weight, S the wing area, ρ the density of air at the altitude the glider is flying, C_D is the drag coefficient, C_L is the lift coefficient, γ_d is the flight path angle of the glider (positive downwards) and RD is the rate of descent of the glider (also positive downward).

We assume that the glider is traveling at low subsonic flight speeds, the air is static and Reynolds number effects are negligible. We further assume that the glider has a parabolic lift-drag polar [40] meaning that the relationship between lift coefficient C_L and drag coefficient

C_D can be expressed as a 2nd order equation:

$$C_D = C_{D0} + K_{LD}(C_L - C_{L0})^2\,, \tag{3.2.4}$$

where C_{D0} and K_{LD} are constants and depend on the shape of the glider. Changing the angle of attack α of the glider corresponds to a change in the lift coefficient C_L. In turn, that changes the drag coefficient C_D by virtue of the relationship 3.2.4.

The objective is to determine a path that loses the least amount of energy, while the glider travels a certain distance along the path. Alternatively, this can be stated as, a path needs to be planed that maximizes the distance traveled s, while the glider is falling a certain height Δh. The relationship between the two is

$$s = \Delta h / \tan(\gamma_d)\,. \tag{3.2.5}$$

As is evident from Equation (3.2.5) the flight path angle needs to be minimized. This can be achieved by maximizing the lift-to-drag ratio C_L/C_D according to Equation (3.2.2). This is depicted in Fig. 3.1.

The optimal flight path angle obtained using $(C_L/C_D)_{max}$ is $\gamma_{d_{min}}$. The velocity corresponding to these flight conditions is called v_{opt}, and the angle of attack for these conditions is called α_{opt}.

This is true for symmetric flight, but for the case of turning flight, the descent rate RD is higher than what is given in Equation (3.2.3). The reason for that is that part of the lift force is used up to counter the centrifugal force induced by the turning, and hence the glider falls faster. This is discussed in the following section.

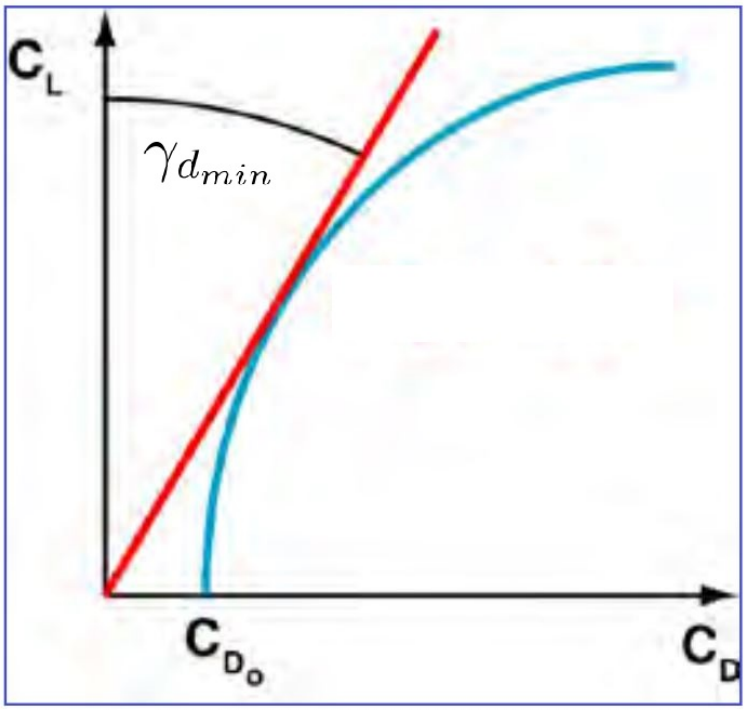

Figure 3.1: Lift Drag Polar and $\gamma_{d_{min}}$

3.2.2 Dynamics of a Glider in Turning Flight

The dynamics of a glider in turning flight, as stated in [40], is given below. Here the glider flight path angle is assumed to be close to zero, $\gamma_d \approx 0$.

$$v = \sqrt{\frac{2W}{S\rho C_L \cos(\phi)}}\,, \tag{3.2.6}$$

$$\tan(\gamma_d) = \frac{C_D}{C_L \cos(\phi)}\,, \tag{3.2.7}$$

$$RD = \sqrt{\frac{2WC_D^2}{S\rho C_L^3 (\cos(\phi))^3}}\,, \tag{3.2.8}$$

$$\kappa = \frac{g\tan(\phi)}{v^2}, \tag{3.2.9}$$

where ϕ is the bank angle, κ is the curvature of the path taken by the glider and the other symbols are defined as in Section 3.2.1.

Using the same reasoning used in the previous section, to optimize the path, we need to

minimize the flight path angle γ_d. Notice that the C_D and C_L values that optimize the path in symmetric flight also optimize it in gliding flight. Hence the angle of attack α_{opt} also does not need to change. The only difference is that now $\tan(\gamma_d)$ is scaled by a constant, which depends on the bank angle ϕ.

For a glider to have an achievable path, the curvature of the glider must be bounded $|\kappa| < \kappa_{max}$. This means that the bank angle of the glider must also be upper and lower bounded. This translates to $\gamma_{d_{min}}$ also being bounded. So the extra amount of descent caused by turning flight will also be bounded. Hence, this thesis neglects the effects of higher descent rate due to turning flight. We assume that the glider flight path angle γ_d is constant and equal to $\gamma_{d_{min}}$, mentioned in Section 3.2.1.

So now that the 3-dimensional path planning problem has been converted to a 2-dimensional path planning problem, we turn to optimization. Since we assume that the flight path angle is constant $\gamma_{d_{min}}$, the primary aim of this section is to minimize the arc length of the curve. Generally speaking, computing the arc length of a curve is not an easy problem. Reference [41] introduces the Pythagorean Hodograph (PH) Bézier Curves that have the property of having an analytical expression for the arc length of the curve. Due to this great ease of computing the arc length, the computational cost of the problem is drastically reduced. The succeeding sections deal with using PH Bézier curves to plan paths for the gliders.

3.3 Pythagorean Hodograph Bézier Curves

3.3.1 Bézier curves

Bézier curves were invented by Dr. Pierre Bézier as a tool to design shapes for car manufacturing in the early 1960s. The curve $\boldsymbol{r}$ is defined by a set of control points $\bar{r}_k \in \mathbb{R}^2$:

$$\boldsymbol{r}(\zeta) = \sum_{k=0}^{n} \bar{r}_k b_k^n(\zeta)\,, \tag{3.3.1}$$

where n is the degree of the Bézier curve, $\zeta \in [0,1]$ is a dimensionless parameter, and $b_k^n(\zeta)$ are the Bernstein polynomials defined as

$$b_k^n(\zeta) = \binom{n}{k}(1-\zeta)^{n-k}\zeta^k\,. \tag{3.3.2}$$

The control points $\bar{r}_1, ..., \bar{r}_n \in \mathbb{R}^2$ define the Bézier curve. The curve starts at the first control point and ends at the last control point. The curve is entirely contained inside the control polygon (the convex polygon defined by the control points). The curve starts tangent to the line joining $\bar{r}_0$ and $\bar{r}_1$ and ends tangent to the line joining $\bar{r}_{n-1}$ and $\bar{r}_n$. The set of Bézier curves also have the useful property of being closed under the operations of differentiation, integration, summation, multiplication and composition among others. The parametric derivative of a Bézier curve $\boldsymbol{r}(\zeta)$ is also a Bézier curve, whose control points depend on the control points of $\boldsymbol{r}(\zeta)$:

$$\boldsymbol{r'}(\zeta) = d\boldsymbol{r}/d\zeta = \boldsymbol{h}(\zeta) = \sum_{k=0}^{n-1} \bar{h}_k b_k^{n-1}(\zeta)\,, \tag{3.3.3}$$

$$\bar{h}_k = n(\bar{r}_{k+1} - \bar{r}_k) \text{ for } k = 0, ..., n-1\,.$$

Similarly the product of two Bézier curves is also a Bézier curve. Let $\boldsymbol{r}(\zeta)$ be a Bézier curve of degree n and $\boldsymbol{w}(\zeta)$ be a Bézier curve of degree m. Then the control points $\bar{c}_k$ of their product $\boldsymbol{c}(\zeta) = \boldsymbol{r}(\zeta).\boldsymbol{w}(\zeta)$ are given by (3.3.4)

$$\bar{c}_k = \sum_{j=max(0,k-n)}^{min(m,k)} \frac{\binom{m}{j}\binom{n}{k-j}}{\binom{m+n}{k}} \boldsymbol{r}_{k-j}\boldsymbol{w}_j \text{ for } k = 0, ..., (m+n)\,. \tag{3.3.4}$$

3.3.2 Pythagorean Hodograph Curves

The arc length of a curve can be calculated as follows

$$s(\zeta) = \int_0^\zeta ||r'(\omega)||d\omega = \int_0^\zeta \sqrt{x'^2(\omega) + y'^2(\omega)}d\omega\,.$$

The the parametric speed of the curve $\sigma(\zeta)$ is given by

$$x'^2(\zeta) + y'^2(\zeta) = \sigma^2(\zeta)\,. \tag{3.3.5}$$

This results in arc length

$$s(\zeta) = \int_0^{\zeta} \sigma(\omega)\delta\omega\,. \tag{3.3.6}$$

PH curves force a Pythagorean structure on the hodograph $\boldsymbol{r'}(\zeta) = [x'(\zeta), y'(\zeta)]^T$ of the curve. Theorem 17.1 of [41] states that the PH condition will be fulfilled, if the hodograph $\boldsymbol{r}'(\zeta)$ is defined as

$$\begin{aligned} x'(\zeta) &= u^2(\zeta) - v^2(\zeta)\,, \\ y'(\zeta) &= 2u(\zeta)v(\zeta)\,, \\ \sigma(\zeta) &= u^2(\zeta) + v^2(\zeta)\,. \end{aligned} \tag{3.3.7}$$

If $\boldsymbol{r}(\zeta)$ is a Bézier curve, $u(\zeta)$ and $v(\zeta)$ are also lower-order Bézier curves by virtue of the closedness property of the class of Bézier curves. Having found a closed form solution for the arc length, an expression for the curvature of the curve is needed. It is given as

$$\kappa(\zeta) = 2\frac{u(\zeta)v'(\zeta) - u'(\zeta)v(\zeta)}{\sigma^2(\zeta)}\,. \tag{3.3.8}$$

Since $u(\zeta)$ and $v(\zeta)$ are Bézier curves, $\kappa(\zeta)$ can be transformed into a rational Bézier curve. For a trajectory to be *continuously flyable*, the curvature also needs to be continuous throughout the trajectory.

3.4 Solution for a Single Glider

This section deals with using PH Bézier curves to plan smooth paths over the set of waypoints that need to be visited by a glider. The waypoints are the solutions to the VSP, computed by BLGS as described in Chapter 2 .Each section of the trajectory going from waypoint-to-

waypoint is called a path. Each path is a PH Bézier curve denoted by $\boldsymbol{r}_j^l$

$$\boldsymbol{r}_j^l(\zeta_j^l) = \sum_{k=0}^{n} \bar{r}_{k,j}^l b_k^n(\zeta_j^l)\,, \tag{3.4.1}$$

where $l \in \mathcal{I}_v$ is a robot from the group of robots, n is the order of the curve and $1 \leq j \leq Card(\hat{\boldsymbol{\lambda}}_{\hat{\boldsymbol{\tau}},l})$, where j corresponds to a waypoint in the sequence $\hat{\boldsymbol{\lambda}}_{\hat{\boldsymbol{\tau}},l}$. The sequence $\hat{\boldsymbol{\lambda}}_{\hat{\boldsymbol{\tau}},l}$ is the sequence produced by the lower level graph search, corresponding to the node $\hat{\boldsymbol{\tau}}$ (in upper level graph) for robot l, $\bar{r}_{k,j}^l$ is the control point for the jth trajectory of lth glider. $\zeta_j^l \in [0,1]$ is the dimensionless parameter for this path. All the paths corresponding to the robot l, joined together are called the route of l.

Quintic (fifth order) PH Bézier curves are used to generate the paths. The reason for using quintics is that they are the simplest PH Bézier curves that can be used for first-order Hermite interpolation [39].

These paths need to be patched together to form a route. To satisfy the continuity of the route for robot l, the paths $\boldsymbol{r}_j^l$ need to satisfy the following constraints:

$$\boldsymbol{r}_1^l(0) = x_l \in \mathcal{B}_x, \boldsymbol{r}_i^l(1) = \boldsymbol{r}_{i+1}^l(0) = \hat{\lambda}_i, \forall 1 \leq i \leq (n-1),$$

$$\boldsymbol{r}_n^l(1) = g_l \in \mathcal{B}_g\,, \tag{3.4.2}$$

where $\hat{\lambda}_i \in \hat{\boldsymbol{\lambda}}_{\hat{\boldsymbol{\tau}},l}$. This states that the paths need to start and end at the corresponding waypoints and the first and the last waypoints are the start and the goal positions of the robot, respectively. Since the robot also has a starting orientation, we have

$$y'^l_1(0)/x'^l_1(0) = \tan(\theta_l), \text{ where } \theta_l \in \mathcal{I}_\theta\,. \tag{3.4.3}$$

Moreover, the curvature of the paths also needs to be bounded. This constraint can be expressed as:

$$|\kappa_j^l(\zeta_j^l)| \leq \kappa_{max} \quad \forall \zeta_j^l \in [0,1]\,. \tag{3.4.4}$$

Two paths meeting at an interest point need to satisfy continuity of parametric speed and

curvature. We start with parametric speed:

$$\frac{d}{d\zeta_j^i}\begin{bmatrix}x_j^i(\zeta_j^i)\\ y_j^i(\zeta_j^i)\end{bmatrix}\Bigg|_{\zeta_j^i=1} = \frac{d}{d\zeta_{j+1}^i}\begin{bmatrix}x_{j+1}^i(\zeta_{j+1}^i)\\ y_{j+1}^i(\zeta_{j+1}^i)\end{bmatrix}\Bigg|_{\zeta_{j+1}^i=0},$$

$$\boldsymbol{r'}_j^l(1) = \boldsymbol{r'}_{j+1}^i(0)\,,$$

using equations (3.3.3), (3.3.4) and (3.3.7)

$$[(\bar{u}_{2,j}^l)^2 - (\bar{v}_{2,j}^l)^2, 2\bar{u}_{2,j}^l\bar{v}_{2,j}^l]^T = [(\bar{u}_{0,j+1}^l)^2 - (\bar{v}_{0,j+1}^l)^2, 2\bar{u}_{0,j+1}^l\bar{v}_{0,j+1}^l]^T\,.$$

This condition is satisfied, if

$$\bar{u}_{0,j+1}^l = \bar{u}_{2,j}^l,\ \bar{v}_{0,j+1}^l = \bar{v}_{2,j}^l\,. \tag{3.4.5}$$

Furthermore the curvature of the route also needs to be continuous. The curvature $\kappa_j^l(\zeta_j^l)$ throughout the length of one path will always be continuous, since it is a rational Bézier curve and it is bounded by κ_{max}. The curvature of route is also required to be continuous at the interest points. If $j \in \mathcal{I}_{ip}$ then

$$\kappa_j^l(1) = \kappa_{j+1}^l(0)\,,$$

$$2\frac{u_j^l(1)v'^l_j(1) - u'^l_j(1)v_j^l(1)}{(\sigma_j^l(1))^2} = 2\frac{u_{j+1}^l(0)v'^l_{j+1}(0) - u'^i_{j+1}(0)v_{j+1}^l(0)}{(\sigma_{j+1}^l(0))^2}\,,$$

using equations (3.3.1), (3.3.5), (3.3.7) and (3.3.3), we have

$$\frac{\bar{v}_{2,j}^l\bar{u}_{1,j}^l - \bar{u}_{2,j}^l\bar{v}_{1,j}^l}{((\bar{u}_{2,j}^l)^2 + (\bar{u}_{2,j}^l)^2)^2} = \frac{\bar{u}_{0,j+1}^l\bar{v}_{1,j+1}^l - \bar{v}_{0,j+1}^l\bar{u}_{1,j+1}^l}{((\bar{u}_{0,j+1}^l)^2 + (\bar{u}_{0,j+1}^l)^2)^2}\,.$$

The constraint is satisfied, if

$$\bar{v}_{2,j}^l(\bar{u}_{1,j+1}^l + \bar{u}_{1,j}^l) = \bar{u}_{2,j}^l(\bar{v}_{1,j}^l + \bar{v}_{1,j+1}^l)\,. \tag{3.4.6}$$

Notice that the constraints (3.4.5), (3.4.6) need to be satisfied at interest points, but not at

thermals. This is due to the spiral ascending maneuverer that the gliders execute to gain energy at thermals. This thesis deals with the routes leading up to the thermals, but not the ascending maneuver.

Now that the route has been formulated it needs to be optimized, while making sure that the constraints are met. The optimization variables are $\bar{u}^l_{k,j}, \bar{v}^l_{k,j}$. The cost function to be optimized is the arc length of the curve $s^l_j(1)$ for each path and for each robot, while the constraints (3.4.5), (3.4.6) and (3.4.4) are respected.

So now the problem can be formulated as an optimization problem, where the cost to be minimized is

$$\min_{\bar{u}^l_{k,j}, \bar{v}^l_{k,j}} \sum_{l=1,j=1}^{l=n_v, j=Card(\hat{\boldsymbol{\lambda}}_{\hat{\boldsymbol{\tau}},l})} s^l_j(1)\,, \tag{3.4.7}$$

$$\text{subject to constraints (3.4.2)-(3.4.6)}\,,$$

where $l \in \mathcal{I}_v$, n is the number of elements in $\hat{\boldsymbol{\lambda}}_{\hat{\boldsymbol{\tau}},l}$.

The function *fmincon* from the MATLAB Optimization Toolbox is used for solving the optimization problem (3.4.7). The next section discusses simulation results for the approach.

3.5 Simulations

We simulate several scenarios with 2 robots that have to achieve 4 interest points cooperatively, and there are 3 thermals available. As in Section 2.6 the squares represent interest points, circles represent the starting points of gliders and crosses their goals and the thermals are represented by diamonds. The control points for the PH Bézier curves are also shown.

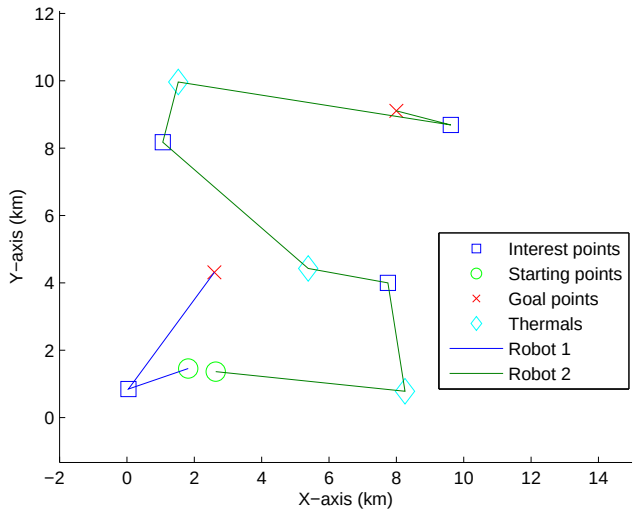

Figure 3.2: VSP for Scenario 1

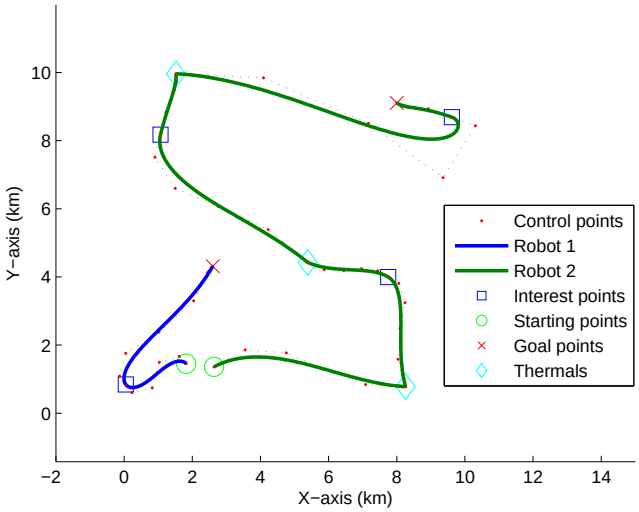

Figure 3.3: EPP for Scenario 1

First, the VSP plans the assignment and sequencing of the waypoints for each robot, and

then the EPP plans the best path over the sequence. The curvature of the paths needs to be continuous at the interest points, but not at the thermals as discussed in section 3.4. The PH Bezier curve planned for EPP in scenario 2 does a loop in the beginning, because the maximum curvature constraint restricts the curve to go straight to the interest point.

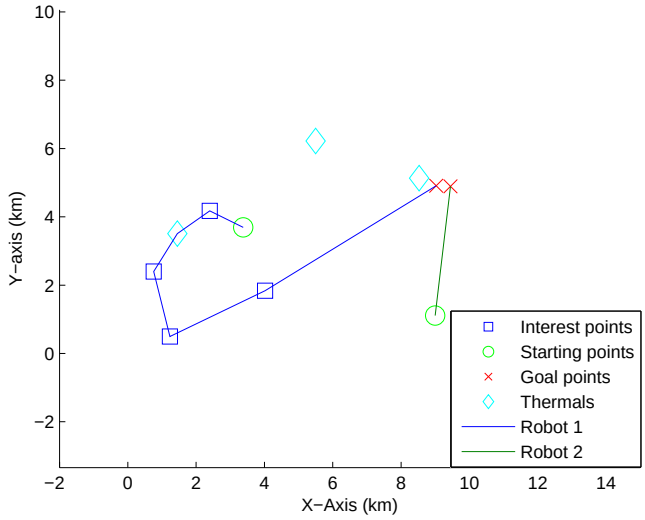

Figure 3.4: VSP for Scenario 2

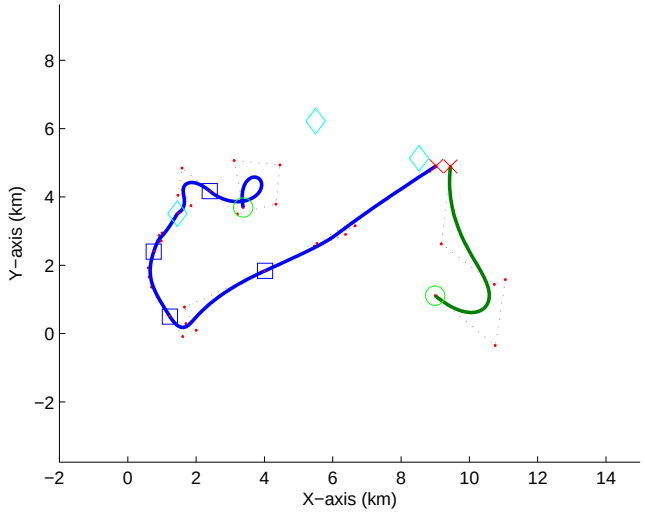

Figure 3.5: EPP for Scenario 2

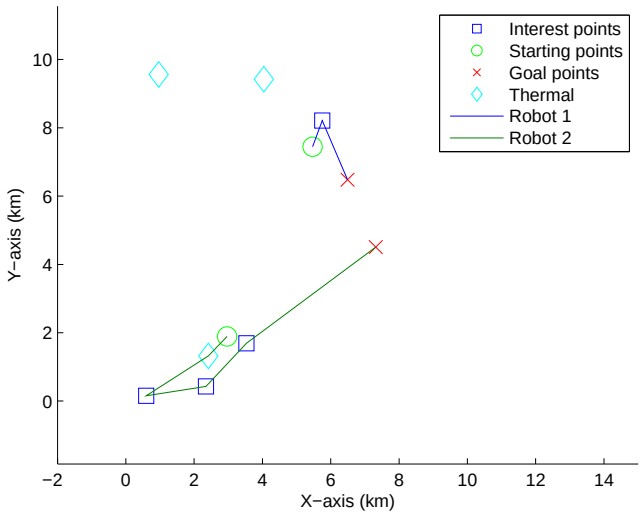

Figure 3.6: VSP for Scenario 3

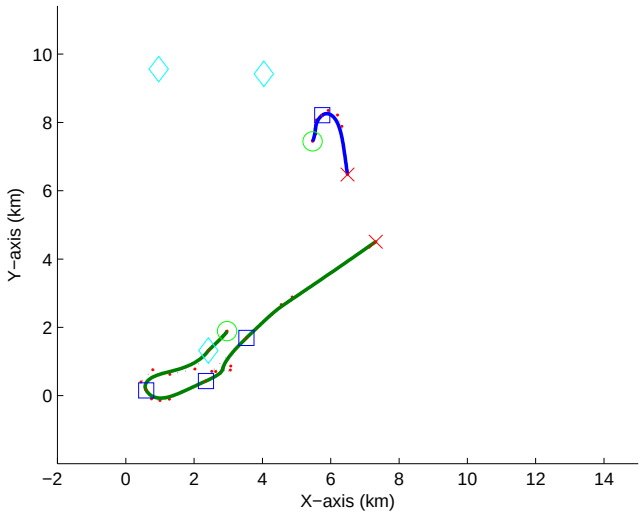

Figure 3.7: EPP for Scenario 3

CONCLUSION

The thesis addresses the problem of a path planning for a group of Cooperative Autonomous Soaring gliders. The task given to the gliders is to visit a set of *interest points*, while also being provided a set of *thermals*. Thermals can be used by the gliders to gain energy and increase the reach of the gliders. Since the gliders are constantly losing energy (while not visiting thermals) the paths planned should minimize the energy lost by the gliders. The problem is decoupled into two subproblems, (i) assigning the waypoints (interest points and thermals) to the robots and computing the best sequence over them, (ii) planning achievable paths over these sequences, which optimizes the energy lost by the gliders.

The first problem also called VSP, is solved by modifying a Multi-Tier Graph Search (MTGS) based approach (originally meant to solve the Euclidean Team Orienteering Problem). The problem is to find the best assignment of waypoints for the gliders. The different assignments, are evaluated based on the energy lost by the gliders, while traveling over these assignments. Since the actual energy lost is not known, an estimate is used instead. The Bi-Level Graph Search (BLGS), which is a modified version of MTGS, then uses this estimate to find the best assignment. One thing to remember here is that this best assignment depends on how good the estimate of energy is. For the kind of estimate used in this thesis, as the distances between the waypoints increase, the estimate approaches the actual value.

The second problem also called the EPP, is solved by using Pythagorean Hodograph (PH) Bézier curves. Once the best assignment of waypoints to robots has been computed by the VSP, we plan paths for the gliders over these assignments. The paths must be achievable by the gliders and must minimize the energy lost by the gliders. As it turns out, minimizing the arc length of a path corresponds to minimizing the energy lost by gliders. The PH Bézier curves provide us with an easy way to check the feasibility of the path, as well as an

analytical expression for its arc length. Nonlinear programming techniques are used to solve this minimization problem.

Future work for this research includes obtaining performance guarantees for this algorithm like, the upper bound on the energy lost by the gliders, and the upper bound on the deviation, from the most optimal solution.

APPENDIX

A.1 Proof of Theorem 1 Section 2.3.4

Proof:

1) We prove by contradiction. Assume there is a path $\tilde{\pi}_{\bar{G}_M}$ that goes from start to goal whose time does not exceed T_{MAX} and has a larger number of waypoints $\tilde{k} > \hat{k}$.

$c(\hat{\pi}_{\bar{G}_M}) \leq c(\tilde{\pi}_{\bar{G}_M})$ from Lemma 2.

$\Rightarrow \sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i) + (M - \hat{k}) \times P_M \leq \sum_{i=1}^{\tilde{k}+1} c(n_{i-1}, n_i) + (M - \tilde{k}) \times P_M$

$\Rightarrow (\tilde{k} - \hat{k}) \times (T_{MAX} + 1) \leq \sum_{i=1}^{\tilde{k}+1} c(n_{i-1}, n_i) - \sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i)$

Any path cost in G_M is bounded from above by T_{MAX}.

$\Rightarrow (\tilde{k} - \hat{k}) \times (T_{MAX} + 1) \leq T_{MAX}$

This leads to a contradiction since $\tilde{k} > \hat{k}$. Thus, the maximum number of waypoints that can be covered is $\hat{k}$.

2) Assume another solution $\dot{\pi}_{\bar{G}_M} \in \bar{G}_M$ that has same number of waypoints, $\dot{k} = \hat{k}$.

$c(\hat{\pi}_{\bar{G}_M}) \leq c(\dot{\pi}_{\bar{G}_M})$ from Lemma 2.

$\Rightarrow \sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i) + (M - \hat{k}) \times P_M \leq \sum_{i=1}^{\dot{k}+1} c(n_{i-1}, n_i) + (M - \dot{k}) \times P_M$

$\Rightarrow \sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i) \leq \sum_{i=1}^{\dot{k}+1} c(n_{i-1}, n_i)$ since $\dot{k} = \hat{k}$.

Hence, $\sum_{i=1}^{\hat{k}+1} c(n_{i-1}, n_i)$ is minimum $\forall \pi_{\bar{G}_M}$ with fixed number of waypoints $\hat{k}$.

A.2 Proof of Theorem 2 Section 2.3.4

Proof:

1) We prove by contradiction. Assume a solution that has a less number of unvisited waypoints. Consider a path $\tilde{\pi}_{\bar{G}_T}$ such that it has less unvisited waypoints where $\tilde{k} < \hat{k}$.

$c(\hat{\pi}_{\bar{G}_T}) \leq c(\tilde{\pi}_{\bar{G}_T})$ from Lemma 2.

$\Rightarrow \sum_{i=1}^{N} c(\hat{\pi}_i) + \hat{k} \times P_T \leq \sum_{i=1}^{N} c(\tilde{\pi}_i) + \tilde{k} \times P_T$

$\Rightarrow (\hat{k} - \tilde{k}) \times P_T \leq \sum_{i=1}^{N} c(\tilde{\pi}_i) - \sum_{i=1}^{N} c(\hat{\pi}_i)$

$\Rightarrow (\hat{k} - \tilde{k}) \times (\sum_{i=1}^{N} T_{MAX_i} + 1) \leq \sum_{i=1}^{N} T_{MAX_i}$

This leads to a contradiction since $\tilde{k} < \hat{k}$. Thus, the minimum number of unvisited waypoints is $\hat{k}$.

2) Assume another solution $\dot{\pi}_{\bar{G}_T} \in \bar{G}_T$ that has same number of waypoints, $\dot{k} = \hat{k}$.

$c(\hat{\pi}_{\bar{G}_T}) \leq c(\dot{\pi}_{\bar{G}_T})$ from Lemma 2.

$\Rightarrow \sum_{i=1}^{N} c(\hat{\pi}_i) + \hat{k} \times P_T \leq \sum_{i=1}^{N} c(\dot{\pi}_i) + \dot{k} \times P_T$

$\Rightarrow \sum_{i=1}^{N} c(\hat{\pi}_i) \leq \sum_{i=1}^{N} c(\dot{\pi}_i)$ since $\dot{k} = \hat{k}$.

Hence, $\sum_{i=1}^{N} c(\hat{\pi}_i)$ is minimum $\forall \pi_{\bar{G}_M}$ with fixed number of waypoints $\hat{k}$.

A.3 MATLAB Code for Visitation Sequence Planning

```
function [  ] = MultiTierGraphSearch(  )
%MULTITIERGRAPHSEARCH The top level graph search
%   Detailed explanation goes here
%   Author: Muhammad Aneeq uz Zaman
tic;
addpath('C:\Users\mazaman2\Dropbox\Project\path');
% global N M X T_max dt_max thermals intpts goals robot_pos

load('Data&Results/data2.mat','M','N','T_max','X','dt_max',...
'goals','intpts','robot_pos','scale','thermals');
bigmed = zeros(1,N+M+1);
bigmed(1,1) = factorial(N+M+1)/factorial(N+M);
for g = 2:N+M+1
bigmed(1,g) = bigmed(1,g-1) + factorial(N+M+1)/factorial(N+M+1-g);
end
bigtop = (X+1)^(N+M);
open = zeros(1,bigtop);
open = 1;                                   % adding the root to the open set
t = tree(0,bigtop);                         % tree structure for storing waypoint
    ↪ assignment
f_init = zeros(X,1);
best_path_yet = (N+M+1)*ones(bigtop,X);
% adding vertices pertaining to no waypoint assignment
for x = 1:X
f_init(x,1) = costtotraverse(robot_pos(x,1:2), goals(x,:));
end
f = tree(f_init,bigtop);                % tree for storing f(cost) values
% for each vertex assignment for different robots
unvisit = tree(zeros(X,1),bigtop);      % tree for storing the number of
    ↪ assigned
% unvisited interestpoints for each robot
```

```
q = 1;                                  % current vertex
A = zeros(1,N+M);
A_new = zeros(1,N+M);
A = int2digits(t.get(q),N+M);

while ( ~isgoal(A) )
% choose the vertex q with smallest f value in the open set
small = inf;
for i = 1:length(open)
if small > sum(f.get(open(i)))
q = open(i);
small = sum(f.get(open(i)));
end
end
disp(q);
A = int2digits(t.get(q),N+M);

% Expand q, iterate for each successor q' of q, assigning all waypoints (
    ↪ thermals
% and interest points)
for way = 1:N+M
% check if way has already been assigned to any robot in vertex q
if A(1,way) ~= 0
continue;
end
% assign the waypoint to each robot seperately
for robot = 1:X
%             robot
% calculating new matrix
A_new = A;
A_new(1,way) = robot;

if any(t.find(digits2int(A_new)))
continue;
end
```

```
% add a new vertex q' in the graph which should be ordered
q_prime = t.addnode(q,digits2int(A_new));

% run single robot interleave to calc the cost f of q'
% just for the robot which has the new waypoint
% the function returns the cost plus the 'interest points' not
% visited
ccost = f.get(q);
coordset = [intpts;thermals;goals(robot,:)];
best_path_yet(q_prime,:) = best_path_yet(q,:);
[ccost(robot,1), unvis, best_path_yet(q_prime,robot)] = single_robot_plan(
    ↪ A_new,robot);

% adding unvisited assigned interest points to the tree
u = unvisit.get(q);
u(robot,1) = unvis;
unvisit.addnode(q,u);

% if q' is the goal node add the additional terminal cost
% this is the terminal cost for the case where one thermal is
% assigned to one robot
if isgoal(A_new)
ccost(robot,1) = ccost(robot,1) + sum(unvis)*(sum(T_max) + M*dt_max + 1);
end
f.addnode(q,ccost);

% add q' to open set

end
end
% take q off open sets
open = open(open ~= q);
end

open = 0;
```

```
toc;
save('MTGS_vars.mat');
%% Plotting Results
load('MTGS_vars.mat');
figure;
axis([-2 scale+5 -2 scale+1]);
axis equal;
hold on;
ColOrd = [         0         0    1.0000
0    0.5000         0
1.0000         0         0
0    0.7500    0.7500
0.7500         0    0.7500
0.7500    0.7500         0
0.2500    0.2500    0.2500];
%scatter(intpts(1:N-1,1), intpts(1:N-1,2), 's');
scatter(intpts(:,1), intpts(:,2), 120, 's', 'b');
scatter(robot_pos(:, 1), robot_pos(:, 2), 120, 'o', 'g');
scatter(goals(:, 1), goals(:, 2), 120, 'x', 'r');
scatter(thermals(:,1), thermals(:,2), 120, 'd', 'c');
for robot = 1:X
a = int2digitstpath( best_path_yet(q,robot) )+1;
pt_kind = (a<=N+1 & a>1)+2;
pt_kind (1) = 1;pt_kind (end) = 4;
coordset(end,1:2) = goals(robot,:);
coordset_m = [robot_pos(robot,1:2); coordset];
path_d = coordset_m(a,:);
Col = ColOrd(robot,:);
plot(path_d(:,1), path_d(:,2), 'Color', Col);
way_kind{robot} = pt_kind;
path_rob{robot} = path_d;
end
psi_i = (robot_pos(:,3)-pi)*180/pi;
save ('pathplan.mat','path_rob','psi_i','way_kind');

%% Single_Robot_plan
```

```
function [pathcost, unvis, best_path] = single_robot_plan(combo, rob)

%     global N M X T_max dt_max thermals intpts goals robot_pos

waypoints = [find(combo == rob),N+M+1];
% no of assigned interest points
n_assigned = length(find(waypoints<=N));
% no of assigned thermals
m_assigned = length(waypoints) - n_assigned - 1;
t_med= tree(0,bigmed(1,numel(waypoints)));                    % tree of
    ↪ waypoints, starting at origin of robot
f_med = tree(0,bigmed(1,numel(waypoints)));                   % tree of f-values
    ↪  (numerical) for each vertex
Time = tree(T_max(rob,1),bigmed(1,numel(waypoints)));     % tree that tracks
    ↪ T_max for each vertex
n_cov = tree(0,bigmed(1,numel(waypoints)));                   % no. of covered
    ↪ waypoints at each vertex
openset = zeros(1,bigmed(1,numel(waypoints)));
openset = 1;                                                  % indices of open
    ↪ vertices on tree
q_med = 1;                                                        % current
    ↪ index

% while the goal is not expanded
while (~isgoalsingle(int2digitstpath(t_med.get(q_med)),N+M+1))
% quit algo if path not found
if isempty(openset)
pathcost = -1;
unvis = -1;
%             save('SingleRobot.mat');
return;
end
% get q with the smallest f value from the open set
small_med = inf;
```

```
% iterate for each successor and the goal (+1 is for the goal)
for waypt = 1:length(waypoints)
% check if the waypoint is already in the queue, if yes continue
if any(path == waypoints(waypt))
end
if waypt == length(waypoints)
new_path = [path,N+M+1];
else
new_path = [path,waypoints(waypt)];
end

% calculate cost of transition form last waypoint to new one
% use euclidean distance for now
if path(1,end) == 0
cost = costtotraverse(robot_pos(rob, 1:2), coordset(waypoints(waypt),:));
else
cost = costtotraverse(coordset(path(1,end),:), coordset(waypoints(waypt),:));
end

% add cost to cost of parent vertex
cum_cost_med = cost + f_med.get(q_med);

% check if cost exceeds T_max_i, if yes continue
if Time.get(q_med) < cum_cost_med
continue;
end

% check if waypoint is goal, if yes add additional penalty to the
% cost
n = length(find(new_path<=N))-1;        % no of interest points in new path
m = length(new_path) - n - 1;           % no of thermals in new path
if waypt == length(waypoints)
m = m - 1;                          % since the dest has been counted as a
    ↪ thermal
cum_cost_med = cum_cost_med + (n_assigned - n)*(T_max(rob,1) + m_assigned*
    ↪ dt_max + 1);
```

```
end

% add vertex with the new waypoint in tree
q_prime_med = t_med.addnode(q_med, digits2int(new_path));
%disp(t.tostring)

% add corresponding node for f tree
f_med.addnode(q_med, cum_cost_med);
%disp(f.tostring)

% add corresponding T_max_i to tree
Time.addnode(q_med, T_max(rob,1)+m*dt_max);
%T_prime
%disp(T.tostring)

n_cov.addnode(q_med,n);

% add this new vertex to the open set
openset = [openset, q_prime_med];

end
openset = openset(openset ~= q_med);
end
unvis = n_assigned - n_cov.get(q_med);
pathcost = f_med.get(q_med);
%     pathcost = f_med.get(q_med) - unvis*(T_max(rob,1) + m_assigned*dt_max +
    ↪ 1);
best_path = t_med.get(q_med);
end

end

function [ num2 ] = int2digitstpath( num )
%INT2DIGITS Summary of this function goes here
%   Detailed explanation goes here
```

```
num2 = sscanf( sprintf( '%u', num ), '%1d' )';
if num2(1) == 0
return;
end
num2 = [0,num2];
end

function [ num2 ] = int2digits( num , len)
%INT2DIGITS Summary of this function goes here
%   Detailed explanation goes here

num2 = sscanf( sprintf( '%u', num ), '%1d' )';
num2 = [zeros(1,len-length(num2)),num2];
end

function [ d ] = digits2int( arr )
%DIGITS2INT Summary of this function goes here
%   Detailed explanation goes here
s = arr.*(10.^(length(arr)-1:-1:0));
d = sum(s);
end

function output = isgoal(A)
if isempty(find(A==0,1))
output = 1;
else
output = 0;
end
end

function isgoal = isgoalsingle (a, goal)
if a(1,end) == goal
isgoal = 1;
else
isgoal = 0;
end
```

```
% cost of going for a to b
function cost = costtotraverse(a, b)
cost = norm(a-b);
end
```

A.4 Code for Visitation Sequence Planning with LPA* modification

```
%#codegen
function [  ] = MultiTierGraphSearch_LPAstar_mexble(  )
%MULTITIERGRAPHSEARCH The top level graph search
%   Detailed explanation goes here
%   Author: Muhammad Aneeq uz Zaman

global N M X T_max dt_max robot_pos coordset tpath tkey tg trhs openset ttime
    ↪ allwd leafset
n_asg = 0;
m_asg = 0;
cost = 0;
uvis = 0;
gindex = 0;

S =  coder.load('data4.mat');
M = S.M;
N = S.N;
T_max = S.T_max;
X = S.X;
dt_max = S.dt_max;
goals = S.goals;
intpts = S.intpts;
robot_pos = S.robot_pos;
scale = S.scale;
thermals = S.thermals;
%% Preallocating Data
bigmed = 0;
for a = 1:N+M+1
bigmed = bigmed + factorial(N+M+1)/factorial(N+M+1-a);
end
```

```
maxstruct.path = tree(0,bigmed);maxstruct.key = tree([0;0],bigmed);maxstruct.g
    ↪  = tree(0,bigmed);
maxstruct.rhs = tree(0,bigmed);maxstruct.time = tree(0,bigmed);
    ↪ maxstruct.openset = zeros(1,bigmed);
maxstruct.leafset = zeros(bigmed); maxstruct.best = 0;
bigtop = (X+1)^(N+M);
mystruct = repmat(maxstruct,X,bigtop); tkey = tree(maxstruct.key);tg = tree(
    ↪ maxstruct.g);
trhs = tree(maxstruct.rhs);ttime = tree(maxstruct.time);leafset = zeros(bigmed
    ↪ );
openset = zeros(bigmed);
%%
coordset = [intpts;thermals;goals(1,:)];
open = zeros(1,bigtop);
open = 1;                                   % adding the root to the open set
t = tree(0,bigtop);                         % tree structure for storing waypoint
    ↪  assignment
f_init = zeros(X,1);
% adding vertices pertaining to no waypoint assignment
for x = 1:X
Initialize(x);
coordset(end,1:2) = goals(x,:);
allwd = N+M+1;
[pathcost, ~, ind] = single_robot_plan_init(x);
mystruct(x,1).path = tree(tpath); mystruct(x,1).key = tree(tkey);mystruct(x,1)
    ↪ .g = tree(tg);
mystruct(x,1).rhs = tree(trhs);mystruct(x,1).openset = openset; mystruct(x,1)
    ↪ .time =

f = tree(f_init,bigtop);                    % tree for storing f(cost) values
% for each vertex assignment for different robots

unvisit = tree(zeros(X,1),bigtop);          % tree for storing the number of
    ↪ assigned
% unvisited interestpoints for each robot
q = 1;                                      % current vertex
```

```
A = zeros(1,N+M);
A_new = zeros(1,N+M);
A = int2digits(t.get(q),N+M);

while ( ~isgoal(A) )
% choose the vertex q with smallest f value in the open set
small = inf;
for i = 1:length(open)
cum_cost = sum(f.get(open(i)));
if small > cum_cost
q = open(i);
small = cum_cost;
end
end
disp(q);
A = int2digits(t.get(q),N+M);

% Expand q, iterate for each successor q' of q, assigning all waypoints (
   ↪ thermals
% and interest points)
for way = 1:N+M
% check if way has already been assigned to any robot in vertex q
if A(1,way) ~= 0
continue;
end
% assign the waypoint to each robot seperately
for robot = 1:X
%             robot
% calculating new matrix
A_new = A;
A_new(1,way) = robot;

if any(t.find(digits2int(A_new)))
%                 disp('WTF');
continue;
end
```

```
% add a new vertex q' in the graph which should be ordered
q_prime = t.addnode(q,digits2int(A_new));

% run single robot interleave to calc the cost f of q'
% just for the robot which has the new waypoint
% the function returns the cost plus the 'interest points' not
% visited
ccost = f.get(q);
%% Getting Data from mystruct
tpath = tree(mystruct(robot,q).path);
tkey = tree(mystruct(robot,q).key);
tg = tree(mystruct(robot,q).g);
trhs = tree(mystruct(robot,q).rhs);
openset = mystruct(robot,q).openset;
leafset = mystruct(robot,q).leafset;
ttime = tree(mystruct(robot,q).time);
allwd = [find(A_new == robot),N+M+1];
coordset(end,1:2) = goals(robot,:);
%%
[ccost(robot,1), unvis, ind] = single_robot_plan(way, robot);

% adding unvisited assigned interest points to the tree
u = unvisit.get(q);
u(robot,1) = unvis;
unvisit.addnode(q,u);

% if q' is the goal node add the additional terminal cost
% this is the terminal cost for the case where one thermal is
% assigned to one robot
if isgoal(A_new)
ccost(robot,1) = ccost(robot,1) + sum(unvis)*(sum(T_max) + M*dt_max + 1);
end
f.addnode(q,ccost);
%% Storing Data into mystruct
for r = 1:X
```

```
if r == robot

end
end
%%
% add q' to open set
open = [open,q_prime];
end
end
% take q off open sets
open = open(open ~= q);
end
%% Plotting Results
figure;
axis([-1 scale+1 -1 scale+1]);
hold on;
%scatter(intpts(1:N-1,1), intpts(1:N-1,2), 's');
scatter(intpts(:,1), intpts(:,2), 's');
scatter(thermals(:,1), thermals(:,2), 'd');
scatter(robot_pos(:, 1), robot_pos(:, 2), 'o');
scatter(goals(:, 1), goals(:, 2), 'x');
for robot = 1:X
tree_path = mystruct(robot,q).path;
a = int2digitstpath(tree_path.get(mystruct(robot,q).best))+1;
coordset(end,1:2) = goals(robot,:);
coordset_m = [robot_pos(robot,1:2); coordset];
path_d = coordset_m(a,:);
ColOrd = get(gca,'ColorOrder');
[j,robot] = size(ColOrd);
if ColRow == 0
ColRow = j;
end
Col = ColOrd(ColRow,:);
plot(path_d(:,1), path_d(:,2), 'Color', Col);
end
%%
```

```
end

function [pathcost, unvis, ind] = single_robot_plan_init(r)

global N M  robot allwd n_asg m_asg cost uvis gindex leafset
% tpath contains the tree of paths from prev iteration
% tvalues values contains all the values of the tree
% open set is the open set in that tree
n_asg = length(find(allwd<=N));
m_asg = length(find(allwd>N & allwd<=M+N));
cost = inf;
uvis = 0;
gindex = 0;
robot = r;

% this is the part inside the "if edge costs change" loop
% the cost of traversal to the new vertex and the penalty costs
% have changed

% for all directed edges with changed edge costs, do
%   update edge cost
%   UpdateVertex(u)
ComputeShortestPath();

pathcost = cost;
unvis = uvis;
ind = gindex;

end

function [pathcost, unvis, ind] = single_robot_plan(new_way, r)

global N M robot tpath allwd n_asg m_asg cost uvis gindex leafset openset
% tpath contains the tree of paths from prev iteration
% tvalues values contains all the values of the tree
% open set is the open set in that tree
```

```
allwd = allwd(allwd ~= 0);
n_asg = length(find(allwd<=N));
m_asg = length(find(allwd>N & allwd<=M+N));
cost = inf;
uvis = 0;
gindex = 0;
robot = r;

% this is the part inside the "if edge costs change" loop
% the cost of traversal to the new vertex and the penalty costs
% have changed

% for all directed edges with changed edge costs, do
%   update edge cost
%   UpdateVertex(u)

% only update nodes in the openset and leafset
for j = 1:length(leafset)
path = int2digitstpath(tpath.get(leafset(j)));
if any(path == new_way)
UpdateVertex(leafset(j));
elseif any(path == N+M+1)
end
end

ComputeShortestPath()

pathcost = cost;
unvis = uvis;
ind = gindex;

end

function key = CalculateKey(s)
global tg trhs
% Calculate key
```

```
if s == 0
key = [inf;inf];
else
key = [min(tg.get(s),trhs.get(s)) + Heuristic; min(tg.get(s),trhs.get(s))];
end
end

function Initialize(rrobot)
global N M tpath tkey tg trhs ttime openset T_max allwd leafset

% open is null

% Configure all rhs and g values to be inf, might have to deal with it in a
% different manner
tpath = tree(0);
ttime = tree(T_max(rrobot,1));
tg = tree(inf);
% rhs(start) = 0
trhs = tree(0);
allwd = N+M+1;

% insert goal into tree while calculating key
leafset = 1;
openset = 1;
tkey = tree([0;0]);
leafset = [];
end

function UpdateVertex(u)
global N M coordset robot_pos tpath dt_max T_max robot tkey tg trhs openset
    ↪ ttime allwd n_asg m_asg cost uvis gindex
% if u != start, rhs(u) = min(cost(u,s_p) + g(s_p)) + terminal cost, where
% s_p is pred(u), remember cost(u,s_p)=inf if the path cost exceeds T_max + dt
if u ~= 1
if any(allwd == path(end))
if path(end-1) == 0
```

```
rhs = tg.get(trhs.getparent(u)) + costtotraverse(coordset(path(end),:),
    ↪ robot_pos(robot, 1:2));
else
rhs = tg.get(trhs.getparent(u)) + costtotraverse(coordset(path(end),:),
    ↪ coordset(path(end-1),:));
end
if rhs > ttime.get(ttime.getparent(u))
rhs = inf;
end
else
rhs = inf;
end
if isgoalsingle (path, N+M+1)
n = length(find(path<=N))-1;
rhs = rhs + (n_asg - n)*(T_max(robot,1) + m_asg*dt_max + 1);
if rhs < cost
cost = rhs;
uvis = n_asg - n;
gindex = u;
end
end
trhs = trhs.set(u, rhs);
end

% if u in open, remove from open
openset = openset(openset ~= u);

% if g(u) != rhs(u), insert in open with new calculated key, remember
% g(u)=inf if its a new vertex
if tg.get(u) ~= trhs.get(u)
key = CalculateKey(u);
tkey = tkey.set(u, key);

% insert u into openset according to lexial ordering
flag = 0;
if isempty(openset)
```

```
openset = u;
elseif lexiless(key,tkey.get(openset(1)))
openset = [u;openset];
else
for i = 2:length(openset)
if lexiless(key,tkey.get(openset(i)))
openset = [openset(1:i-1);u;openset(i:end)];
flag = 1;
break;
end
end
if flag == 0
openset = [openset;u];
end
end
end
end

function ComputeShortestPath()
global N M dt_max T_max robot tpath tkey tg trhs openset ttime gindex leafset
% loop while top key in open set < key of goal or rhs(goal)!=g(goal)
if gindex == 0
rhsu = inf; gu = inf;
else
rhsu = trhs.get(gindex);gu = tg.get(gindex);
end
while lexiless(tkey.get(openset(1)),CalculateKey(gindex)) || rhsu ~= gu

% u = pop the top vertex from open set
u = openset(1);
openset = openset(2:end);

% if g(u) > rhs(u), g(u) = rhs(u), for all succ(u) updatevertex
tg = tg.set(u,trhs.get(u));
% if u has already been expanded, update all its successors
if ~tg.isleaf(u)
```

```
child = t.getchildren(u);
for i = 1:length(child)
UpdateVertex(child(i));
end
% if u is leaf, calculate succ(u), initialize g, rhs and other tree vertices
   ↪ as inf and updatevertex
elseif ~isgoalsingle(int2digitstpath(tpath.get(u)),N+M+1)
leafset = leafset(leafset~=u);
for rway = 1:N+M+1
path = int2digitstpath(tpath.get(u));
if any(path == rway)
continue;
end
new_path = [path,rway];
index = tpath.addnode(u,digits2int(new_path));
m = length(find(new_path>N & new_path<=M+N));
ttime.addnode(u,T_max(robot,1) + m*dt_max);
tkey.addnode(u,[0;0]);
tg.addnode(u,inf);
trhs.addnode(u,inf);
% the new node is added in leafset
leafset = [leafset,index];
% update
UpdateVertex(index);
end
end
else
% else, g(u) = inf, for all succ(u) and u updatevertex
tg = tg.set(u,inf);
% if u has already been expanded, update all its successors
% if u is leaf, calculate succ(u), initialize g, rhs and other tree vertices
   ↪ as inf and updatevertex
if ~tg.isleaf(u)
child = tg.getchildren(u);
for i = 1:length(child)
UpdateVertex(child(i));
```

```
end
elseif isgoalsingle(int2digitstpath(tpath.get(u)),N+M+1)
UpdateVertex(u);
else
leafset = leafset(leafset~=u);
for kway = 1:N+M+1
path = int2digitstpath(tpath.get(u));
if any(path == kway)
continue;
end
new_path = [path,kway];
index = tpath.addnode(u,digits2int(new_path));
m = length(find(new_path>N && new_path<=M+N));
ttime.addnode(u,T_max(robot,1) + m*dt_max);
tkey.addnode(u,[0;0]);
tg.addnode(u,inf);
trhs.addnode(u,inf);
% the new node is added in leafset
leafset = [leafset,index];
% update
UpdateVertex(index);
end
end
end
if gindex ~= 0
rhsu = trhs.get(gindex);
gu = tg.get(gindex);
end
if isempty(openset)
break;
end
end

function [ num2 ] = int2digits( num , len)
%INT2DIGITS Summary of this function goes here
%   Detailed explanation goes here
```

```
buff = num2str(num);
temp = zeros(1,2*length(buff));
temp([1:2:2*length(buff)])=buff;
num2=str2num(char(temp));
num2 = [zeros(1,len-length(num2)),num2];
end

function [ num2 ] = int2digitstpath( num )
%INT2DIGITS Summary of this function goes here
%   Detailed explanation goes here
buff = num2str(num);
temp = zeros(1,2*length(buff));
temp([1:2:2*length(buff)])=buff;
num2=str2num(char(temp));
num2 = [0,num2];
end

function [ d ] = digits2int( arr )
%DIGITS2INT Summary of this function goes here
%   Detailed explanation goes here
s = arr.*(10.^(length(arr)-1:-1:0));
d = sum(s);
end

function output = searchtree(t,A_new,N,M)
depth = 1;
queue = zeros(1,N+M);
node_no = 1;
node_no_parent = 0;
output = 0;
% children = zeros(1,N+M);
while (node_no <= t.nnodes)
A = int2digits(t.get(node_no),N+M);
if all(A_new == A)
output = 1;
return;
```

```
elseif ~t.isleaf(node_no) && all(A(A~=0) == A_new(A~=0))
depth = depth + 1;
queue(depth) = 1;
node_no_parent = node_no;
children = t.getchildren(node_no_parent);
node_no = children(1);
else                                        % if not even partial match or
    ↪ isleaf
queue(depth) = queue(depth) + 1;         % go to next node at same depth
children = t.getchildren(node_no_parent);
while queue(depth) > numel(children)        % if node larger than maximum
    ↪ branch no, go up one step and repeat
if depth == 1
return;
end
queue(depth)=0;
depth = depth - 1;
queue(depth) = queue(depth) + 1;
node_no_parent = t.getparent(node_no_parent);
children = t.getchildren(node_no_parent);
end
node_no = children(queue(depth));
end
end
end

function output = isgoal(A)
if length(find(A==0)) == 0
output = 1;
else
output = 0;
end
end

function isgoal = isgoalsingle (a, goal)
if a(1,end) == goal
```

```
isgoal = 1;
else
isgoal = 0;
end
end

function cost = costtotraverse(a, b)
cost = norm(a-b);
end

function h = Heuristic ()
h = 0;
end

function l = lexiless (key1, key2)
if key1(1)<key2(1) || (key1(1)==key2(1) && key1(2)<key2(2))
l = 1;
else
l = 0;
end
end
```

REFERENCES

[1] "Helios crash probed," 2007. [Online]. Available: http://www.nasa.gov/missions/research/helios.html/

[2] P. B. MacCready, "Optimum airspeed selector," *Journal of Soaring*, 1958.

[3] H. Reichmann, "Cross-country soaring," *Soaring Society of America*, pp. 91–92, 1993.

[4] J. Wharingtoni, Ph.D. dissertation.

[5] M. J. Allen and V. Lin, "Guidance and control of an autonomous soaring uav," NASA, Tech. Rep. TM-2007-214611/REV1, April 2007.

[6] K. Cheng and J. W. Langelaan, "Guided exploration for coordinated autonomous soaring flight," in *AIAA Guidance, Navigation, and Control Conference*, January 2014.

[7] K. Andersson, I. Kaminer, V. Dobrokhodov, and V. Cichella, "Thermal centering control for autonomous soaring; stability analysis and flight test results," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 3, pp. 963–975, 2012.

[8] W. B. Kagabo and J. R. Kolodziej, "Trajectory determination for energy efficient autonomous soaring," in *American Control Conference (ACC), 2011.* IEEE, 2011, pp. 4655–4660.

[9] A. T. Klesh, P. T. Kabamba, and A. R. Girard, "Optimal cooperative thermalling of unmanned aerial vehicles," in *Optimization and Cooperative Control Strategies.* Springer, 2009, pp. 355–369.

[10] J. Le Ny, E. Frazzoli, and E. Feron, "The curvature-constrained traveling salesman problem for high point densities," in *Decision and Control, 2007 46th IEEE Conference on.* IEEE, 2007, pp. 5985–5990.

[11] J. Le Ny and E. Feron, "An approximation algorithm for the curvatureconstrained traveling salesman problem," in *Proceedings of the 43rd Annual Allerton Conference on Communications, Control and Computing*, 2005, pp. 620–9.

[12] 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*, pp. 497–516, 1957.

[13] S. E. Butt and D. M. Ryan, "An optimal solution procedure for the multiple tour maximum collection problem using column generation," *Computers & Operations Research*, vol. 26, no. 4, pp. 427–441, 1999.

[14] S. Boussier, D. Feillet, and M. Gendreau, "An exact algorithm for team orienteering problems," *4or*, vol. 5, no. 3, pp. 211–230, 2007.

[15] H. Tang and E. Miller-Hooks, "A tabu search heuristic for the team orienteering problem," *Computers & Operations Research*, vol. 32, no. 6, pp. 1379–1407, 2005.

[16] C. Archetti, A. Hertz, and M. G. Speranza, "Metaheuristics for the team orienteering problem," *Journal of Heuristics*, vol. 13, no. 1, pp. 49–76, 2007.

[17] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden, "Metaheuristics for tourist trip planning," in *Metaheuristics in the service industry*. Springer, 2009, pp. 15–31.

[18] D. Thakur, M. Likhachev, J. Keller, V. Kumar, V. Dobrokhodov, K. Jones, J. Wurz, and I. Kaminer, "Planning for opportunistic surveillance with multiple robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 5750–5757.

[19] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93.

[20] V. H. Schulz, H. G. Bock, and R. W. Longman, "Optimal path planning for satellite mounted robot manipulators," *NASA STI/Recon Technical Report A*, vol. 95, p. 81363, 1993.

[21] K. P. Bollino, L. R. Lewis, P. Sekhavat, and I. M. Ross, "Pseudospectral optimal control: a clear road for autonomous intelligent path planning," in *Proceedings of the AIAA InfoTech at Aerospace Conference and Exhibit*, 2007, pp. 1228–1241.

[22] I. M. Ross and F. Fahroo, "Pseudospectral methods for optimal motion planning of differentially flat systems," *Automatic Control, IEEE Transactions on*, vol. 49, no. 8, pp. 1410–1413, 2004.

[23] T. Inanc, K. Misovec, and R. M. Murray, "Nonlinear trajectory generation for unmanned air vehicles with multiple radars," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 4. IEEE, 2004, pp. 3817–3822.

[24] Z. Hao, K. Fujimoto, and Y. Hayakawa, "Optimal trajectory generation for nonlinear systems based on double generating functions," in *American Control Conference (ACC), 2013*. IEEE, 2013, pp. 6382–6387.

[25] S. M. LaValle, "Rapidly-exploring random trees a ew tool for path planning," 1998.

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

[27] O. A. Yakimenko, "Direct method for rapid prototyping of near-optimal aircraft trajectories," *Journal of Guidance, Control, and Dynamics*, vol. 23, no. 5, pp. 865–875, 2000.

[28] R. Choe, V. Cichella, E. Xargay, N. Hovakimyan, A. C. Trujillo, and I. Kaminer, "A trajectory-generation framework for time-critical cooperative missions," *AIAA Infotech@ Aerospace, Boston, MA. AIAA*, vol. 4582, 2013.

[29] I.-M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem," *European Journal of Operational Research*, vol. 88, no. 3, pp. 475–489, 1996.

[30] L. Ke, C. Archetti, and Z. Feng, "Ants can solve the team orienteering problem," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 648–665, 2008.

[31] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden, "Metaheuristics for tourist trip planning," in *Metaheuristics in the service industry*. Springer, 2009, pp. 15–31.

[32] W. Souffriau, P. Vansteenwegen, G. V. Berghe, and D. Oudheusden, "A greedy randomised adaptive search procedure for the team orienteering problem," in *EU/MEeting*, 2008, pp. 23–24.

[33] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, 1995.

[34] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *Journal of Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

[35] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a," *Artificial Intelligence*, vol. 155, no. 1, pp. 93–146, 2004.

[36] P. Morin and C. Samson, "Trajectory tracking for nonholonomic vehicles," in *Robot Motion and Control*. Springer, 2006, pp. 3–23.

[37] P. Bhatta, "Nonlinear stability and control of gliding vehicles," Ph.D. dissertation, Princeton University, 2004.

[38] E. Xargay, I. Kaminer, A. Pascoal, N. Hovakimyan, V. Dobrokhodov, V. Cichella, A. Aguiar, and R. Ghabcheloo, "Time-critical cooperative path following of multiple uavs over time-varying networks," DTIC Document, Tech. Rep., 2011.

[39] R. T. Farouki and T. Sakkalis, "Pythagorean hodographs," *IBM Journal of Research and Development*, vol. 34, no. 5, pp. 736–752, 1990.

[40] G. J. Ruijgrok, *Elements of airplane performance.* Delft university press, 1990.

[41] R. T. Farouki, *Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable.* Springer, 2008.