SAFE TRAJECTORY PLANNING TECHNIQUES FOR AUTONOMOUS AIR VEHICLES

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Waseem Ahmed Kamal MSc (UMIST)
Department of Engineering
University of Leicester

November 2005

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

This dissertation explores optimal path planning techniques for safe navigation of autonomous air vehicles. Finding trajectories for multiple vehicles moving in a dynamic environment while satisfying all constraints is a challenging problem. No single technique can be used independently for this problem. Different techniques for real time path planning have been developed and compared. First the problem was addressed using optimal control theory. Equations governing necessary conditions were derived for a combined objective of terrain avoidance, radar avoidance and minimum path length. Using these equations, analytical solutions for radar risk minimization problem have been derived. A gradient method was used to get an optimal solution for different radar geometries. Mixed integer linear programming (MILP) formulation employing branch and bound techniques was investigated for both single and multiple autonomous air vehicle trajectory planning. A novel real-time receding horizon approach using MILP has been proposed that uses binary variables to model the soft and hard constraints for radar zones. A three dimensional probabilistic approach for the path planning unmanned air vehicles(UAVs) has been considered as well. For this approach a probabilistic cost function has been developed that accounts the various factors of fuel, collision, crash to ground objects etc. The novelty of the algorithm relies in its ability to be used in real time due to very low computational load in spite of the fact that it finds a path in three dimensions. The paths are locally optimal and are feasible for the UAV to follow. For graph-based global optimality, a software has been developed that includes extra subroutines to modify the already implemented Voronoi code in order to remove the infinity and far away nodes and also includes the corner points of the operational area. This software has been employed to find the Length Constraint Least Risk (LCLR) paths and also different techniques were compared. Although the aim of the research was to explore and develop different real time techniques for the safe navigation of UAVs, the thesis also concludes by considering the cooperative control of a team of UAVs and proposes an architecture for this purpose.

Acknowledgement

First of all I thank Almighty ALLAH for blessing me with ability, courage and strength to complete my studies.

I would like to express my gratitude to Professor Ian Postlethwaite and Dr Da-Wei Gu for their invaluable supervision and help throughout my PhD studies. The research work was supported by TROSS Scholarship Scheme of the Government of Pakistan and the UK Engineering and Physical Sciences Research Council and BAE Systems. Support from all these organisations is gratefully acknowledged. The later stages of my studies benefited from involvement in the EPSRC/BAE Systems Integrated Project in Aeronautical Engineering. I am grateful for this opportunity.

I would also like to thank my research colleagues Dr Muhammed Khalid Khan and Dr Li Qun Yao from the University of Leicester for not only their invaluable help throughout this work but also for their friendship and the working environment they provided during the times spent at university.

I wish to thank all my colleagues in the Control and Instrumentation Group at Department of Engineering, University of Leicester for their friendship and lively environment they provided. Dr Mathew Turner, Dr Sarah Gatley, Dr Guido Herrmann, Dr Yoonsoo Kim, Dr Jongrae Kim are some of the people whom I cannot express how grateful I am. I am grateful to my parents for their prayers, guidance and support throughout this journey. Also my wife and children Talha, Montaha and Maheen gave me support and inspired me throughout during my stay at Leicester. I could never have done this without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The capabilities and roles of Unmanned Air Vehicles (UAVs) are evolving and new concepts are required for their control [19]. Today's UAVs typically require several ground based operators per aircraft but future UAVs will be designed to make tactical decisions autonomously and will be integrated into coordinated teams to achieve high level goals, thereby allowing one operator to control a group of vehicles. New methods in planning and execution are required to coordinate the operation of a fleet of UAVs. An overall control system architecture must be developed that can perform optimal coordination of the vehicles, evaluate the overall system performance in real time and quickly reconfigure to account for changes in the environment. Cooperative control of multiple UAVs is a subject to which interest is increasing in the research community. Research has focused primarily on three areas: UAV formation flight, cooperative path planning (e.g. rendezvous), and resource allocation (e.g. target assignment).

1.1.1 Areas of Application

The use of UAVs is vast and development in many different areas of applications is possible. Examples include:

Telecommunications A UAV could be used to provide live video for news

and sports events. An internet link could also be provided to include the same content on the web.

Weather and atmospheric monitoring UAVs could be used to obtain temporary weather information and they could be launched at any time when needed. Also UAVs could be used for atmospheric monitoring of pollution.

Emergency communication node In the event of an emergency, UAVs could be used to act as communication relays to assist in the coordination of relief efforts. This could be accomplished even in the midst of widespread destruction since UAVs could be sent from a distant location.

Security and border patrol UAVs could be used in conjunction with ground personnel to give real time information on the location of trespassers. Also, this same information could be used to identify a suspect.

Military There are many military applications such as the suppression of enemy air defences and there will be an emphasis on this area in the thesis.

1.1.2 Advantages of UAVs over Manned Aircraft

The potential advantages of UAVs over their manned counterparts are substantial and provide the motivation to further developments.

Manoeuvrability Due to the absence of a pilot, the forces that a UAV can sustain are limited only by the material made to construct it. This can significantly increase the manoeuvrability of a UAV relative to its manned counterpart. Instead of the usual $9g$ limit for a manned aircraft, UAVs could potentially be designed to achieve the $40g$ to $50g$ performance of a modern missile.

Low human risk UAVs are ideal for situations in which the risk to the pilot of a manned aircraft would be unacceptably high. One example of this situation is the SEAD (suppression of enemy air defences) mission which

involves attacking well defended areas containing a high concentration of enemy surface to air missiles (SAMs). Such missions are extremely risky as the chance of being attacked by these defensive forces is high. Therefore UAVs are well suited for this type of mission since there is no risk to any pilot. After UAVs have made the initial attack manned aircraft could come in for subsequent strikes at the target.

Low cost UAVs could cost significantly less than manned aircraft. Most of these savings will come from there being no need for highly trained pilots.

Weight savings Because the UAV contains no pilot, it also does not need to contain any of the support equipment that a pilot requires such as an ejection system or instrumentation. This weight saving can be dedicated to improving the performance of the UAV or to increasing its payload of sensors or weapons.

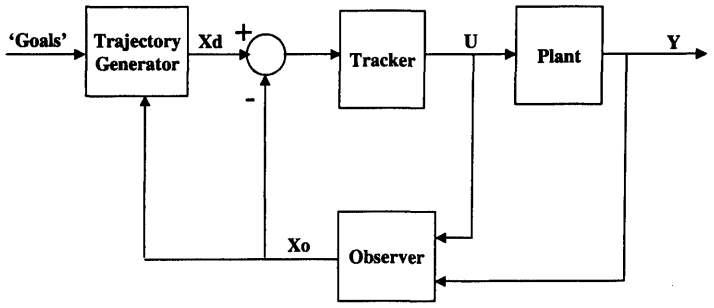


Figure 1.1: A two degree of freedom motion controller

1.2 Controlling Autonomous UAVs

Controlling a vehicle is not a simple task, because the environment as well as the dynamics of the vehicle must be taken into account while planning its motion. A typical two degree of freedom motion controller is given in Figure 1.1. The motion controller consists of two parts: a trajectory generator (high level control) and a trajectory tracker (low level control). Higher level control or intelligent motion planning resides at the upper level of the architecture and

is an essential part of an autonomous vehicle. A set of goals is fed into a trajectory generator which also receives information about the system states from the system observer along with information about the environment. The error between planned and actual trajectories is passed on to the trajectory tracker which generates a corrective action. The performance of these main blocks depends on the performance of sensors, estimation algorithms and communication. The vehicle can be a free flying robot, spacecraft, aircraft, helicopter, mobile robot or any autonomous guided vehicle. The aforementioned objects are quite different with respect to their motion abilities and ranges of environmental perception. For example, fixed wing aircraft cannot fly with low velocities and their turning radii are quite large compared to mobile robots which can stop almost immediately and they can relatively easily reorient themselves. Moreover, in direct contrast to mobile robots, aircraft can detect obstacles at long distances and the number of obstacles in their environment is usually relatively small. This variety of vehicles makes the development of a general motion planning algorithm difficult. Before describing motion planning in detail in the next section, the other key technologies, which are necessary to achieve the goal of increased autonomy of unmanned vehicles to perform a mission, will be described.

1.2.1 Sensor Technology

To enable autonomy of unmanned vehicles, each vehicle must be equipped with sensors that can provide the necessary information for the controller and the planner to make decisions and operate in the environment. Dynamic planning cannot be done without the data from the on-board sensors that detect not only objects in the environment, but also the states of the vehicle itself. Examples of sensors used to measure the states of a vehicle are optical encoders, tachometers, Inertial Measurement Units (IMUs), Global Positioning Systems (GPSs). A range of devices can be used to sense the environment in which a vehicle operates. Ultrasonic and infrared sensors are proximity sensors which can detect a nearby obstacle. A more efficient proximity sensor is a laser range finder. An optical device like a CCD camera can be used to detect objects

in the environment, but it requires image processing to interpret the recorded images. An advantage of optical devices is that they can not only detect the position of an object but can also identify the type of the object and its shape.

1.2.2 Estimation Algorithms

The raw data detected from sensors is often unusable by the controller and the planner of each vehicle. The data must first be processed by an estimation algorithm. Estimation of the current position of the vehicle alone is not a trivial problem, especially for indoor applications where GPS cannot be used. One approach for this estimation problem is Markov localization proposed by Fox et al. [35]. A similar method based on particle filters is Monte Carlo localization proposed by Thrun et al. [101]. Both of these approaches provide an estimate of the current position of the vehicle using on-board proximity sensors. In order to compute future actions, the planner of each vehicle also requires information about the environment which can be represented as a 2D or 3D map of the environment. In uncertain environments, the estimator must be able to generate and update the map in real-time while the vehicle is operating in the field. A more interesting estimation problem is how to fuse all the sensor data detected by multiple vehicles in the field and use it to build a map of the environment. The vehicle estimator must also be able to identify and monitor failures that might occur during the mission using the sensor data. A failure can result in reduced capabilities or limitations on the vehicle performance. In addition, the estimator must be capable of predicting the future states and capabilities of the vehicle. The planner requires this information in order to adapt the action plans according to the failure and to insure the completion of the mission objectives.

1.2.3 Communication Technology

Communication is essential for the implementation of autonomous vehicles, especially in a distributed planning architecture where each vehicle plans its own actions. It is the basis for the interactions among the vehicles and also with hu-

man operators. Vehicles cannot cooperate and coordinate their actions if they cannot communicate and share information. The structure or the pattern of the flow of the exchanged information among the vehicles is called a communication network. The number of research efforts in networking has skyrocketed recently with the popularity of the internet. However, more research is needed to improve robustness, reliability, and security.

1.2.4 Robust Adaptive Control Systems

An autonomous vehicle must have an on-board control system that receives the commands from the planner and controls the actuators in response to the input commands. The control system must be robust to the noise, disturbances and un-modelled dynamics in the system. A typical controller is designed using a model of the system which is free of noise and disturbances. This situation is unlikely to exist in real-world systems. To handle this problem, the controller must be designed with the consideration that these uncertainties exist in the system. The design must not only consider the performance of the control system, but also the robustness and stability of the system to the unknown (or partially known) uncertainties. Another desirable feature of the controller is the ability to adapt the characteristics of the controller in order to maintain the desired behaviour and response of the system in the event of failure or an unpredicted situation. There is an extensive literature on adaptive control systems [103], [50], [87]. Adaptive control has been an active research subject especially in aerial vehicle applications. Aircraft operate over a wide range of speeds and altitudes, and their dynamics are complex and nonlinear. Typical flight control designs are based on linearized models of the aircraft at a certain operating point. To control an aeroplane at different operating points, gain scheduling can be used to select the controller gains from a pre-computed lookup table based on the current operating point. However, this approach may involve many different operation points, and the unpredicted changes in the system may lead to instability of the system or a loss of performance. Thus, to improve the robustness and performance of autonomous systems, unmanned vehicles should be equipped with robust adaptive control systems.

1.3 Motion Planning

Usually the task of motion planning is divided into two stages: path planning and trajectory planning. Path planning has an environmental description as an input along with start and goal points. It generates a geometrical path which the vehicle needs to follow. This path, in sequence, is then time-parameterized by a trajectory planner. The trajectory planner, based on the dynamic characteristics of the vehicle, determines the time-dependent characteristics like positions, velocities and accelerations.

Motion planning is an important issue in the development of autonomous air vehicles. Without a pilot, computer algorithms must be developed that generate a flight path in real time. This is a challenging problem for several reasons. For military applications, the algorithm must compute a stealthy path, steering the aircraft's radar signature around known enemy radar locations. In addition, it is often desirable to have trajectories that correspond to an optimal performance according to some criteria, e.g. minimum energy consumption and minimum time usage. An important role of the trajectory generator is to provide the tracking controller with feasible reference trajectories that comply with constraints inherent to the system or externally imposed such as system dynamics, path and actuator constraints, and end-point conditions. Trajectories that do not comply with a system's dynamics and constraints have a small likelihood of implementation, since they might place demands on the controller beyond its limitations. The algorithm must allow for coordination among multiple UAVs and it must run in real time, since enemy threats can change during a mission, forcing a path replan. It must be memory and computationally efficient, since it will be run on an airborne processor.

Path planners are generally divided into local and global and do not usually take into account the dynamics of the vehicle. The former group works in on-line mode while the later both on-line and off-line modes. Global path planning requires all information before any motion of a vehicle is performed. When global information is known in advance but it is neither perfect nor

predictable, then there is a tendency to design so called local path planners [9, 27]. Although such planners lead to the loss of path optimality but their actions are still focused on target reaching while avoiding threats. The known approaches to path planning might be divided into four categories: deterministic, stochastic, learning-based, and reflexive (behavioural).

Deterministic approaches are used most often in global path planners. Two such approaches are Voronoi diagrams [99] and the visibility graphs [47]. A clear disadvantage of all global motion planners is the replanning needed each time the environment changes. This happens frequently with moving threats. A local path planner does not suffer from this disadvantage.

When moveable obstacles are considered and their characteristics are known in advance, they may be incorporated in the global path/trajectory planner by adding to spatial variables a time variable and solving the task problem in such an augmented space [59]. More often, to represent dynamic environments, i.e. with moveable obstacles and unknown or partly unknown characteristics, a stochastic approach is used [14, 108]. In stochastic approaches, planners usually work as local planners and the decision to move a vehicle in a particular direction is made on the basis of minimizing a risk probability function. The popular probabilistic approaches to path planning are *randomized kinodynamics* [45, 62] and probabilistic road map (PRM) approaches [51, 60, 61] etc.

Machine learning techniques are applied in unknown environments [46]. These techniques are based on the assumption that the world is not known in advance, but knowledge about it may be acquired. There are some regularities in the environment and the vehicle can gain knowledge about them while navigating.

All methods mentioned above build a model of the world and plan actions consistent with this model. Also, there are other methods which do not employ any world model building procedures and they act behaviourally (reflexively). Instead of building models, reflexive planners act quickly to avoid the nearest obstacles. Unfortunately they do not exhibit intelligent behaviour, so usually they are implemented as a low level control system controlled by higher levels.

Such a hierarchical architecture organized in layers has been proposed in [11]. The problem of trajectory planning has been studied for decades in a variety of different contexts. The aerospace path planning problems of the 1960's were solved largely through the application of the calculus of variations, itself invented hundred of years earlier, e.g. [13, 57, 64]. The development of industrial robotic manipulators in the 1970s and 1980s encouraged researchers to study new path planning methods largely motivated by collision avoidance. More recently the focus in robotics has shifted to path planning for autonomous mobile robots. Although most of this work concerns wheeled vehicles, e.g. [20], path planning algorithms have been developed for UAVs as well as underwater robots.

A mixed integer linear programming (MILP) formulation of the autonomous vehicle path planning problem is proposed in Richards et al. [84, 85], Richards and How [86], Schouwenaars et al. [89]. The MILP formulation of the cost, obstacle avoidance and collision avoidance were considered in [89] and two solution strategies proposed: a receding horizon strategy and a fixed arrival time approach. It was shown that receding horizon strategies, while computationally more attractive than strategies aimed at computing complete trajectories a priori can lead the system to unsafe conditions where the MILP is no longer feasible. In [84, 86], constraints to enforce upper limits to both velocity and acceleration were further included and the MILP formulation was extended to consider multiple waypoint path planning in which each vehicle is required to visit a set of points in an order decided during the optimization for minimum completion time. The approach proposed in [84] decomposes this large problem into assignment and trajectory problems, while capturing key features of the coupling between them. This allows the control architecture to solve an allocation problem first to determine a sequence of waypoints for each vehicle to visit and then concentrate on designing paths to visit these pre-assigned waypoints. Since the assignment is based on a reasonable estimate of the trajectories, this separation causes a minimal degradation in the overall performance but still it cannot be used in real time.

1.4 Assumptions

The algorithms developed in the thesis are based on the following assumptions:

- The UAVs have limitations on speed, acceleration and duration of flight.
- The environment in which UAV operates is assumed to consists of a number of surface to air missiles (SAMs).
- A SAM fire unit is assumed to consist of one radar used for both surveillance and tracking and a number of missile launchers.
- The fire units are integrated to form an integrated air defence system. The radar can be switched off, if an incoming anti-radiation missile is detected.
- Some of the surveillance radars are on alert using continuous transmission, while the other air defence units remain silent and serve as pop-up threats.
- Each UAV is equipped with suitable sensors:
  - An inertial measurement unit (IMU), which can be used to measure the system states.
  - a proximity sensor (e.g. infrared sensor), which can be used to sense the environment in which a vehicle operates.
  - a radar warning receiver (RWR), which gives a bearing on an emitting radar with certain accuracy. The range of RWR mounted onboard a UAV is assumed slightly higher than the range of the of the radar as defined in the design challenge of the GARTEUR Flight Mechanics Action Group 14 [36].
- Each vehicle contains a guidance system that is capable of guiding the vehicle along the path generated by the planner.
- Each vehicle has a perfect controller to activate the on-board payload according to the planned actions.

- Each vehicle has communication devices to communicate and share information with the other vehicles and the ground station. The bandwidth and quality of those communication channels may be limited.
- Each vehicle has an estimator which merges data provided by the onboard sensors and shared information from the other vehicles in order to estimate the necessary states of the system needed by the planner.

1.5 Contributions and Structure of the Thesis

Most of the previous work in path planning to the targets that avoids threats is either computationally extensive which does not allow re-planning when the environment changes or gives trajectories that are not feasible within the dynamic constraints of the UAVs. There are also factors like terrain avoidance, collision avoidance, fuel consumption etc that need to considered while planning paths. The development of a comprehensive theory of cooperative control for UAVs is a vast problem, beyond the scope of a single dissertation. A realistic, yet challenging problem statement for this dissertation is as follows:

The objective is to develop real time approaches to a problem of UAVs path planning that increase the UAVs survivability, while meeting mission specifications, by decreasing susceptibility to threats containing multiple radars, collocated with launch sites for radar-guided SAMs and also seeking fundamental truths concerning autonomous air vehicle and their cooperative control.

On the basis of the research carried out, four papers have been published [43], [52], [54], [53]. The main contribution of the thesis are:

- An optimal control formulation for safe navigation considering multiple objectives is proposed and necessary conditions governing the optimal solution are derived. The overall objective consists of the sub-objectives of terrain avoidance, radar avoidance and minimum path length. Furthermore the analytical solution for the optimal trajectory and cost function for single radar risk minimization has been derived and a gradient method is proposed to get an optimal numerical solution for different radar geometries.

- A novel real time receding horizon control technique using mixed integer linear programming is proposed that solves the problem of infeasibility encountered by using finite horizon, and its efficiency is demonstrated through example scenarios. The strategy uses the novel dynamical soft boundary concept for modelling of the radar zones and collision avoidance constraints.

- A three dimensional probabilistic approach for unmanned air vehicles (UAVs) path planning is presented. For this approach, a probabilistic cost function is developed that accounts for various factors in the model like limited fuel, collisions, crashes with ground objects etc. The novelty of the algorithm relies in its real time applicability due to a very low computational load in spite of the fact that it finds a path in three dimensions. The paths are locally optimal and are feasible for the UAV to follow by keeping the turn angle within certain maximum limits.

- Software is developed (see Appendix A) that includes extra subroutines to modify the available Voronoi code in order to remove the infinity and far away nodes and also includes the corner points of the operational area. Local optimizations are used to best tune the path obtained from this software. The software is used to find the Length Constraint Least Risk (LCLR) paths.

- All the above techniques are compared for different parameters including success rate, peak risk, average risk, maximum time to compute a waypoint and total flight time.

- Finally, a decentralized cooperative control architecture is proposed that can be used for a fleet of UAVs.

The structure of the thesis is as follows:

After the introduction in chapter 1, Chapter 2 presents an extensive literature survey and discusses some techniques in detail. Chapter 3 is about the optimal control approach to the design of a safe UAV path. Chapter 4 discusses the mixed/integer linear programming method and uses a receding

horizon strategy with soft dynamical boundaries for radar zone modelling to solve the problem. Chapter 5 presents the novel probabilistic formulation. Chapter 6 introduces the Voronoi diagram approach as a global path planner. It gives details of the software developed and uses it to find the length constraint path for a UAV. A comparison of the different path planning techniques is given at the end of this chapter. The proposed cooperative control architecture is given in Chapter 7.

Finally conclusions and future directions of research are given in Chapter 8.

Chapter 2

Literature Review

Planning is a main part of an array of engineering problems especially in operations research and artificial intelligence. In the past several decades, many researchers have put considerable efforts into research related to planning. For robot path planning alone, there exists a great number of different proposed approaches and system architectures having prose and cones over the others. The recent advancements in the technology of sensors, estimation and communication have opened the door to the possibility of using multiple autonomous vehicles to perform complex missions. Therefore multi-vehicle coordination has been one of the most active areas in the last decade. This chapter reviews some of the previously presented research with emphasis on those areas related to the topics of this thesis. The research presented here is a combination of several subproblems: path planning, multi-vehicle coordination, dynamic planning, and cooperative planning. Most of the work previously presented in this field has focused on a subset of these subproblems. Very little published work is targeted at the combined problem. The following sections review previous research related to each of the subproblems.

2.1 Single Vehicle Path Planning

One of the main functions of an autonomous vehicle is to move itself from some initial location to locations that are required for the vehicle to execute assigned tasks. This simple function actually involve several issues. First, the

vehicle must know where it currently is and the next location it should go. Secondly, it must also have the ability to plan a route between the locations and then navigate itself along the planned route. In this section, we discuss in detail, the several different techniques of path planning for a single vehicle to navigate in a given environment.

2.1.1 Graph-based Approaches

One of the most popular approaches to path planning is graph-based search. In this approach, as described by Murphy [71], the environment in which the vehicle operates is discretised and represented by a graph which is composed of a number of nodes linked together with arcs. Each node corresponds to a location in the environment and an arc links two adjacent nodes. There is cost associated with an arc. A path in this graph representation is a series of connected arcs. There is a trade off between how much detail is included in the graph structure and how long it takes to search the graph for an optimal solution. The most popular approaches are the rectilinear graph, the visibility graph and the Voronoi graph. In the rectilinear grid approach, the path planning area is quantized into $M \times N$ locations corresponding to the x and y directions of the two dimensional space. Node spacing is critical in UAV applications due to the turning rate constraint and the computations required to solve the problem in real time. Heading can also be added to the graph structure. It can be quantised to any desired level but a reasonable number seems to be eight different headings at each of the $M \times N$ nodes. If we put a turning rate constraint of 45 degrees on a grid of equal spacing in each direction, then at each node, the UAV may either continue in the current direction, turn 45 degrees to the left or turn 45 degrees to right. Figure 2.1 shows the possible paths for a UAV which starts from node $(1, 1)$ and whose destination is the node $(4, 1)$. The path chosen by the UAV is shown with bold lines in Figure 2.2.

Thrun [100] investigated the problem of high speed navigation of indoor mobile robots using a grid based algorithm called *value iteration*. In this work the map of the environment is created autonomously using sonar and

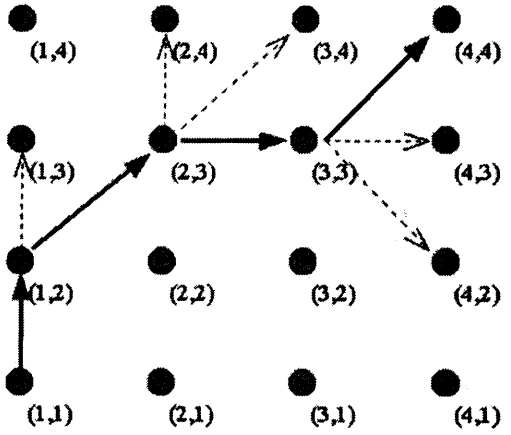


Figure 2.1: Possible paths on rectilinear grid

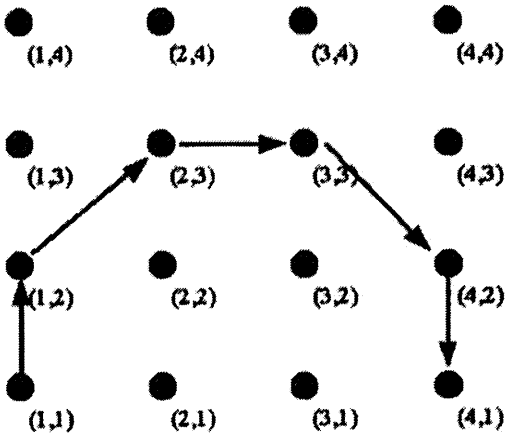


Figure 2.2: Path selected by UAV on rectilinear grid

camera information. The robot location and the map are estimated using Bayesian analysis techniques. This work was tested on various mobile robots and showed that the robots can effectively react to unanticipated obstacles in the test environment.

In the visibility graph approach, the obstacles are usually polygons and the set of possible paths for the UAV are the straight line segments. For each obstacle, the vertices of each polygon are connected to all other vertices of any other polygon provided an unobstructed straight line can be connected between these vertices. In order to avoid collisions with the obstacles, the vehicle in Figure 2.3 is represented by a point and the obstacles are expanded by an amount sufficient so that the real vehicle will not collide with them. The combination of the dotted lines and solid lines (obstacle boundary) gives

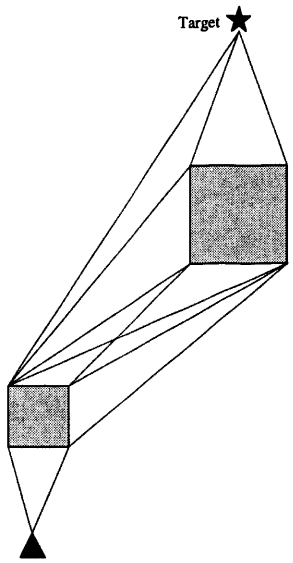


Figure 2.3: Visibility graph approach

the possible paths for the vehicle to follow. Each of the visibility graph line segments are assigned a cost based on many different criteria such as segment length or proximity to an obstacle. These segments are then searched using Dijkstra's algorithm [22] to find the lowest cost path. Neus and Maouche [72] used the visibility graph to solve the path planning problem.

In the Voronoi diagram approach (see Figure 2.4)which is used in many different fields, including computational fluid dynamics, computer graphics and statistics, complete knowledge of the number and location of each obstacle or in our case radar site will be assumed. Such a graph is constructed using Delaunay triangulation and its geometric dual, Voronoi polygons. For every triplet of radar sites, there exists a unique circle that passes through all three. Consider only those triplets whose circle does not enclose any other radar site. The set of all such triplets is called the Delaunay triangulation and the centres of the circles are called Voronoi points. we may now construct a graph by defining the vertices as the voronoi points. Edges are drawn to connect two Voronoi points if and only if their associated Delaunay triangles share an edge. By drawing all such edges, we construct the Voronoi diagram or graph. A graph search algorithm such as Dijkstra's algorithm or the A^* algorithm [73] can be used to find the shortest path. Several researchers have worked on this

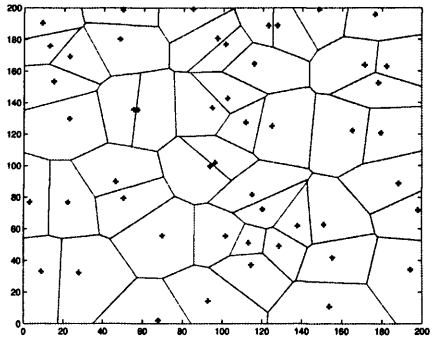


Figure 2.4: Voronoi graph for 50 threats

graph based planning concept and exercised it on many applications.

Mata and Mitchell [66] proposed a new algorithm for path planning on planar polyhedral surfaces. The terrain is presented as a sparse graph called a pathnet which links selected pairs of vertices of subdivisions of the terrain. They also gave analytical and experimental results which showed that the algorithm can provide highly competitive solutions compared with other approaches. One advantage of this method is that the discretisation of the terrain is independent of the scale of the available information about the terrain.

Bander [6] developed an adaptive A^* algorithm which used a heuristic function to guide the search process of the A^* algorithm to converge faster. The heuristic function is derived from a set of pre-determined optimal paths and a set of desirable paths which may or may not be optimal. This work also investigates mechanisms for incorporating several sources of knowledge and possibly human inputs to accelerate the search process.

2.1.2 Probabilistic Road Map Planners

Probabilistic Road Map (PRM) planning is one of the most efficient methods to compute collision free paths for vehicles or robots of virtually any type [55]. It is, in particular, suitable for robots with many degrees of freedom. This method consists of two phases: a building phase and a query phase. The building phase is the construction of a graph called the road map. The nodes in the road are collision free configurations and the edges linking the nodes are collision free paths for a robot to move from one configuration to another. The road map

is constructed by repeating the following two steps. First, a point is randomly picked in the configuration space and tested to see if it is in the collision free space. This step is repeated until the randomly chosen configuration is collision free. The second step is to connect the chosen configuration to the road map using a fast local planner. The above mentioned two steps are repeated by adding nodes and edges to the road map until a certain limit is reached. This limit has to be chosen based on the application. One possibility would be to impose a limit on the size of the road map. Another possibility would be to terminate the algorithm after a specified time has elapsed. The most important part in the road map construction algorithm (and also the query phase described later) is played by the local planner used. This sub-planner within the road map planner is used to compute paths between two configurations within the road map. Unlike the road map planner, this local planner is simple and not very powerful. An example for a possible local planner is a straight line planner that tries to connect two configurations with a straight line. It is obvious that this planner will fail if an obstacle is present between two nodes or configurations. For that reason the local planner is not very useful on its own. Together with an algorithm to find a way within the road map graph, however, the local planner can be used to find very complex paths around arbitrary obstacles. The general idea is that the road map graph describes the global structure of collision free space while the local planner is responsible for the details. Only the local planner actually computes the intermediate configurations of the final path. The query phase is finding a path between initial and goal configurations by connecting these configuration nodes to the road map and searching the road map for a sequence of edges linking the two nodes. This method was originally developed for holonomic robots in a static environment. One problem associated with query phase is that the paths returned may be twisted and overly long. For that reason smoothing and short-cutting the returned path is often required. One possible way is to try and skip waypoints. This is done by trying if the local planner can find a path between waypoint i and waypoint $i+2$, thus skipping waypoint $i+1$. A way to smooth the path for a rigid body robot would be to compute

some kind of spline using the waypoints as orientation, instead of connecting them with the local planner. Care must be taken to ensure that this spline path does not collide with an obstacle. Another problem is how powerful the local planner should be, considering that more powerful planners take more time to compute. Speed is more important than power because the running time of the local planner is the main component in the running time of the building phase as well as the query phase.

Overmars and Svestka [75] applied probabilistic road map techniques to simple holonomic robots such as free flying planner robots as well as non-holonomic robots with constrained kinematics and high degrees of freedom. This work shows that this technique can be extended to handle kinematic constraints in car like robots.

Lavalle and Kuffner [62] proposed a randomised path planning technique related to PRMs. This technique is used to compute collision free kinodynamic trajectories for high degrees of freedom robots with kinematic and dynamic constraints. Using a state space formulation, it transforms an n-dimensional planning problem in configuration space into a 2n-dimensional problem in state space. The key to this approach is the construction of a tree that can be used to explore the state space. This technique was applied to the path planning of hovercraft and satellites in cluttered environments.

Song et al. [94] proposed a new method of building and querying probabilistic roadmaps. In this method, some of the validation checks in the building phase are postponed to the querying phase. A coarse roadmap is built during the building phase and then further refined in the querying phase in the area of interest for the query. The roadmap is also customised to any specific query preferences such as maximum number of sharp turns. The results showed substantial improvement in performance and efficiency of the planning process.

2.1.3 Evolution-based Approaches

There are heuristic approaches that are gaining popularity. One of the most widely used is neural networks. Neural networks are loosely modeled on how

the brain works. They are an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neurons. Each neuron is linked to certain of its neighbors with varying coefficients of connectivity called weights that represents the strengths of these connections. Learning is accomplished by adjusting these weights to cause the overall network to output appropriate results. Before a neural network can be used, these weights need to be determined and this is referred to as training the network. To accomplish this, many inputs are given to the neural network as well as the associated desired outputs. The network weights are then adjusted to make the network output resemble as closely as possible the desired output. Once this training is finished, the output of the neural network is interpreted as the solution to the problem. Glasius in [39] is among many people to implement a neural network path planner. While they report good results for their planner, there are several disadvantages of neural networks. One of the most important is that training a neural network can consume a vast amount of computer time, sometimes months, depending upon the application.

Genetic algorithms have been used to solve the path planning problem, both in robotics and for UAV path planning. GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. GAs simulate the survival of the fittest among individuals over consecutive generations for solving a problem. Each individual represents a point in a search space and a possible solution. A fitness score is assigned to each solution representing the abilities of an individual to compete. The individual with the optimal fitness score is sought. The GA aims to use selective breeding of the solutions to produce offspring better than the parents by combining information from both the parents. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation once all mating opportunities in the old

population have been exhausted. In this way, it is hoped that over successive generations better solutions will thrive while the least fit solutions die out. New generations of solutions are produced containing, on average, more good solutions than in a previous generation. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand. One of the advantages of GAs over other methods is that they can simultaneously search multiple regions of the search space. This has the potential to be useful in UAV trajectory planning as multiple good trajectories can be developed in parallel.

Sugihara and Smith [97] describe their method of using a rectangular grid to parameterize the path and then employing genetic algorithms to find a good sequence of waypoints from the start location to the goal location whilst avoiding obstacles. Also Pellazar [79] describes his genetic algorithm approach to solving the UAV path planning problem. While both authors agree that genetic algorithms can provide high quality solutions, they also admit at the same time that one of the major disadvantages of the approach is the high level of computation that is required.

Fogel and Fogel [32] applied evolutionary programming to an optimal routing problem of autonomous underwater vehicles (AUVs). This work shows that the planning algorithm can handle unexpected changes in dynamic environments. They also consider a number of problems including multiple goal locations, detection avoidance and cooperative goal observation for a pair of AUVs. These complex problems were addressed by only modifying the performance objective function.

Xiao and Zhang [106] presented an adaptive evolutionary path planner for mobile robots. This approach combines off-line planning and on-line planning in the same evolutionary algorithm. In this approach, a path is represented as a set of waypoints chosen at random connecting the initial and goal locations. The probability of selecting different mutation operators is adapted during the search to improve performance.

Potter and Jong [81] developed the cooperative coevolution algorithm for

complex planning problems. This technique divides evolving solutions into several interacting coadapted subcomponents described as cooperating species. This work provides a case study involving the evolution of artificial neural networks and shows that this planning architecture can solve very complex problems which might not be possible with standard evolutionary algorithms.

Capozzi and Vagners [16, 17] presented an evolutionary technique for path planning of an aerial vehicle in a simulated dynamic environment. The planning algorithm was tested in several complex scenarios: varying terrain, wind variations, dynamic obstacles and moving targets. The simulation results show that the algorithm can efficiently search simultaneously in space and time to find feasible, near-optimal solutions.

Hocaoglu and Sanderson [44] developed an evolution based planning algorithm using a multi-resolution path representation. This approach does not require a map of the free configuration space. The use of the multi-resolution path representation reduces the complexity of the planning problem and in turn reduces the computation time. This work shows that the planning system can be applied to mobile robots or manipulators with many degrees of freedom and provides effective results. In addition, they also proposed a multi path planning algorithm that generates multiple alternative paths simultaneously.

2.1.4 Analogous Formulation Approaches

Analogous formulation entails transforming the path planning problem to an entirely different problem which has either already been solved or has convenient methods of solution. An example is the potential field method which can be used as an on-line planner or it can be used as off-line planner in the mass-spring-damper case or a chain link system.

Potential Field as an On-line Planner

The idea of using the physical principle of potential fields has been used to determine optimal paths for autonomous air vehicles. In this approach, the vehicle is treated as a point mass under the influence of an artificial potential field whose variations reflect the configuration of radars/obstacles. A radar exerts

a repulsive force while the goal location exerts an attractive force. The artificial force induced by the potential function at the the current configuration is regarded as the most promising direction of motion, and path generation proceeds along this direction by some increment in an iterative way. So at each iteration the vehicle simply follows the path of steepest decent along the negative gradient of this virtual field. In this way the vehicle will make its way towards the goal point while avoiding the obstacles. This can be used as an online path planning approach. McFarland [67] employed the potential field approach against mono-static radars while studying the effects of minimising radar cross section through vehicle orientation. However, there are some drawbacks with this approach [58].

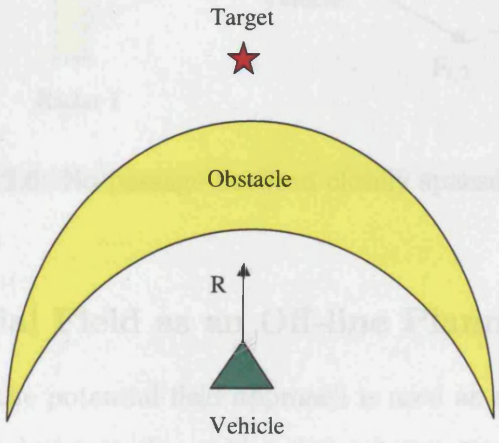


Figure 2.5: Trap situation due to local minima

Disadvantages: The two most important disadvantages are:

Trap Situation: Encountering traps, which are local minima in the potential function due to the arrangement of the obstacles is one of the most common problems with potential fields. This situation may occur when the vehicle runs into a dead end such as a U-shaped obstacle. Figure2.5 illustrate this problem, where R represents the resultant force of the potential field. This situation can be prevented by the use of various algorithms resulting in paths that are non-optimal.

No Passage Between Closely Spaced Obstacles: This situation arises when a vehicle tries to pass between two closely spaced radars. The combined force of repulsion of both radars along with the attractive force of the goal location

will prevent the vehicle from using a path that in realty it should be able to take. In Figure 2.6, F_r is the resultant repulsive force from the two radars and F_a is the attractive force pulling the robot towards the goal location. The vector sum of these two forces is R and this will be the vector that the robot will follow. Although the vehicle could physically fit between the two obstacles, the potential field approach does not generate this solution.

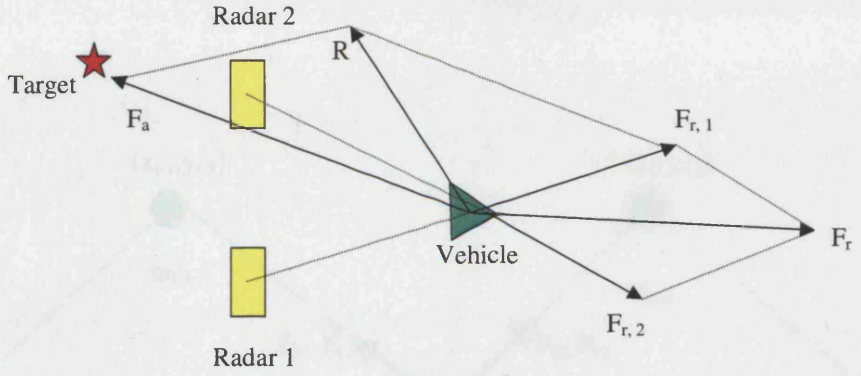


Figure 2.6: No passage between closely spaced radars

2.1.5 Potential Field as an Off-line Planner

In these methods, the potential field approach is used as an off-line approach to find a complete solution to the problem from the start to the goal locations.

Mass-Spring-Damper System

This approach likens the path planning problem to a physical system with the resulting steady state solution describing the path. In [10] Bortoff has used this method to solve the path planning problem. In this method, the UAV path is considered to be made up of a series of point masses connected to one another by springs and dampers as shown in Figure 2.7. One end of the chain is fixed to the UAV location, while the other is attached to the target location. The length of the path can be adjusted by assuming constant speed of the UAV, which we may normalise to 1 without loss of generality. This assumption is made only for planning purposes. The actual speed may be adjusted along the path in order to meet other requirements, e.g. rendezvous with other aircraft.

Let t_0 be the time at the present UAV location and t_f be the final location time. The constant speed assumption simplifies the problem because the path length is just $t_f - t_0$. For proper adjustment of the path length, t_f may be fixed or free.

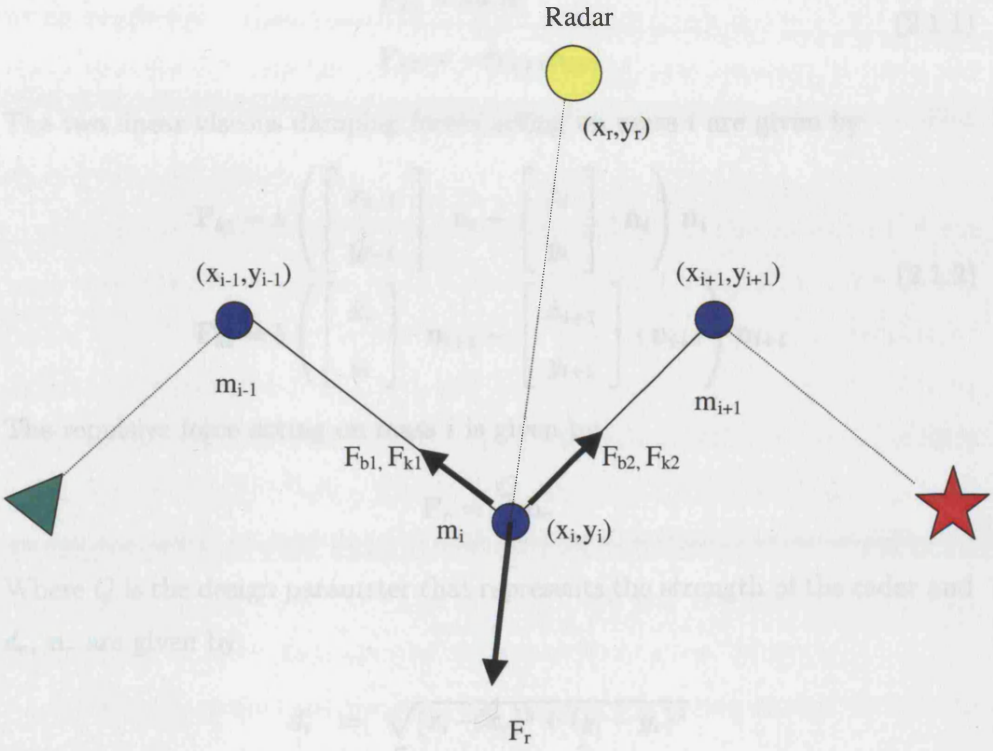


Figure 2.7: Mass spring damper system showing i^{th} mass under the influence of forces

Each threat location is the source of a repulsive force field that acts against each mass according to an inverse fourth law $(1/d^4)$. This causes the series of masses to move away from the threats. This system is given an initial configuration and then allowed to come to a steady state condition. At steady state the point masses define the way-points that should be followed. The steady state solution is calculated by solving a nonlinear, stable initial value problem. Taking $\mathbf{n}_i$ as the normal unit vector pointing from i^{th} mass to $(i-1)^{th}$ mass

$$\mathbf{n}_i = \begin{bmatrix} (x_{i-1} - x_i)/d_i \\ (y_{i-1} - y_i)/d_i \end{bmatrix}$$

where

$$d_i = \sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2}$$

The two spring forces acting at m_i are

$$\begin{aligned} \mathbf{F}_{k1} &= k d_i \mathbf{n}_i \\ \mathbf{F}_{k2} &= -k d_{i+1} \mathbf{n}_{i+1} \end{aligned} \tag{2.1.1}$$

The two linear viscous damping forces acting on mass i are given by

$$\begin{aligned} \mathbf{F}_{b1} &= b \left(\begin{bmatrix} \dot{x}_{i-1} \\ \dot{y}_{i-1} \end{bmatrix} \cdot \mathbf{n}_i - \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} \cdot \mathbf{n}_i \right) \mathbf{n}_i \\ \mathbf{F}_{b2} &= b \left(\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} \cdot \mathbf{n}_{i+1} - \begin{bmatrix} \dot{x}_{i+1} \\ \dot{y}_{i+1} \end{bmatrix} \cdot \mathbf{n}_{i+1} \right) \mathbf{n}_{i+1} \end{aligned} \tag{2.1.2}$$

The repulsive force acting on mass i is given by

$$\mathbf{F}_r = \frac{Q}{d_r^4} \mathbf{n}_r$$

Where Q is the design parameter that represents the strength of the radar and d_r, $\mathbf{n}_r$ are given by

$$\begin{aligned} d_r &= \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2} \\ \mathbf{n}_r &= \begin{bmatrix} (x_i - x_r)/d_r \\ (y_i - y_r)/d_r \end{bmatrix} \end{aligned}$$

If there are N radars, the total repulsive force is

$$\mathbf{F}_R = \sum_{r=1}^{N} \mathbf{F}_r = \sum_{r=1}^{N} \frac{Q}{d_r^4} \mathbf{n}_r \tag{2.1.3}$$

The equation of motion for the i^{th} mass is given by

$$m_i \begin{bmatrix} \ddot{x}_i \\ \ddot{y}_i \end{bmatrix} = \mathbf{F}_{k1} + \mathbf{F}_{k2} + \mathbf{F}_{b1} + \mathbf{F}_{b2} + \mathbf{F}_R \tag{2.1.4}$$

Disadvantages: When the chain of masses is nearby a set of radar sites, the virtual force pulls the masses away from each other, stretching out the chain. Conversely, in areas that are devoid of radar stations, the masses tend to draw close together. This effect is undesirable because the masses are then interpreted as way points. It would be more desirable to have a higher density

of way points near the radar sites, since one would expect the path to turn more often in these areas. In other words, the approach provides its lowest resolution data in the areas where high resolution is required. Also in this formulation, damping is only provided along the direction of the line connecting one mass to its neighbors. Thus, transverse modes that develop in the chain during simulation are not well-damped. This increase the time required to reach the stable equilibrium. By adding damping in the transverse direction, the solution should converge faster.

Chain-Link System One way to overcome some of the difficulties of the mass-spring-damper system is to replace the spring-damper sections of the model with rigid links of equal lengths. This approach was developed by McLain and Beard [69] and is based on the work of Udwadia and Kalaba [102]. In this approach, the threats continue to exhibit the $1/d^4$ repulsive force law but now instead of using springs and dampers, a straightening force from within the chain of rigid links is used to take out sharp corners resulting a flyable path for the UAV. The goal is to approximate the steady state shape of the chain. A brief description of the approach is given below:

Consider a path made up of $n+2$ point masses each having unit mass constrained to a two dimensional surface. Let $z_0 = [x_0, y_0]^T, z_1 = [x_1, y_1]^T, z_2 = [x_2, y_2]^T, \cdots, z_n = [x_n, y_n]^T, z_f = [x_f, y_f]^T \in \Re^2$ be the locations of the masses, where z_0 and z_f are the fixed initial and final positions respectively. If there are no constraints, then the dynamic equations describing the motion of the remaining n point masses are

$$\begin{aligned} \ddot{z}_1 &= u_1 \\ \ddot{z}_2 &= u_2 \\ &\vdots \\ \ddot{z}_n &= u_n \end{aligned} \tag{2.1.5}$$

where $u_1, u_2, \cdots, u_n \in \Re^2$ are the applied forces. Define $z = [z_1^T, z_2^T, \cdots, z_n^T]^T$ and $u = [u_1^T, u_2^T, \cdots, u_n^T]^T$, then we can write (2.1.5) as

$$\ddot{z} = u \tag{2.1.6}$$

Each u_i consists of two terms given by

$$u_i = F_{threat}(z_i) + F_{straighten}(z_i) \tag{2.1.7}$$

where

$$F_{threat}(z_i) = \gamma \sum_{r=1}^{R} \frac{z_i - p_r}{\|z_i - p_r\|^5}$$

such that $p_r \in \Re^2$ is the position of the r^{th} threat, γ is a weighting factor and R is the total number of threats. Also

$$F_{straighten}(z_i) = \delta \left(\frac{z_i - z_{i+2}}{\|z_i - z_{i+2}\|} + \frac{z_i - z_{i-2}}{\|z_i - z_{i-2}\|} \right)$$

where δ is a weighting factor for the straightening force which consists of two terms. The first force pulls the i^{th} node until the sub-chain composed of the $(i-2)^{nd}$, $(i-1)^{st}$ and i^{th} are in a straight line. The second force pushes the i^{th} node in such a way that i, $i+1$ and $i+2$ nodes form a straight line.

Suppose that all point masses remain at a constant distance L from each other. This constraint, in matrix form, is given by

$$\phi = \begin{bmatrix} \|z_1 - z_0\|^2 - L^2 \\ \|z_2 - z_1\|^2 - L^2 \\ \vdots \\ \|z_f - z_n\|^2 - L^2 \end{bmatrix} = 0 \tag{2.1.8}$$

Differentiating this constraint with respect to time results in the velocity constraint

$$\psi = \begin{bmatrix} 2(z_1 - z_0)^T(\dot{z}_1 - \dot{z}_0) \\ 2(z_2 - z_1)^T(\dot{z}_2 - \dot{z}_1) \\ \vdots \\ 2(z_f - z_n)^T(\dot{z}_f - \dot{z}_n) \end{bmatrix} = 0 \tag{2.1.9}$$

Differentiating once again results in the acceleration constraint, which in matrix form is given by

$$A(z)\ddot{z} = b(\dot{z}) \tag{2.1.10}$$

where

$$A(z) = \begin{bmatrix} (z_1 - z_0)^T & 0 & \cdots & 0 \\ -(z_2 - z_1)^T & (z_2 - z_1)^T & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & -(z_n - z_{n-1})^T & (z_n - z_{n-1})^T \\ 0 & \cdots & 0 & -(z_f - z_n)^T \end{bmatrix}$$

$$b(z) = \begin{bmatrix} (\dot{z}_1 - \dot{z}_0)^T(\dot{z}_1 - \dot{z}_0) \\ \vdots \\ (\dot{z}_f - \dot{z}_n)^T(\dot{z}_f - \dot{z}_n) \end{bmatrix}$$

Udwadia and Kalaba [102] show that using Gauss's principle from analytic dynamics, that the equation of motion (2.1.6), subject to the constraint (2.1.10), is given by the equation

$$\ddot{z} = u + A^+(z)(b(\dot{z}) - A(z)u) \tag{2.1.11}$$

where A^+ is the pseudo-inverse of A. The initial conditions for the system (2.1.11) must be chosen such that both $\phi = 0$ and $\psi = 0$ are satisfied. While solving system (2.1.11) numerical errors may cause the constraints (2.1.8) and (2.1.9) to drift from zero. When the constraints drift from zero, equation (2.1.11) no longer models the physical dynamics of the chain and there is no mechanism in equation (2.1.11) to bring the constraints back to zero. To overcome this problem, equation (2.1.11) can be modified to

$$\ddot{z} = u + A^+(z)(b(\dot{z}) - A(z)u) - \alpha\frac{\partial\phi^T}{\partial z}\phi - \beta\frac{\partial\psi^T}{\partial z}\psi \tag{2.1.12}$$

The two additional terms force the constrained accelerations to descend the gradient of the constraints until they are no longer violated. Large values of α and β ensure that the modified equation approximately models the dynamics of the constrained physical system. The dynamics of the chain can now be evolved using equation (2.1.12) and an adaptive step size ODE solver such as Matlab's *ode*45 algorithm. But the primary disadvantage of using that algorithm in this application is that the solution of the chain dynamics cannot be carried out in real time situation due to the adaptive step size of the algorithm. To

obtain a real time solution, equation (2.1.12) can be solved using an Euler type approximation like

$$z^{k+1} = z^k + h\left[u + A^+(z^k)\left(b(\dot{z}^k) - A(z^k)u\right) - \alpha\frac{\partial\phi^T}{\partial z^k}\phi - \beta\frac{\partial\psi^T}{\partial z^k}\psi\right] \quad (2.1.13)$$

where h is the step size.

2.2 Multi-Vehicle Planning

The driving force in the research related to planning for multiple autonomous vehicles is increasing demand of autonomous systems in the applications where a single vehicle is no longer capable of performing the necessary tasks. We categorize the efforts in this field into three groups: centralized approaches, decentralized approaches and market-based approaches.

2.2.1 Centralized Approaches

Centralized approaches are characterized by their system architectures in which there is only one control agent to manage the entire system. This control agent can be one of the vehicles in the system or the central command authority. The following research is relevant.

Adams et al. [1] presented a hierarchical architecture to control distributed teams of unmanned aerial vehicles (UAVs) in a military operational environment. This approach decomposes the system into several levels each of which contains decision making nodes that exchange information and interact with one another. The planning algorithm also accounts for uncertainty in estimated states and the risk of losing team members during a mission. The proposed structure allows human operators to interact with the system at any level.

Bellingham et al. [8] presented a planning system for a fleet of UAVs using mixed-integer linear programming (MILP). The planning algorithm accounts for the probability of losing UAVs during the mission which affects the operation of the other vehicles. The results show that the proposed system can

improve the probability of success of the mission and the probability of survival of the vehicles.

Maddula et al. [65] considered the problem of assigning targets to UAVs in a way that minimizes the maximum path length required for the vehicles to visit all the targets and minimizes the number of threats faced by each UAV. In Maddula's work, the environment is represented by a Voronoi diagram which is a graph of potential collision-free paths and waypoints. Assuming the environment is static, the planning algorithm computes an initial target assignment using a semi-greedy heuristic. The target assignment is further refined using constrained exchange of subpaths in the Voronoi diagram among the UAVs.

Capozzi [15] developed an evolution-based planning system which is capable of coordinating and generating paths for multiple autonomous vehicles. He applied this system to coordinated rendezvous and coordinated coverage of target problems. In this approach, the target assignment is based on the proximity of each vehicles trial path to the targets during the evolution process.

2.2.2 Decentralized Approaches

Problems often arise when applying centralized approaches to manage systems with a large number of vehicles in complex missions. These problems usually result in a lack of responsiveness of the system to changes in the environment. Decentralized approaches divide a complicated problem into manageable subproblems which can be solved by the components of the system. Some researchers have considered this idea and applied it to the problem of multi-vehicle coordination.

Estlin et al. [28], [29] did considerable work related to coordinating multiple rovers at the Jet Propulsion Laboratory. They developed a dynamic planning system to coordinate rovers in performing tasks for planetary science. The planning system is distributed and capable of coordinating activities among the rovers and monitoring plan execution, and performing replanning if necessary. Many Artificial Intelligence (AI) techniques are used in this planning system.

Parker [76], [77] developed a software architecture named ALLIANCE for fault tolerant cooperative control of teams of heterogeneous mobile robots. ALLIANCE is a fully distributed, behaviour-based architecture. Each robot is autonomous and individually has the ability to perform high-level functions and to select appropriate actions based on the requirement of the assigned tasks. This system was implemented on a team of mobile robots to perform a hazardous waste cleanup mission.

Aicardi [2] presented a decentralized approach to coordinate motion of a team of mobile robots based on team theory. The planned motion is computed using an algorithm derived from classical conservative force field techniques. Each decision maker computes its motion plan using sensor data and shared information. The results showed that the planning system is implementable in real-time.

Feddema et al. [30] demonstrated the use of decentralized control theory to analyze the problem of coordinating multiple robotic vehicles. This work focuses on the theoretical analysis of several properties of a system such as stability, observability and controllability. Stability analysis was used to determine limits on system parameters.

2.2.3 Market-Based Approaches

The concept of market-based approaches was introduced by Smith in his work on the Contract Net Protocol (CNP) [93]. This concept uses an economic model to coordinate multi-agent systems. Several researchers have adopted the concept and extended it to be applicable to many different related problems.

Sandholm [88] formalized the bidding and awarding decision process that was undefined in the original contract net task allocation protocol by using marginal cost calculations. Each agent is self-interest motivated so that it makes decisions based on its own local criteria. This work also extends the contract net protocol to allow for bidding clusters of tasks.

Fischer et al. [31] developed the MARS system which is designed for cooperative transportation scheduling of shipping companies. This work presents an extension of the contract net protocol for task decomposition and task

allocation. This approach provides increased flexibility that allows dynamic scheduling and execution.

Wellman and Wurman [104] developed a market-oriented programming technique for solving distributed resource allocation problems. Autonomous agents in the system interact by offering to buy or sell commodities at fixed unit prices. The computation of the resource allocation is finished when trading in the system reaches an equilibrium point. The market system is modelled analytically for equilibrium analysis.

Golfarelli et al. [41], [40], [42] proposed an approach using a negotiation protocol based on the contract net protocol. In this approach, the only possible type of contract is task swapping. No money or price system is used in the approach. The performance of the system can be improved by allowing tasks to be swapped in clusters. A clustering algorithm, which considers both spatial and temporal distances between tasks, is also presented.

Dias and Stentz [25] presented an architecture for coordinating multiple robots based on the concept of free market systems. The proposed market architecture defines explicit revenue and cost functions for the computation of bid prices. The results show that the overall team profit can be maximized by allowing agents to be self-interested.

2.3 Dynamic Planning

In most real-world applications, autonomous vehicles operate in dynamic uncertain environments. Therefore, a practical planning system must have the ability to dynamically replan in the face of unexpected circumstances. This section presents some example work in dynamic planning.

Stentz [95] developed the D* algorithm which is a dynamic variant of the classic A* graph search algorithm. It is designed to generate motion plans for a mobile robot operating in a partially known environment. He shows that the algorithm can handle situations where path cost parameters change during the search process by propagating these changes over only the effected portions of the search space. The planning algorithm is proved to be optimal and efficient

for sensor-equipped robots.

Brumitt and Stentz [12] developed a planning system for multiple mobile robots using the D* search algorithm. The planning system is capable of dynamically reassigning tasks to robots in order to minimize the time to complete a mission and generating optimal paths for the vehicles to accomplish the tasks. In this approach, a set of dynamic planners are used to continually update the paths of all robots to all goals during the mission. The mission planner updates the task assignment plans based on cost information provided by the dynamic planners.

Chien et al. [21] discussed the use of iterative repair techniques for continuous planning in space applications. They present an approach to integrate planning and execution in a feedback control fashion. This continuous planning framework is implemented on a system called CASPER. They showed that this approach can improve the responsiveness of the on-board planning process to changes in the environment or mission objectives.

2.4 Cooperative Planning

There has been increased interest in research related to cooperative planning. The goal of this research is to achieve cooperative behaviour in a system with multiple autonomous vehicles. Work by Cao [107] provides some definitions of cooperative behaviour and presents an extensive survey of the research in this field of study.

Gillen and Jacques [37] presented a system developed for finding and engaging targets using multiple autonomous wide area search munitions in unknown environments. This work investigates methods to improve the cooperative behaviour of the system which in turn increases the overall mission effectiveness. The cooperative engagement is controlled by a parameterized decision rule. A study of the sensitivities of the parameters to the precision of autonomous target recognition is also given.

McLain and Beard [70] presented a cooperative path planning approach for teams of multiple UAVs under timing constraints. This approach introduces

the use of coordination variables and coordination functions which define the cooperative strategy. The path planning problem is solved using a Voronoi diagram and Eppsteins k-best paths algorithm. The results show that the approach provides effective solutions to cooperative planning problems with three types of timing constraints: simultaneous arrival, tight sequencing, loose sequencing.

Polycarpou et al. [80], [78] developed a distributed planning system for cooperative search by a team of autonomous vehicles. Vehicles are equipped with sensors with limited viewing regions and wireless communication devices. The proposed system is capable of on-line learning of the environment and generating a search map which is shared between the vehicles. Each vehicle uses this search map and the predicted states of the other vehicles to compute its own collision-free trajectory that maximizes the team search coverage. The path planning algorithm is based on a q-step dynamic programming algorithm.

2.5 Conclusion

Different existing techniques in literature for autonomous vehicle path planning have been explored. The most popular approach to path planning is graph-based search and to use it efficiently there should be a trade off between how much detail is included in the graph structure and how long it takes to search the graph for an optimal solution. Potential field method has been studied as an on-line planar as well as an off-line planar. But it has certain drawbacks like trap situation and passage between closely spaced vehicles. Also in mass-spring-damper case, when the chain of masses is nearby a set of radar sites, the virtual force pulls the masses away from each other, stretching out the chain and the masses converge in the area which is far from radar sites. This is undesirable because the masses are then interpreted as way points. It would be more desirable to have a higher density of way points near the radar sites, since one would expect the path to turn more often in these areas. This difficulty was removed by chain link system assuming the each mass remain at a fixed distance from its neighbors. The damping force is provided along the direction

of the line connecting one mass to its neighbors. The transverse modes that develop in the chain during simulation are not well damped. This increases the time required to reach stable equilibrium. By adding damping in the transverse direction, the solution should converge faster. The main problem with neural network approach is that training a network can consume a vast amount of computer time, sometimes months, depending upon the application. One of the advantages of GAs over other methods is that they can simultaneously search multiple regions of the search space. This has the potential to be useful in UAV trajectory planning as multiple good trajectories can be developed in parallel. In PRM query phase, the paths returned may be twisted and overly long and due to this reason smoothing and short-cutting the returned path is often required. Another problem is how powerful the local planner should be, considering that more powerful planners take more time to compute. Speed is more important than power because the running time of the local planner is the main component in the running time of the building phase as well as the query phase.

Chapter 3

Multi-Objective Trajectory Design

3.1 Introduction

Due to well established theory and mathematical structure, optimal control will be investigated for the safe navigation of autonomous air vehicles. Starting with a brief review, the problem will be formulated considering different objectives. Necessary equations will be derived and solved for different cases to get analytical solutions since they give good insight into the problem. In some cases, analytic solutions are impossible and then numerical techniques are used to find the solution.

3.2 A Brief Review

Optimal control is a method that can be used to find the safe trajectories for UAVs by modeling different objectives and incorporating these objectives into a single performance index. It attempts to optimize by finding the control histories for a dynamical system for a given time period. Thus it is an indirect method of determining the optimum. The performance index is minimized by finding the time history of the control vector $u(t)$ instead of looking for the

states $\tilde{x}(t)$ themselves. The performance index to minimize is

$$J = \phi[\tilde{x}(t_f), t_f] + \int_{t_0}^{t_f} L[\tilde{x}(t), u(t), t]dt \tag{3.2.1}$$

subject to differential constraints

$$\dot{\tilde{x}} = f[\tilde{x}(t), u(t), t] \tag{3.2.2}$$

and boundary constraints

$$\tilde{x}(t_0) = \tilde{x}_0 \tag{3.2.3}$$

$$\tilde{\psi}[\tilde{x}(t_f), t_f] = 0 \tag{3.2.4}$$

In order to find the optimal solution, we need to clarify some preliminary ideas. If $\tilde{x}(t)$ is a continuous function of time t, then the differentials $d\tilde{x}(t)$ and dt are not independent. But a change in $\tilde{x}(t)$ can be defined that is independent of dt. The variation $\delta\tilde{x}(t)$ in $\tilde{x}(t)$ is defined as the incremental change in $\tilde{x}(t)$ when t is held fixed. The relation between $d\tilde{x}$ and $\delta\tilde{x}$ is given by [64]

$$d\tilde{x}(t) = \delta\tilde{x}(t) + \dot{\tilde{x}}dt \tag{3.2.5}$$

Also if $\tilde{x}(t) \in \Re^n$ is a function of t and

$$F = \int_{t_0}^{t_f} h[\tilde{x}(t), t]dt \tag{3.2.6}$$

then

$$dF = h[\tilde{x}(t_f), t_f]dt_f - h[\tilde{x}(t_0), t_0]dt_0 + \int_{t_0}^{t_f} h_{\tilde{x}}[\tilde{x}(t), t]\delta\tilde{x}dt \tag{3.2.7}$$

This is called the Leibniz rule and will be used here.

The performance index (3.2.1) is augmented by adjoining the differential constraints (3.2.2) and the boundary constraint (3.2.4) using lagrange multipliers $\lambda(t)$ and ν,

$$\bar{J} = \phi[\tilde{x}(t_f), t_f] + \nu^T\tilde{\psi}[\tilde{x}(t_f), t_f] + \int_{t_0}^{t_f} \{L[\tilde{x}(t), u(t), t] + \lambda^T(t)(f[\tilde{x}(t), u(t), t] - \dot{\tilde{x}})\}dt$$

The Hamiltonian is defined as

$$H(t) = L[\tilde{x}(t), u(t), t] + \lambda^T(t)f[\tilde{x}(t), u(t), t]$$

The above augmented performance index in terms of the Hamiltonian is written as

$$\bar{J} = \Phi(t_f) + \int_{t_0}^{t_f} [H(t) - \lambda^T(t)\dot{\tilde{x}}]dt$$

where

$$\Phi(t_f) = \phi[\tilde{x}(t_f), t_f] + \nu^T\tilde{\psi}[\tilde{x}(t_f), t_f]$$

By Leibniz's rule, the increment in $\bar{J}$ as a function of increments in λ, ν, u and t is

$$\begin{aligned} d\bar{J} = \quad & \left[\left(\phi_{\tilde{x}} + \nu^T\tilde{\psi}_{\tilde{x}}\right) d\tilde{x}\right]_{t=t_f} + \left[\left(\phi_t + \nu^T\tilde{\psi}_t\right) dt\right]_{t=t_f} + \left[\tilde{\psi}^T\right]_{t=t_f} d\nu \\ & + \left[\left(H - \lambda^T\dot{\tilde{x}}\right) dt\right]_{t=t_f} - \left[\left(H - \lambda^T\dot{\tilde{x}}\right) dt\right]_{t=t_0} \\ & + \int_{t_0}^{t_f} \left[H_{\tilde{x}}\delta\tilde{x} + H_u\delta u - \lambda^T\delta\dot{\tilde{x}} + \left(H_\lambda - \dot{\tilde{x}}^T\right)\delta\lambda\right] dt \end{aligned} \tag{3.2.8}$$

To eliminate the variation in $\dot{\tilde{x}}$, we integrate the term $-\int_{t_0}^{t_f} \lambda^T\delta\dot{\tilde{x}}dt$ by parts:

$$\begin{aligned} -\int_{t_0}^{t_f} \lambda^T\delta\dot{\tilde{x}}dt &= -[\lambda^T\delta\tilde{x}]_{t=t_f} + [\lambda^T\delta\tilde{x}]_{t=t_0} + \int_{t_0}^{t_f} \dot{\lambda}^T\delta\tilde{x}dt \\ &= -[\lambda^T d\tilde{x}]_{t=t_f} + [\lambda^T\dot{\tilde{x}}dt]_{t=t_f} + [\lambda^T d\tilde{x}]_{t=t_0} - [\lambda^T\dot{\tilde{x}}dt]_{t=t_0} \\ &\quad + \int_{t_0}^{t_f} \dot{\lambda}^T\delta\tilde{x}dt \end{aligned} \tag{3.2.9}$$

where we have used the relation (3.2.5). Substituting this in (3.2.8), we have

$$\begin{aligned} d\bar{J} = \quad & \left[\left(\phi_{\tilde{x}} + \nu^T\tilde{\psi}_{\tilde{x}} - \lambda^T\right) d\tilde{x}\right]_{t=t_f} + \left[\left(\phi_t + \nu^T\tilde{\psi}_t + \lambda^T\dot{\tilde{x}} + H - \lambda^T\dot{\tilde{x}}\right) dt\right]_{t=t_f} \\ & + \left[\tilde{\psi}^T\right]_{t=t_f} d\nu - \left[\left(H - \lambda^T\dot{\tilde{x}} + \lambda^T\dot{\tilde{x}}\right) dt\right]_{t=t_0} + [\lambda^T d\tilde{x}]_{t=t_0} \\ & + \int_{t_0}^{t_f} \left[\left(H_{\tilde{x}} + \dot{\lambda}^T\right)\delta\tilde{x} + H_u\delta u - \lambda^T\delta\dot{\tilde{x}} + \left(H_\lambda - \dot{\tilde{x}}^T\right)\delta\lambda\right] dt \end{aligned} \tag{3.2.10}$$

According to Lagrange theory, the constrained minimum of J is attained at the unconstrained minimum of $\bar{J}$. This is achieved when $d\bar{J} = 0$ for all independent increments it its arguments. Here we have assumed that t_0 and $\tilde{x}(t_0)$ are both fixed and known so that dt_0 and $d\tilde{x}(t_0)$ are both zero. Summarising the above results: the optimal control input $u(t)$ that minimizes the performance index J must satisfy

$$\text{Performance Index:} \quad \phi[\tilde{x}(t_f), t_f] + \int_{t_0}^{t_f} L[\tilde{x}(t), u(t), t]dt \tag{3.2.11}$$

$$\text{Differential Constraints:} \quad \dot{\tilde{x}} = H_\lambda^T = f(\tilde{x}, u, t) \tag{3.2.12}$$

$$\text{Co-State Equations:} \quad \dot{\lambda}^T = -H_{\tilde{x}} = -L_{\tilde{x}} - \lambda^T f_{\tilde{x}} \tag{3.2.13}$$

$$\text{Initial Conditions:} \quad \tilde{x}(t_0) = \tilde{x}_0 \tag{3.2.14}$$

Terminal Constraints: $$\psi[\tilde{x}(t_f), t_f] = 0 \tag{3.2.15}$$

Boundary Condition: $$[(\Phi_{\tilde{x}} - \lambda^T)dx]_{t=t_f} + [(\Phi_t + H)dt]_{t=t_f} = 0 \tag{3.2.16}$$

Optimality Condition: $$H_u = L_u + \lambda^T f_u = 0 \tag{3.2.17}$$

The time derivative of the Hamiltonian is

$$\begin{aligned} \dot{H} &= H_t + H_{\tilde{x}}\dot{\tilde{x}} + H_u\dot{u} + \dot{\lambda}^T f \\ &= H_t + H_u\dot{u} + (H_{\tilde{x}} + \dot{\lambda}^T)f \end{aligned} \tag{3.2.18}$$

For the optimal solution $H_u = 0$ and $H_{\tilde{x}} + \dot{\lambda}^T = 0$. Hence

$$\dot{H} = H_t \tag{3.2.19}$$

In the case when H is not an explicit function of t, then

$$\dot{H} = 0 \tag{3.2.20}$$

Hence for time invariant systems and cost functions, the Hamiltonian is a constant on the optimal trajectory

3.3 Radar Equation

A radar transmits an electromagnetic signal and this signal is reflected back from the target. A monostatic radar configuration consists of a collocated transmitter and receiver. When the transmitter and receiver are separated geographically, this is known as a bistatic radar configuration. By observing the time interval between the transmitted pulse and the reflected echo, the range to the target can be determined and is given by

$$d = \frac{c\Delta t}{2}$$

where c is the velocity of the radar signal and Δt is the time elapsed from transmission to reception of the echo signal. Consider a radar radiating a signal of power P which is uniformly distributed on the surface of a sphere of

radius d having the radar at its centre. Since radars use a directional antenna, we can adjust the signal power by a gain factor G. So the power density at the target which is at a distance d from the radar is given by

$$\text{Power density at the target} = \frac{PG}{4\pi d^2}$$

Suppose there is no absorption of energy by the target and after colliding with the surface of the target, it is fully reflected back. Suppose σ is the radar cross section of the target which varies significantly with the attitude (elevation and azimuth angles θ_{el} and θ_{az}, respectively) of the UAV with respect to the radar and A_e is the effective area of the receiving antenna, then the power of the reflected signal at radar receiver is given by

$$\text{echo power at the radar receiver} = \frac{PG\sigma(\theta_{el}, \theta_{az})A_e}{(4\pi d^2)^2}$$

The ratio of the received radar power to the transmitted power is called the radar equation and is given by

$$\frac{\text{received radar power}}{\text{transmitted power}} = \frac{G\sigma A_e}{(4\pi)^2 d^4} = \frac{\text{constant}}{d^4}$$

Hence for $\sigma(\theta_{el}, \theta_{az}) = 1$ this ratio is inversely proportional to the fourth power of the distance between the radar and the target.

Theorem 1 *If the cost of a UAV at a distance d_i from a radar site located at (x_i, y_i) and having strength α_i is $J = \frac{\alpha_i}{d_i^4}$, then the cost to go from an initial point (x_1^1, y_1^1) to a final point (x_2^N, y_2^N) on a path consisting of N line segments for a single radar exposure minimization can be written as*

$$J_i = \frac{\alpha_i}{2v} \sum_{j=1}^{N} \frac{(1 + m^{j2})^{\frac{3}{2}}}{(c^j - y_i + m^j x_i)^3} [(\theta_2^j - \theta_1^j) + \cos(\theta_2^j + \theta_1^j) \sin(\theta_2^j - \theta_1^j)]$$

where

$$\begin{aligned}
\theta_1^j &= \arctan[\frac{x_1^j(1 + m^{j2}) + m^j c^j - m^j y_i - x_i}{c^j - y_i + m^j x_i}] \\
\theta_2^j &= \arctan[\frac{x_2^j(1 + m^{j2}) + m^j c^j - m^j y_i - x_i}{c^j - y_i + m^j x_i}] \\
c^j &= y_1^j - m^j x_1^j \\
m^j &= \frac{y_2^j - y_1^j}{x_2^j - x_1^j}
\end{aligned}$$

with (x_1^j, y_1^j) *and* (x_2^j, y_2^j) *are initial and final points of the the* j^{th} *line segment and* v *is the speed of a UAV respectively.*

Proof: The cost to travel a path can be obtained by integrating the radar equation and is given by

$$J_i = \int_{t_0}^{t_f} \frac{\alpha_i}{d_i^4} dt \tag{3.3.21}$$

where α_i is the strength of the radar located at (x_i, y_i). Suppose the vehicle is moving with constant speed v. Then $\frac{ds}{dt} = v$ or $dt = \frac{ds}{v}$ which can be written as

$$dt = \frac{\sqrt{1 + (\frac{dy}{dx})^2}}{v} dx$$

Hence the cost becomes

$$J_i = \int_{x_0}^{x_f} \frac{\alpha_i \sqrt{1 + (\frac{dy}{dx})^2} dx}{v[(x - x_i)^2 + (y - y_i)^2]^2}$$

The path consists of N straight line segments, and so the above equation can be written as a sum of integrations over each segment j

$$J_i = \frac{\alpha_i}{v} \sum_{j=1}^{N} \int_{x_1^j}^{x_2^j} \frac{\sqrt{1 + (\frac{dy}{dx})^2} dx}{[(x - x_i)^2 + (y - y_i)^2]^2} \tag{3.3.22}$$

We will consider the integral cost to travel for the j^{th} edge only

$$J_{ij} = \int_{x_1^j}^{x_2^j} \frac{\sqrt{1 + (\frac{dy}{dx})^2} dx}{[(x - x_i)^2 + (y - y_i)^2]^2} \tag{3.3.23}$$

The equation of the line segment passing through (x_1^j, y_1^j) and (x_2^j, y_2^j) is given by $y = m^j x + c^j$. Where $m^j = \frac{y_2^j - y_1^j}{x_2^j - x_1^j}$ is the slope and $c^j = y_1^j - m^j x_1^j$ is the y-intercept. Putting these values in the above expression, we have

$$J_{ij} = \int_{x_1^j}^{x_2^j} \frac{\sqrt{1 + m^{j^2}} dx}{[(x - x_i)^2 + (m^j x + c^j - y_i)^2]^2}$$

$$J_{ij} = \frac{1}{(1 + m^{j^2})^{\frac{3}{2}}} \int_{x_1^j}^{x_2^j} \frac{dx}{\left[x^2 + \frac{2(m^j c^j - m^j y_i - x_i)}{1 + m^{j^2}} x + \frac{x_i^2 + (c^j - y_i)^2}{1 + m^{j^2}}\right]^2}$$

Let $a = \frac{(m^j c^j - m^j y_i - x_i)}{1 + m^{j^2}}$ and $b = \frac{x_i^2 + (c^j - yi)^2}{1 + m^{j^2}}$, then

$$J_{ij} = \frac{1}{(1 + m^{j^2})^{\frac{3}{2}}} \int_{x_1^j}^{x_2^j} \frac{dx}{[x^2 + 2ax + b]^2}$$

or

$$J_{ij} = \frac{1}{(1+m^{j2})^{\frac{3}{2}}} \int_{x_1^j}^{x_2^j} \frac{dx}{[(x+a)^2 + (b-a^2)]^2}$$

Now $b - a^2$ can be simplified as

$$\begin{aligned} b - a^2 &= \frac{x_i^2 + (c^j - y_i)^2}{1+m^{j2}} - \frac{[m^j(c^j - y_i) - x_i]^2}{[1+m^{j2}]^2} \\ &= \left[\frac{c^j - y_i + m^j x_i}{1+m^{j2}}\right]^2 = c^2(say) \end{aligned}$$

Therefore we can write

$$J_{ij} = \frac{1}{(1+m^{j2})^{\frac{3}{2}} c^4} \int_{x_1^j}^{x_2^j} \frac{dx}{\left[\left(\frac{x+a}{c}\right)^2 + 1\right]^2}$$

Let $X = \frac{x+a}{c} \Rightarrow dx = cdX$; at $x = x_1^j, X = \frac{x_1^j + a}{c} = X_1$ and at $x = x_2^j, X = \frac{x_2^j + a}{c} = X_2$. Then the above integral becomes

$$J_{ij} = \frac{1}{(1+m^{j2})^{\frac{3}{2}} c^3} \int_{X_1}^{X_2} \frac{dX}{(1+X^2)^2}$$

Let us substitute $X = \tan\theta \Rightarrow dX = \sec^2\theta d\theta$; at $X = X_1, \theta_1 = \arctan X_1$ and at $X = X_2, \theta_2 = \arctan X_2$. Then after simplification

$$\begin{aligned} J_{ij} &= \frac{(1+m^{j2})^{\frac{3}{2}}}{(c^j - y_i + m^j x_i)^3} \int_{\arctan X_1}^{\arctan X_2} \frac{\sec^2\theta d\theta}{(1+\tan^2\theta)^2} \\ &= \frac{(1+m^{j2})^{\frac{3}{2}}}{2(c^j - y_i + m^j x_i)^3} \int_{\arctan X_1}^{\arctan X_2} (1+\cos 2\theta) d\theta \\ &= \frac{(1+m^{j2})^{\frac{3}{2}}}{2(c^j - y_i + m^j x_i)^3} \left[(\theta_2^j - \theta_1^j) + \frac{\sin 2\theta_2^j - \sin 2\theta_1^j}{2}\right] \\ &= \frac{(1+m^{j2})^{\frac{3}{2}}}{2(c^j - y_i + m^j x_i)^3} [(\theta_2^j - \theta_1^j) + \cos(\theta_2^j + \theta_1^j)\sin(\theta_2^j - \theta_1^j)] \end{aligned}$$

Hence the cost to travel with constant speed v for the whole path J_i is given by

$$J_i = \frac{s_i}{v} \sum_{j=1}^{N} J_{ij} = \frac{s_i}{v} \sum_{j=1}^{N} \frac{(1+m^{j2})^{\frac{3}{2}}}{2(c^j - y_i + m^j x_i)^3} [(\theta_2^j - \theta_1^j) + \cos(\theta_2^j + \theta_1^j)\sin(\theta_2^j - \theta_1^j)]$$

or

$$J_i = \frac{s_i}{2v} \sum_{j=1}^{N} \frac{(1+m^{j2})^{\frac{3}{2}}}{(c^j - y_i + m^j x_i)^3} [(\theta_2^j - \theta_1^j) + \cos(\theta_2^j + \theta_1^j)\sin(\theta_2^j - \theta_1^j)] \quad (3.3.24)$$

where

$$\theta_1^j = \arctan X_1 = \arctan \frac{x_1^j + a}{c} = \arctan \left[\frac{x_1^j(1+m^{j^2}) + m^j c^j - m^j y_i - x_i}{c^j - y_i + m^j x_i} \right]$$

Similarly

$$\theta_2^j = \arctan X_2 = \arctan \frac{x_2^j + a}{c} = \arctan \left[\frac{x_2^j(1+m^{j^2}) + m^j c^j - m^j y_i - x_i}{c^j - y_i + m^j x_i} \right]$$

and

$$\begin{aligned} m^j &= \frac{y_2^j - y_1^j}{x_2^j - x_1^j} \\ c^j &= y_1^j - m^j x_1^j \end{aligned}$$

Remark: If there are M radars, then the cost to travel from one point to another point on a piecewise linear path is the sum of the costs due to each radar and is given by

$$J = \sum_{i=1}^{M} J_i = \frac{1}{2v} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{s_i(1+m^{j^2})^{\frac{3}{2}}}{(c^j - y_i + m^j x_i)^3} [(\theta_2^j - \theta_1^j) + \cos(\theta_2^j + \theta_1^j) \sin(\theta_2^j - \theta_1^j)] \tag{3.3.25}$$

3.4 Equations of Motion

The UAV is constrained to fly above the terrain profile at an altitude of h_c which is called terrain clearance. The terrain clearance may be constant or a specified function of downrange and crossrange. Two coordinate systems will be used to extract the equations of motion as shown in Figure 3.1. A local coordinate system (x_l, y_l, z_l) is taken with its origin located on the terrain profile and the $x_l y_l-$plane coincident with the local tangent plane with z_l defining the outward normal. The inertial coordinate system (x, y, h) is defined with the $xy-$plane coincide with the flat ground and h is the height axis taken positive in the upward direction. The x_l-axis of the local level frame is taken parallel to the $xh-$plane of the inertial coordinate system. So the velocity vector is constrained to lie in the $x_l y_l-$plane making a heading angle ψ with the x_l-axis. The equations of motion in the local level frame can be written

as

$$\begin{aligned} \dot{x}_l &= v\cos\psi \\ \dot{y}_l &= v\sin\psi \\ \dot{z}_l &= 0 \end{aligned} \tag{3.4.26}$$

Taking $g(x,y)$ as the terrain profile in inertial frame and defining $G(x,y)$ as the algebraic sum of the terrain profile and terrain clearance, then the the UAV height in inertial frame is given by

$$h = G(x,y) = g(x,y) + h_c \tag{3.4.27}$$

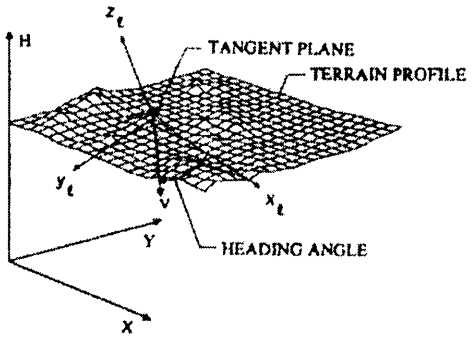


Figure 3.1: Coordinate system

In order to transform the system (3.4.26) to the inertial frame, unit vectors (i,j,k) along the local level frame need to be determined. The outward drawn unit normal vector k and the unit tangent vector i on the terrain profile $G(x,y)$ are given by

$$k = \frac{1}{\sqrt{1+G_x^2+G_y^2}} \begin{bmatrix} -G_x \\ -G_y \\ 1 \end{bmatrix} \tag{3.4.28}$$

$$i = \frac{1}{\sqrt{1+G_x^2}} \begin{bmatrix} 1 \\ 0 \\ G_x \end{bmatrix} \tag{3.4.29}$$

The unit vector j can be obtained by the cross product $k \times i$

$$j = \frac{1}{\sqrt{1+G_x^2+G_y^2}\sqrt{1+G_x^2}} \begin{bmatrix} -G_xG_y \\ 1+G_x^2 \\ G_y \end{bmatrix} \tag{3.4.30}$$

where G_x and G_y are the partial derivatives of G with respect to x and y respectively. Taking I ,J, K as the unit vectors along inertial frame and transforming the velocity from the local level tangent plane to the inertial frame

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} i.I & j.I & k.I \\ i.J & j.J & k.J \\ i.K & j.K & k.K \end{bmatrix} \begin{bmatrix} \dot{x}_l \\ \dot{y}_l \\ \dot{z}_l \end{bmatrix} \tag{3.4.31}$$

which implies that

$$\begin{aligned} \dot{x} &= \frac{v\cos\psi}{\sqrt{1+G_x^2}} - \frac{G_xG_yv\sin\psi}{\sqrt{(1+G_x^2)(1+G_x^2+G_y^2)}} \\ \dot{y} &= \frac{\sqrt{1+G_x^2}v\sin\psi}{\sqrt{1+G_x^2+G_y^2}} \\ \dot{h} &= v\sin\gamma \end{aligned} \tag{3.4.32}$$

where γ is the flight path angle given by

$$\gamma = \arcsin\left(\frac{G_x\dot{x}+G_y\dot{y}}{v}\right) \tag{3.4.33}$$

3.5 Derivation of Optimal Trajectory

The performance index to be minimized in its most general form is the tradeoff among different objective functions of flight time, radar exposure and terrain following:

$$J = \int_0^{t_f} \left[q_1 + q_2G(x,y) + q_3\sum_{k=1}^{M}\frac{s_k}{d_k^4}\right]dt \tag{3.5.34}$$

where d_k is the distance of the current position of the UAV from the k^{th} radar which is assumed to lie on the terrain profile and s_k is the strength of that radar.

$$d_k = \sqrt{(x-x_k)^2+(y-y_k)^2+(G(x,y)-g(x_k,y_k))^2} \tag{3.5.35}$$

subject to

$$x(0) = x_0,\ y(0) = y_0,\ h(0) = g(x_0,y_0) + h_c = h_0 \tag{3.5.36}$$

$$x(t_f) = x_f,\ y(t_f) = y_f,\ h(t_f) = g(x_f,y_f) + h_c = h_f \tag{3.5.37}$$

If h_c is constant, the height at any point depends on $g(x,y)$, which in turn depends upon the states x and y. Here

$$\phi = 0, \ \tilde{\psi} = \begin{bmatrix} x(t_f) - x_f \\ y(t_f) - y_f \end{bmatrix} \tag{3.5.38}$$

$$\Phi = \phi + \nu^T \tilde{\psi} = \nu_x(x(t_f) - x_f) + \nu_y(y(t_f) - y_f) \tag{3.5.39}$$

The Hamiltonian is

$$\begin{aligned} H(x,y,v,\psi,\lambda,t) &= L(x,y,v,\psi,t) + \lambda^T f(x,y,v,\psi,t) \\ &= q_1 + q_2 G(x,y) + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4} \\ &+ \lambda_x \left[\frac{v\cos\psi}{C_1} - \frac{vG_xG_y\sin\psi}{C_1C_2} \right] + \lambda_y \left[\frac{C_1 v \sin\psi}{C_2} \right] \end{aligned} \tag{3.5.40}$$

where

$$C_1 = \sqrt{1 + G_x^2} \tag{3.5.41}$$

$$C_2 = \sqrt{1 + G_x^2 + G_y^2} \tag{3.5.42}$$

Suppose v is constant, then from (3.2.13), the costate equations are

$$\begin{aligned} -\dot{\lambda}_x &= q_2 G_x - 4q_3 \sum_{k=1}^{M} \frac{s_k d_{k_x}}{d_k^5} - \frac{\lambda_x v G_x G_{xx}}{C_1^3}\cos\psi - \frac{\lambda_x v(G_{xx}G_y + G_x G_{xy})}{C_1 C_2}\sin\psi \\ &+ \frac{\lambda_x v G_x G_y (C_1^2 G_x G_{xx} + C_1^2 G_y G_{xy} + C_2^2 G_x G_{xx})}{C_1^3 C_2^3}\sin\psi \\ &- \frac{\lambda_y v (C_1^2 G_x G_{xx} + C_1^2 G_y G_{xy} - C_2^2 G_x G_{xx})}{C_1 C_2^3}\sin\psi \end{aligned} \tag{3.5.43}$$

$$\begin{aligned} -\dot{\lambda}_y &= q_2 G_y - 4q_3 \sum_{k=1}^{M} \frac{s_k d_{k_y}}{d_k^5} - \frac{\lambda_x v G_x G_{xy}}{C_1^3}\cos\psi - \frac{\lambda_x v(G_{xy}G_y + G_x G_{yy})}{C_1 C_2}\sin\psi \\ &+ \frac{\lambda_x v G_x G_y (C_1^2 G_x G_{xy} + C_1^2 G_y G_{yy} + C_2^2 G_x G_{xy})}{C_1^3 C_2^3}\sin\psi \\ &- \frac{\lambda_y v (C_1^2 G_x G_{xy} + C_1^2 G_y G_{yy} - C_2^2 G_x G_{xy})}{C_1 C_2^3}\sin\psi \end{aligned} \tag{3.5.44}$$

By the optimality condition (3.2.17)

$$\lambda_x \left[-\frac{v\sin\psi}{C_1} - \frac{vG_xG_y\cos\psi}{C_1C_2} \right] + \frac{\lambda_y C_1 v\cos\psi}{C_2} = 0$$

$$\Rightarrow \qquad \lambda_y = \frac{C_2 \sin\psi + G_x G_y \cos\psi}{C_1^2 \cos\psi}\lambda_x \tag{3.5.45}$$

The proposed optimal control problem has a constant of motion because the variational Hamiltonian is not explicitly dependent on time and the final time is free

$$\Rightarrow \qquad H(t) = 0, \quad 0 \le t \le t_f$$

or

$$q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4} + \lambda_x \left[\frac{v\cos\psi}{C_1} - \frac{G_x G_y v \sin\psi}{C_1 C_2}\right] + \lambda_y \frac{C_1 v \sin\psi}{C_2} = 0 \tag{3.5.46}$$

Then substituting λ_y from (3.5.45) into (3.5.46), we get

$$\begin{aligned} q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4} \quad &+ \quad \lambda_x \left[\frac{v\cos\psi}{C_1} - \frac{G_x G_y v \sin\psi}{C_1 C_2}\right] \\ &+ \quad \frac{C_1 v \sin\psi}{C_2}\left[\frac{C_2 \sin\psi + G_x G_y \cos\psi}{C_1^2 \cos\psi}\lambda_x\right] = 0 \end{aligned}$$

$$\Rightarrow \qquad \lambda_x = -\left(q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right) \frac{C_1 \cos\psi}{v} \tag{3.5.47}$$

and using equation (3.5.47) in (3.5.45)

$$\Rightarrow \qquad \lambda_y = -\frac{1}{C_1 v}\left(C_2 \sin\psi + G_x G_y \cos\psi\right)\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \tag{3.5.48}$$

Now substituting λ_x and λ_y from (3.5.47) and (3.5.48) in equation (3.5.43) gives

$$\begin{aligned} -\dot{\lambda}_x \;=\; & q_2 G_x - 4 q_3 \sum_{k=1}^{M} \frac{s_k d_{k_x}}{d_k^5} + \frac{G_x G_{xx}}{C_1^2}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \cos^2\psi \\ &+ \frac{G_{xx} G_y + G_x G_{xy}}{C_2}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \cos\psi \sin\psi \\ &- \frac{G_x G_y (G_x G_{xx} + G_y G_{xy})}{C_2^3}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \cos\psi \sin\psi \\ &- \frac{G_x^2 G_y G_{xx}}{C_1^2 C_2}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \cos\psi \sin\psi \\ &+ \frac{(C_1^2 G_x G_{xx} + C_1^2 G_y G_{xy} - C_2^2 G_x G_{xx})}{C_1^2 C_2^2}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \sin^2\psi \\ &+ \frac{G_x G_y (G_x G_{xx} + G_y G_{xy})}{C_2^3}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \cos\psi \sin\psi \\ &- \frac{G_x^2 G_y G_{xx}}{C_1^2 C_2}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \cos\psi \sin\psi \end{aligned} \tag{3.5.49}$$

or

$$
\begin{aligned}
-\dot{\lambda}_x &= q_2 G_x - 4q_3 \sum_{k=1}^{M} \frac{s_k d_{k_x}}{d_k^5} + \frac{1}{C_1^2 C_2^2}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \\
&\cdot \left[C_2^2 G_x G_{xx} \cos^2\psi + C_2\left(C_1^2 G_y G_{xx} + C_1^2 G_x G_{xy} - 2G_x^2 G_y G_{xx}\right)\cos\psi \sin\psi\right. \\
&+ \left.\left(C_1^2 G_x G_{xx} + C_1^2 G_y G_{xy} - C_2^2 G_x G_{xx}\right)\sin^2\psi\right] \qquad (3.5.50)
\end{aligned}
$$

Now differentiating equation (3.5.47) with respect to time and using the values of $\dot{x}$ and $\dot{y}$ from (3.4.32)

$$
\begin{aligned}
-\dot{\lambda}_x &= \frac{1}{v(C_1^2 C_2)}\left[q_1 + q_2 G + q_3 \sum_{k=1}^{M} \frac{s_k}{d_k^4}\right] \\
&\cdot \left[C_2 G_x G_{xx} v\cos^2\psi + (C_1^2 G_x G_{xy} - G_x^2 G_y G_{xx}) v\cos\psi\sin\psi - C_1^3 C_2 \sin\psi\dot{\psi}\right] \\
&+ \frac{q_2\cos\psi}{C_2}\left[C_2 G_x\cos\psi - G_x^2 G_y \sin\psi + C_1^2 G_y \sin\psi\right] \\
&- \frac{4q_3\cos\psi}{C_2}\sum_{k=1}^{M}\frac{s_k}{d_k^5}\left[C_2 d_{k_x}\cos\psi - d_{k_x} G_x G_y \sin\psi + C_1^2 d_{k_y}\sin\psi\right]
\end{aligned}
\qquad (3.5.51)
$$

where

$$
d_{k_x} = \frac{(x - x_k) + (G - g_k)G_x}{d_k} \qquad (3.5.52)
$$

$$
d_{k_y} = \frac{(y - y_k) + (G - g_k)G_y}{d_k} \qquad (3.5.53)
$$

Finally equating (3.5.50) and (3.5.51), we will get a differential equation involving $\dot{\psi}$, which together with (3.4.32), defines the motion of the UAV over terrain. Hence, using an adjoint-control transformation, the optimal control problem solution was reduced to a search for the initial value of the heading angle. Two cases will be considered to get a differential equation for $\dot{\psi}$.

3.5.1 Case 1: $q_3 = 0$

In this case, there are no threats and the optimal trajectory is found to avoid the terrain while minimising the formulated performance index. Equating the

right hand sides of equations (3.5.50) and (3.5.51):

$$
\begin{aligned}
& q_2 G_x + \frac{1}{C_1^2 C_2^2}\left[q_1 + q_2 G\right] \\
\cdot\ & \left[C_2^2 G_x G_{xx}\cos^2\psi + C_2\left(C_1^2 G_y G_{xx} + C_1^2 G_x G_{xy} - 2G_x^2 G_y G_{xx}\right)\cos\psi\sin\psi\right. \\
+\ & \left.\left(C_1^2 G_x G_{xx} + C_1^2 G_y G_{xy} - C_2^2 G_x G_{xx}\right)\sin^2\psi\right] \\
=\ & \frac{1}{v(C_1^2 C_2)}\left[q_1 + q_2 G\right] \\
\cdot\ & \left[C_2 G_x G_{xx} v\cos^2\psi + (C_1^2 G_x G_{xy} - G_x^2 G_y G_{xx}) v\cos\psi\sin\psi - C_1^3 C_2\sin\psi\dot{\psi}\right] \\
+\ & \frac{q_2\cos\psi}{C_2}\left[C_2 G_x\cos\psi - G_x^2 G_y\sin\psi + C_1^2 G_y\sin\psi\right]
\end{aligned}
$$

Simplifying above equation

$$
\begin{aligned}
& v\left(q_1 + q_2 G\right)\left[C_2\left(C_1^2 G_y G_{xx} - G_x^2 G_y G_{xx}\right)\cos\psi\right. \\
& \qquad \left. + (C_1^2 G_x G_{xx} + C_1^2 G_y G_{xy} - C_2^2 G_x G_{xx})\sin\psi\right] \\
=\ & -C_1^3 C_2^2\left(q_1 + q_2 G\right)\dot{\psi} + q_2 v C_1^2 C_2\left[-C_2 G_x\sin\psi\right. \\
& \qquad \left. - G_x^2 G_y\cos\psi + C_1^2 G_y\cos\psi\right] \\
\Rightarrow\ & \dot{\psi} = \frac{A\cos\psi + B\sin\psi}{C} \qquad (3.5.54)
\end{aligned}
$$

where

$$
\begin{aligned}
A &= C_1^2 C_2 v q_2\left(C_1^2 G_y - G_x^2 G_y\right) - C_2 v(q_1 + q_2 G) G_y G_{xx} && (3.5.55)\\
B &= v(q_1 + q_2 G)\left(G_x G_y^2 G_{xx} - C_1^2 G_y G_{xy}\right) - C_1^2 C_2^2 v q_2 G_x && (3.5.56)\\
C &= C_1^3 C_2^2 (q_1 + q_2 G) && (3.5.57)
\end{aligned}
$$

3.5.2 Case 2: $q_2 = 0,\ G = 0,\ M = 1,\ s_1 = 1$

This is the case when the objective is to find the optimal minimum risk trajectory due to one radar without terrain and the UAV is constrained to fly in the $xy-$plane. Equating (3.5.49) and (3.5.51)

$$
-\frac{4q_3 d_x}{d^5} = \frac{1}{v}\left(q_1 + \frac{q_3}{d^4}\right)\left(-\sin\psi\dot{\psi}\right) - \frac{4q_3\cos\psi}{d^5}\left(d_x\cos\psi + d_y\sin\psi\right)
$$

Using the values of d_x and d_y from (3.5.52), (3.5.53) in the above equation and rearranging to find $\dot{\psi}$, we get

$$
\dot{\psi} = \frac{4q_3 v\left[(x - x_1)\sin\psi - (y - y_1)\cos\psi\right]}{d^2(q_1 d^4 + q_3)} \qquad (3.5.58)
$$

and when $q_1 = 0$, the above equation reduces to

$$\dot{\psi} = \frac{4v\left[(x - x_1)\sin\psi - (y - y_1)\cos\psi\right]}{d^2} \tag{3.5.59}$$

Next an analytical solution for a single radar risk minimization will be derived and discussed using the simplified equation (3.5.59).

3.6 Single Radar Risk Minimization

Consider a radar located at the origin as shown in Figure 3.3. It is desired to find the optimal aircraft trajectory that connects two prescribed points A and B in the plane such that the radio frequency energy reflected from the aircraft is minimized. Using the polar coordinates

$$x = R\cos\theta \tag{3.6.60}$$

$$y = R\sin\theta \tag{3.6.61}$$

in Figure 3.2, we have

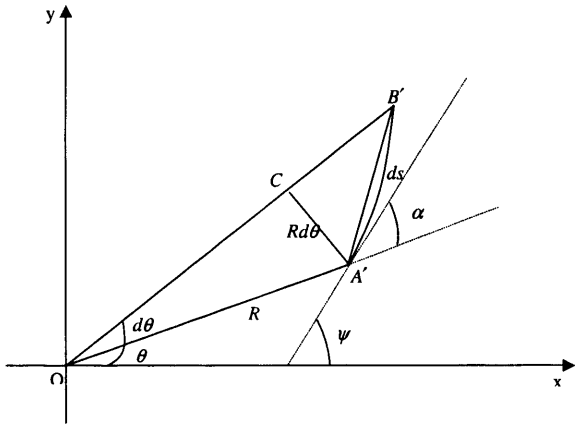


Figure 3.2: Relation between θ and ψ

$$\psi = \theta + \alpha \tag{3.6.62}$$

$$\sin\alpha = \frac{Rd\theta}{ds} \tag{3.6.63}$$

$$\cos\alpha = \frac{dR}{ds} \tag{3.6.64}$$

$$\tan\alpha = \frac{Rd\theta}{dR} \tag{3.6.65}$$

From these expressions and equation (3.5.59) we can obtain $\dot{\psi}$ as

$$\begin{aligned}\dot{\psi} &= \frac{4v[R\cos\theta\sin\psi - R\sin\theta\cos\psi]}{R^2} \quad \text{(by (3.6.60) and (3.6.61))} \\ &= \frac{4v\sin(\psi-\theta)}{R} \\ &= \frac{4v\sin\alpha}{R} \quad \text{(by (3.6.62))} \\ &= \frac{4vd\theta}{ds} \quad \text{(by (3.6.63))} \end{aligned} \tag{3.6.66}$$

This can be written as

$$d\psi = \frac{4vd\theta}{ds}dt \tag{3.6.67}$$

Now $v = \frac{ds}{dt}$, which implies $dt = \frac{ds}{v}$ and substituting into equation (3.6.67), we get the differential form of control law as

$$d\psi = 4d\theta \tag{3.6.68}$$

Integrating the above equation implies

$$\psi = 4\theta + C_1 \tag{3.6.69}$$

which is the optimal control law and is dependent on the angle θ. C_1 is the constant of integration to be determined. Putting the value of ψ from (3.6.62) in the optimal control law (3.6.69), we get an optimal relation between α and θ as

$$\alpha = 3\theta + C_1 \tag{3.6.70}$$

Taking the tangent of both sides

$$\begin{aligned}\tan(3\theta + C_1) &= \tan\alpha \\ &= \frac{Rd\theta}{dR} \quad \text{(by (3.6.65))} \end{aligned} \tag{3.6.71}$$

Rearranging the above equation, we have

$$\frac{1}{3}\frac{3\cos(3\theta + C_1)}{\sin(3\theta + C_1)}d\theta = \frac{dR}{R}$$

and integrating this gives

$$\ln C_2 + \frac{1}{3}\ln[\sin(3\theta + C_1)] = \ln R$$

$$\ln C_2[\sin(3\theta + C_1)]^{\frac{1}{3}} = \ln R$$

or

$$R = C_2[\sin(3\theta + C_1)]^{\frac{1}{3}} \tag{3.6.72}$$

After obtaining this expression for the optimal trajectory, the next step is to determine the two constants C_1 and C_2. When $R = R_f$, $\theta = \theta_f$ which implies

$$C_2 = \frac{R_f}{[\sin(3\theta_f + C_1)]^{\frac{1}{3}}} \tag{3.6.73}$$

Putting this value of C_2 back into equation (3.6.72) gives

$$R = R_f \left[\frac{\sin(3\theta + C_1)}{\sin(3\theta_f + C_1)}\right]^{\frac{1}{3}} \tag{3.6.74}$$

Also when $R = R_0$, $\theta = \theta_0$, so from the above equation

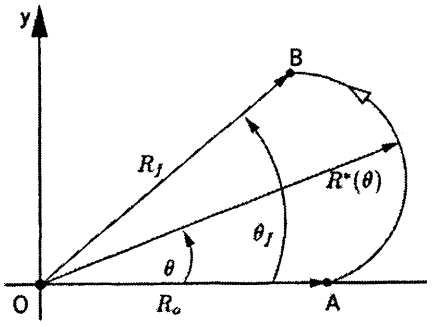


Figure 3.3: Optimal trajectory

$$\left[\frac{R_0}{R_f}\right]^3 = \frac{\sin(3\theta_0 + C_1)}{\sin(3\theta_f + C_1)}$$

From here we can write

$$\cos C_1 = \frac{\cos 3\theta_0 - \left(\frac{R_0}{R_f}\right)^3 \cos 3\theta_f}{\left(\frac{R_0}{R_f}\right)^3 \sin 3\theta_f - \sin 3\theta_0} \sin C_1 \tag{3.6.75}$$

Finally, the optimal trajectory is

$$R = R_f \left[\frac{\sin 3\theta \frac{\cos 3\theta_0 - \left(\frac{R_0}{R_f}\right)^3 \cos 3\theta_f}{\left(\frac{R_0}{R_f}\right)^3 \sin 3\theta_f - \sin 3\theta_0} + \cos 3\theta}{\sin 3\theta_f \frac{\cos 3\theta_0 - \left(\frac{R_0}{R_f}\right)^3 \cos 3\theta_f}{\left(\frac{R_0}{R_f}\right)^3 \sin 3\theta_f - \sin 3\theta_0} + \cos 3\theta_f}\right]^{\frac{1}{3}}$$

which can be simplified to

$$R = R_f \left[\frac{\sin(3\theta - 3\theta_0) - \left(\frac{R_0}{R_f}\right)^3 \sin(3\theta - 3\theta_f)}{\sin(3\theta_f - 3\theta_0)}\right]^{\frac{1}{3}} \tag{3.6.76}$$

For the special case of $\theta_0 = 0$,

$$R = R_f \left[\frac{\sin 3\theta - \left(\frac{R_0}{R_f}\right)^3 \sin(3\theta - 3\theta_f)}{\sin 3\theta_f} \right]^{\frac{1}{3}} \tag{3.6.77}$$

When $\sin 3\theta_f = 0$ or $\theta_f = \frac{\pi}{3}$, the equation (3.6.77) becomes undefined which means no optimal trajectory exists for this angle and a path length constraint or time constraint must be introduced to recover the solution. Hence to get an unconstrained solution for single radar exposure minimization, one must have $0 < \theta_f < \frac{\pi}{3}$. Figure 3.4 shows the optimal trajectory for $R_0 = R_f = 1$ and $\theta_f = 45^o$. The optimal cost while travelling the optimal trajectory can be

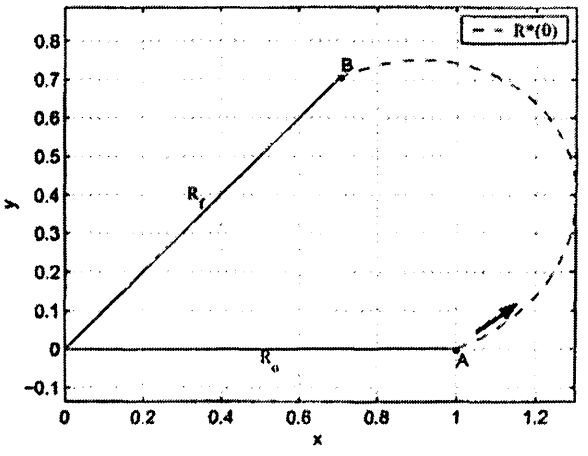


Figure 3.4: Optimal trajectory for $R_0 = R_f = 1$ and $\theta_f = 45^o$

derived by considering expression (3.5.34), which for this simplified case has the form

$$J = \int_0^{t_f} q_3 \frac{s}{d^4} dt \tag{3.6.78}$$

Taking $q_3 = 1$ and using polar coordinates

$$J = \frac{s}{v} \int_0^{\theta_f} \frac{\sqrt{R^2 + \frac{dR}{d\theta}}}{R^4} d\theta \tag{3.6.79}$$

From equation (3.6.71)

$$\frac{dR}{d\theta} = \frac{R}{\tan(3\theta + C_1)}$$

and therefore

$$\begin{aligned} R^2 + \left(\frac{dR}{d\theta}\right)^2 &= R^2 + \frac{R^2}{\tan^2(3\theta + C_1)} \\ &= \frac{R^2\left[\tan^2(3\theta + C_1) + 1\right]}{\tan^2(3\theta + C_1)} \\ &= \frac{R^2 \sec^2(3\theta + C_1)}{\tan^2(3\theta + C_1)} \\ &= \frac{R^2}{\sin^2(3\theta + C_1)} \end{aligned}$$

$$\Rightarrow \sqrt{R^2 + \left(\frac{dR}{d\theta}\right)^2} = \frac{R}{\sin(3\theta + C_1)} \tag{3.6.80}$$

Putting this in (3.6.79) and after simplifying

$$\begin{aligned} J &= \frac{s}{v}\int_0^{\theta_f} \frac{d\theta}{R^3 \sin(3\theta + C_1)} \\ &= \frac{s}{v}\int_0^{\theta_f} \frac{d\theta}{C_2^3 \sin(3\theta + C_1)\sin(3\theta + C_1)} \quad \text{(by (3.6.72))} \\ &= \frac{s}{vC_2^3}\int_0^{\theta_f} \csc^2(3\theta + C_1)d\theta \\ &= -\frac{s}{vC_2^3}|\cot(3\theta + C_1)|_0^{\theta_f} \\ &= \frac{s}{3vC_2^3}[\cot(C_1) - \cot(3\theta_f + C_1)] \\ &= \frac{s\sin(3\theta_f + C_1)}{3vR_f^3}[\cot(C_1) - \cot(3\theta_f + C_1)] \\ &= \frac{s\sin(3\theta_f)}{3vR_f^3\sin(C_1)} \end{aligned} \tag{3.6.81}$$

where C_1 is given in equation (3.6.75). Note that when θ_f approaches $\frac{\pi}{3}$, C_1 approaches 0 and hence equation (3.6.81) becomes indeterminate i.e., $\left(\frac{0}{0}\right)$. So in order to extract useful information about the existence of J we apply L'Hospital's rule. From equation (3.6.81)

$$\begin{aligned} J_{\theta_f=\frac{\pi}{3}} &= \lim_{\theta_f\to\frac{\pi}{3}} \frac{s\sin(3\theta_f)}{3vR_f^3\sin(C_1)} \\ &= \lim_{\theta_f\to\frac{\pi}{3}} \frac{\frac{d}{d\theta_f}s\sin(3\theta_f)}{\frac{d}{d\theta_f}3vR_f^3\sin(C_1)} \\ &= \lim_{\theta_f\to\frac{\pi}{3}} \frac{s\cos(3\theta_f)}{vR_f^3\cos(C_1)\frac{d}{d\theta_f}C_1} \\ &= -\lim_{\theta_f\to\frac{\pi}{3}} \frac{s}{vR_f^3\frac{d}{d\theta_f}C_1} \end{aligned} \tag{3.6.82}$$

Now differentiating equation (3.6.75) with respect to θ_f, we have

$$\sec^2 C_1 \frac{dC_1}{d\theta_f} = \frac{\left[1 - \left(\frac{R_0}{R_f}\right)^3 \cos 3\theta_f\right] 3\left(\frac{R_0}{R_f}\right)^3 \cos 3\theta_f - 3\left(\frac{R_0}{R_f}\right)^6 \sin^2 3\theta_f}{\left[1 - \left(\frac{R_0}{R_f}\right)^3 \cos 3\theta_f\right]^2}$$

from which

$$\lim_{\theta_f \to \frac{\pi}{3}} \frac{d}{d\theta_f} C_1 = -\frac{3\left(\frac{R_0}{R_f}\right)^3}{1 + \left(\frac{R_0}{R_f}\right)^3}$$

Finally, using this in (3.6.82), we get

$$J_{\theta_f = \frac{\pi}{3}} = \frac{s}{3v}\left[\frac{1}{R_0^3} + \frac{1}{R_f^3}\right] \tag{3.6.83}$$

This reveals that for $\theta_f \to \frac{\pi}{3}$, the trajectory approaches infinity but the optimal cost approaches a finite value. The expression for the optimal path length l^* is not in closed form. Numerical integration can be employed to work out this integral.

3.6.1 Special Cases

Several interesting special cases concerning the optimal trajectory given in (3.6.77) can now be considered. In the case when $\theta_f = 0$ and $R_f > R_0$, the origin O and the points A and B are co-linear and the optimal trajectory is a straight line as shown in Figure 3.5.

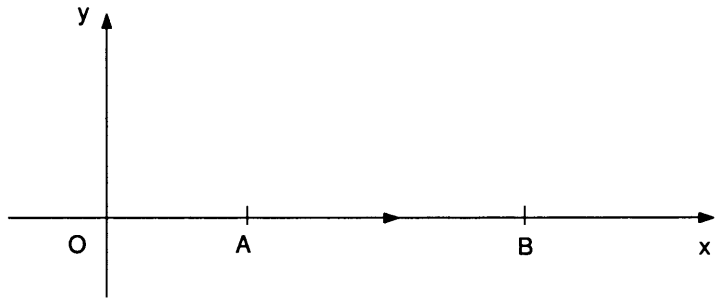


Figure 3.5: Optimal trajectory for the special case where $\theta_f = 0$

Theorem 2 *The optimal trajectory which connects points A and B at a distance $R_0 = R_f$ from the radar located at the origin O and minimizes exposure to the radar is*

$$R^*(\theta) = R_0 \sqrt[3]{\frac{\cos\left(3\theta - \frac{3\theta_f}{2}\right)}{\cos\left(\frac{3\theta_f}{2}\right)}}, \quad 0 < \theta \leq \theta_f \tag{3.6.84}$$

where θ_f is the angle $\angle AOB$. This result holds provided $0 < \theta_f < \frac{\pi}{3}$

Proof: When $R_f = R_0$, we can write (3.6.77) as

$$R = R_0 \left[\frac{\sin(3\theta) - \sin(3\theta - 3\theta_f)}{\sin(3\theta_f)}\right]^{\frac{1}{3}}$$

$$\text{or} R = R_0 \left[\frac{2\cos\left(\frac{3\theta+3\theta-3\theta_f}{2}\right)\sin\left(\frac{3\theta_f}{2}\right)}{2\sin\left(\frac{3\theta_f}{2}\right)\cos\left(\frac{3\theta_f}{2}\right)}\right]^{\frac{1}{3}}$$

$$\text{or} R = R_0 \sqrt[3]{\frac{\cos\left(3\theta - \frac{3\theta_f}{2}\right)}{\cos\left(\frac{3\theta_f}{2}\right)}}, \quad 0 < \theta \leq \theta_f$$

The optimal trajectory is shown in Figure 3.4 for the case where $\frac{R_f}{R_0} = 1$ and $\theta_f = 45°$. By inspection of Figure 3.4 we see that the extremal trajectory is indeed symmetric when $R_0 = R_f$ as expected.

3.7 Risk Minimization Two Radars

3.7.1 Problem Formulation and Scenarios

After understanding the nature of the one radar exposure minimization problem, the next stage is to extend the formulation to multiple radars. For this purpose, risk minimization problem due to two radars will be examined. An analytical solution as in the single radar case would be desirable but due to the complexity of the problem a numerical technique using gradient method will be employed to obtain optimal trajectories for different scenarios. The geometry of the possible locations of the radars and end points is shown in Figure 3.6. Three parameters were varied to examine the effects upon the optimal trajectories: the downrange distance between the radar locations (P), the crossrange

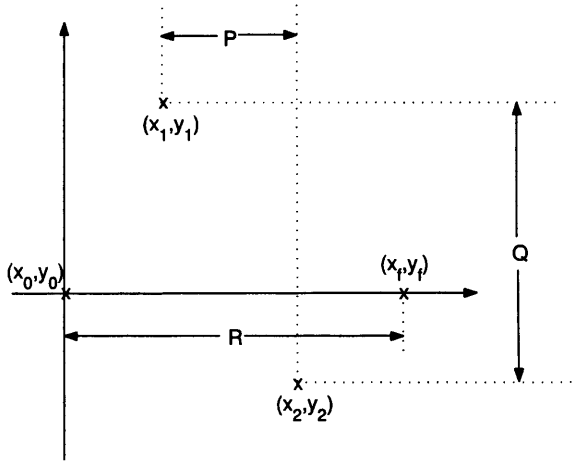


Figure 3.6: Possible locations of radars and end points

distance between the radar locations (Q) and the downrange distance between the initial and final points (R). When varying P and Q, the end points of the path were fixed at $(x_0, y_0) = (0, 0)$ and $(x_f, y_f) = (20, 0)$ and when varying R, the radars were fixed at $(x_1, y_1) = (5, 5)$ and $(x_2, y_2) = (15, -5)$. Two ratios of the radar transmission power were examined for each of the cases: $\alpha_1/\alpha_2 = 1/1$ and $\alpha_1/\alpha_2 = 2/1$. The ordinary or weighted Voronoi trajectory, as applicable, was computed to be compared with the optimal trajectory. Table 3.1 summarises the scenarios examined, where Δ is the incremental distance for the varying parameters P, Q and R. A common graphical technique for optimal

Scenario	P	Q	R	Range	Δ
1	Varied	Fixed	Fixed	$10 - 6.5$	0.5
2	Fixed	Varied	Fixed	$10 - 5.5$	0.5
3	Fixed	Fixed	Varied	$20 - 15.5$	0.5

Table 3.1: Scenarios for trajectory optimization against two radars

path planning against multiple radars is to make use of the Voronoi diagram. Starting with full knowledge of the radar locations, the Voronoi diagram is constructed of polygons whose edges are equidistant from all of the neighbouring radars. Hence, travel along the Voronoi edge ensures that an equal amount of power is reflected to each radar. This is true, however, only for the case

where the transmission powers of the radars are equal. When the radars have different transmission powers, i.e. $\alpha_1 \neq \alpha_2$, the Voronoi edge is no longer a line but a circular arc. It is easy to see that if the points have equal weight, the resulting locus is a circle of infinite radius or a line. Since this is a widely used path planning technique, it provides a useful comparison to the path length and objective cost of the calculated optimal trajectories.

3.7.2 Performance Index and its Discrete Approximation

For the case of a single flight against two radars, the power received by the radar is now considered a function of each radar's transmission power and range

$$P_r = P_t/d^4 \tag{3.7.85}$$

The geometry of the problem is shown in Figure 3.7, where $d_1(t)$ and $d_2(t)$ are the distances of a UAV at time t from radars located at (x_1, y_1) and (x_1, y_1) respectively. Cartesian coordinates will be used for the formulation of flight against two radar threats. The performance index is given by

$$J = \int_0^{\frac{l}{v}} \left[\frac{\alpha_1}{d_1(t)} + \frac{\alpha_2}{d_2(t)}\right] dt \tag{3.7.86}$$

If the optimal trajectory is considered to be made up of small line segments,

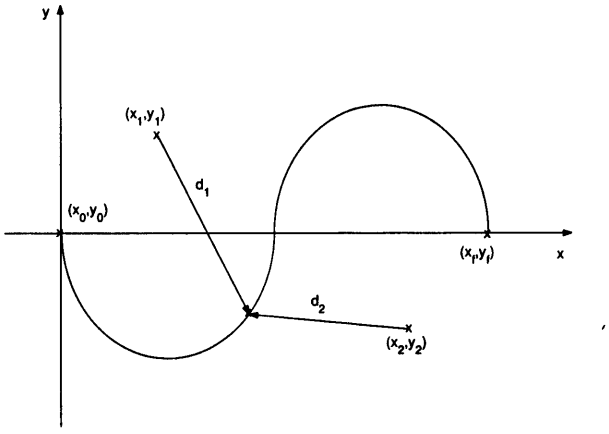


Figure 3.7: Geometry of two radar problem

then the approximate cost while travelling along the optimal trajectory can

be deduced from (3.3.25) by setting taking $M = 2$ and N according to the accuracy required.

3.8 Two Radar Case: Results and Comparison

This section examines the case of air vehicle flight against two radars. A short development of the camparison path is presented first, followed by the results of the optimization.

3.8.1 Comparison Path for Equal Power Radars

Given two equal power radars located at (x_1, y_1) and (x_2, y_2), the perpendicular bisector of the line segment connecting the radars will be the optimal path between these two radars. The equation of the line connecting the radars is

$$y(x) = m(x - x_2) + y_2 \tag{3.8.87}$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.8.88}$$

The perpendicular bisector of the line has slope $-1/m$ and passes through the midpoint (x_m, y_m) of the line

$$y_\perp(x) = -\frac{1}{m}(x - x_m) + y_m \tag{3.8.89}$$

$$x_m = \frac{x_1 + x_2}{2} \tag{3.8.90}$$

$$y_m = \frac{y_1 + y_2}{2} \tag{3.8.91}$$

The comparison path will be constructed from three line segments: the shortest path line from the initial point, the perpendicular bisector and the shortest path line to the final point completing the curve shown in Figure 3.8. The equation of the perpendicular from (x_0, y_0) to the Voronoi line (3.8.89) is

$$\tilde{y}(x) = m(x - x_0) + y_0 \tag{3.8.92}$$

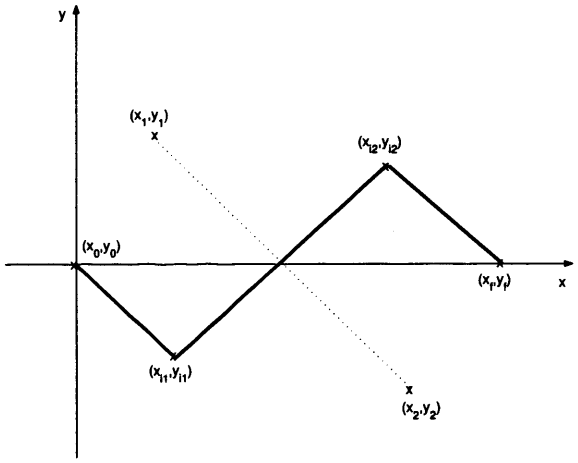


Figure 3.8: Comparison path for radars of equal transmission power, $\alpha_1 = \alpha_2$

The first intercept point (x_{i1}, y_{i1}) is a common point on the lines (3.8.89) and (3.8.92). Equating $\tilde{y}$ to equation (3.8.89) and solving for x yields

$$x_{i1} = \frac{m^2 x_0 + m(y_m - y_0) + x_m}{m^2 + 1} \tag{3.8.93}$$

Back substitution of x_{i1} into $\tilde{y}$ gives

$$y_{i1} = \frac{m^2 y_m + m(x_m - x_0) + y_0}{m^2 + 1} \tag{3.8.94}$$

Taking a similar approach at the final line segment results in

$$x_{i2} = \frac{m^2 x_f + m(y_m - y_f) + x_m}{m^2 + 1} \tag{3.8.95}$$

$$y_{i2} = \frac{m^2 y_m + m(x_m - x_f) + y_f}{m^2 + 1} \tag{3.8.96}$$

where m, x_m and y_m are given by (3.8.88), (3.8.90) and (3.8.91) respectively.

3.8.2 Comparison Path for Unequal Power Radars

For the case when the radars are of unequal power, a weighted path is used. The weighted path can found by equating the power received by each radar as follows

$$\frac{\alpha_1}{R_1^4} = \frac{\alpha_2}{R_2^4} \tag{3.8.97}$$

$$\Rightarrow \alpha_1 R_2^4 = \alpha_2 R_1^4$$

$$\sqrt{\alpha_1} R_2^2 = \pm\sqrt{\alpha_2} R_1^2 \tag{3.8.98}$$

where

$$R_1 = \sqrt{(x-x_1)^2+(y-y_1)^2} \tag{3.8.99}$$

$$R_2 = \sqrt{(x-x_2)^2+(y-y_2)^2} \tag{3.8.100}$$

Substituting these values of R_1 and R_2 into equation (3.8.98) yields

$$\sqrt{\alpha_1}\left[(x-x_2)^2+(y-y_2)^2\right] = \pm\sqrt{\alpha_2}\left[(x-x_1)^2+(y-y_1)^2\right] \tag{3.8.101}$$

First, considering the positive sign and rearranging the above equation, we can write

$$\begin{aligned}(1-\sqrt{\frac{\alpha_2}{\alpha_1}})x^2 - 2(x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1)x \;+\; & (1-\sqrt{\frac{\alpha_2}{\alpha_1}})y^2 - 2(y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1)y \\ = \;& \sqrt{\frac{\alpha_2}{\alpha_1}}x_1^2+\sqrt{\frac{\alpha_2}{\alpha_1}}y_1^2 - x_2^2 - y_2^2\end{aligned}$$

Dividing the above equation throughout by $1-\sqrt{\frac{\alpha_2}{\alpha_1}}$ gives

$$x^2 - 2\frac{x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}x + y^2 - 2\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}y = \frac{\sqrt{\frac{\alpha_2}{\alpha_1}}x_1^2+\sqrt{\frac{\alpha_2}{\alpha_1}}y_1^2 - x_2^2 - y_2^2}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}$$

Now completing the square by adding $\left[\frac{x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 + \left[\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2$ to the left and right hand sides of the above equation results in

$$\begin{aligned}\left[x-\frac{x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 + \left[\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 \;=\; & \frac{\sqrt{\frac{\alpha_2}{\alpha_1}}x_1^2+\sqrt{\frac{\alpha_2}{\alpha_1}}y_1^2 - x_2^2 - y_2^2}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}} \\ & + \left[\frac{x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 + \left[\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2\end{aligned}$$

$$\left[x-\frac{x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 + \left[y-\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 = \left[\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\right]^2 \tag{3.8.102}$$

Similarly by considering the case of the with negative sign in equation (3.8.101), we obtain the final expression

$$\left[x - \frac{x_2 + \sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1+\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 + \left[y - \frac{y_2 + \sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1+\sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 = \left[\frac{i\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1+\sqrt{\frac{\alpha_2}{\alpha_1}}}\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\right]^2 \tag{3.8.103}$$

Equations (3.8.102) and (3.8.103) are equations of circles but (3.8.103) has an imaginary radius and so will be discarded. Hence, the path between two radars of unequal powers is a circle with

$$\text{centre at} \quad \left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}, \frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\right) \tag{3.8.104}$$

$$\text{and radius} \quad \frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\sqrt{(x_2-x_1)^2+(y_2-y_1)^2} \tag{3.8.105}$$

The complete comparison path is obtained by joining the starting and final points of the path to the circle along the shortest distance lines as shown in Figure 3.9. This path will be used for comparison with other paths. The

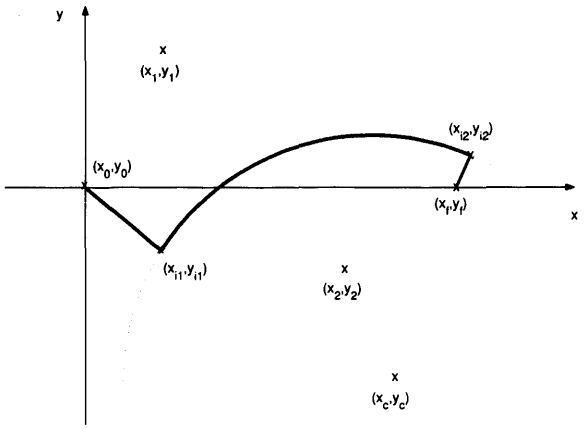


Figure 3.9: Comparison path for radars of unequal transmission powers

comparison path consists of three path segments: a shortest distance straight line path from the starting point (x_0, y_0) to the circle at (x_{i1}, y_{i1}), the circular path from (x_{i1}, y_{i1}) to (x_{i2}, y_{i2}) and the shortest distance straight line path from the destination point (x_f, y_f) to the circular segment at (x_{i2}, y_{i2}). The

intercept (x_{i1}, y_{i1}) will be along the line through (x_0, y_0) and (x_c, y_c) given by

$$y = y_0 + \frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1 - y_0\left(1 - \sqrt{\frac{\alpha_2}{\alpha_1}}\right)}{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1 - x_0\left(1 - \sqrt{\frac{\alpha_2}{\alpha_1}}\right)}(x - x_0) \tag{3.8.106}$$

To find the intersection point we substitute y from the above equation into (3.8.102) to give

$$\left[x - \frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2 + \left[y_0 + \frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1 - y_0\left(1 - \sqrt{\frac{\alpha_2}{\alpha_1}}\right)}{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1 - x_0\left(1 - \sqrt{\frac{\alpha_2}{\alpha_1}}\right)}(x - x_0) - \frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}}\right]^2$$
$$= \left[\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}\right]^2$$

Solving for x we obtain

$$x_{i1} = \frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} \pm \frac{\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_0\right)}{\sqrt{\left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_0\right)^2 + \left(\frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - y_0\right)^2}}$$

Since we are concerned with the path that passes between the radars, we select the negative sign and substitute this value of x in (3.8.106) to give

$$x_{i1} = \frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - \frac{\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_0\right)}{\sqrt{\left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_0\right)^2 + \left(\frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - y_0\right)^2}} \tag{3.8.107}$$

$$y_{i1} = \frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - \frac{\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \left(\frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - y_0\right)}{\sqrt{\left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_0\right)^2 + \left(\frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - y_0\right)^2}} \tag{3.8.108}$$

Similarly, the other intercept point (x_{i2}, y_{i2}) is given by

$$x_{i2} = \frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - \frac{\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_f\right)}{\sqrt{\left(\frac{x_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} x_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - x_f\right)^2 + \left(\frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}} y_1}{1 - \sqrt{\frac{\alpha_2}{\alpha_1}}} - y_f\right)^2}} \tag{3.8.109}$$

$$y_{i2} = \frac{y_2 - \sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}} - \frac{\frac{\sqrt{\sqrt{\frac{\alpha_2}{\alpha_1}}}}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}}\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\left(\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}} - y_f\right)}{\sqrt{\left(\frac{x_2-\sqrt{\frac{\alpha_2}{\alpha_1}}x_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}} - x_f\right)^2 + \left(\frac{y_2-\sqrt{\frac{\alpha_2}{\alpha_1}}y_1}{1-\sqrt{\frac{\alpha_2}{\alpha_1}}} - y_f\right)^2}} \tag{3.8.110}$$

3.8.3 Scenario 1: Varying Downrange Between Radars

The first case explores changing the downrange location of the radars and the effect on the optimal trajectories. Eight optimal trajectories were produced corresponding to eight different symmetric radar geometries. A summary of the optimal trajectory cost J^* (obtained using the gradient method and the analytical relation of the discrete approximation of the performance index) and path length l^*, the comparison path cost J_{vor} and path length l_{vor} and the straight line cost J_{line} and path length l_{line} is presented in Table 3.2. Intuitively, it is expected that the paths will be symmetric about the midpoint of the line connecting the radars and the path will bend away from the nearest radar. The results of the optimization do in fact prove this to be true. In Figure 3.10, the optimal path bends away from the nearest radar and intersects the downrange axis at the mid point of the radars as the radars move towards the endpoints. From Table 3.2, the optimal path length increases as P increases. From Figure 3.10, it can be inferred that when the radars are at the same x-coordinate, the path length will be the shortest than the previous trajectories and is a straight line. The variation of the optimal costs with P are shown in Figure 3.12. Since the gradient method was used to calculate the optimal trajectories, the cost is determined at discrete points ignoring contributions between the segments of these points. This is why the cost by the gradient method is lower than for others. The largest cost arises from the direct line path between the end points. The analytical expression (3.3.25), after simplifying for the two radar case, was used to calculate the exact optimal cost for the optimal trajectory obtained from the gradient method. By comparing this discrete optimal cost with comparison path cost, one can see from Figure 3.12 that they are almost equal.

P	J^*_{grad}	J^*_{dis}	l^*	J_{comp}	l_{comp}	J_{line}	l_{line}
10	0.0108	0.0217	24.8883	0.0231	28.2843	0.0454	20
9.5	0.0119	0.0238	24.3770	0.0247	28.2452	0.0460	20
9	0.0131	0.0262	23.8328	0.0269	28.1113	0.0466	20
8.5	0.0146	0.0291	23.1759	0.0297	27.8539	0.0470	20
8	0.0162	0.0324	22.4685	0.0329	27.4398	0.0474	20
7.5	0.0180	0.0360	21.8474	0.0365	26.8328	0.0477	20
7	0.0198	0.0396	21.2407	.0.0402	25.9973	0.0479	20
6.5	0.0215	0.0430	20.7224	0.0436	24.9035	0.0480	20

Table 3.2: Calculated data for two equal power radars for scenario 1

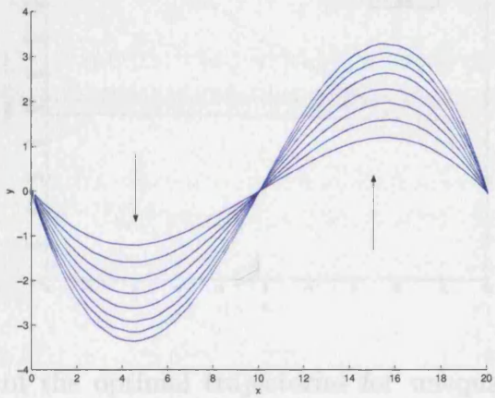


Figure 3.10: Optimal trajectories for equal power radars for scenario 1

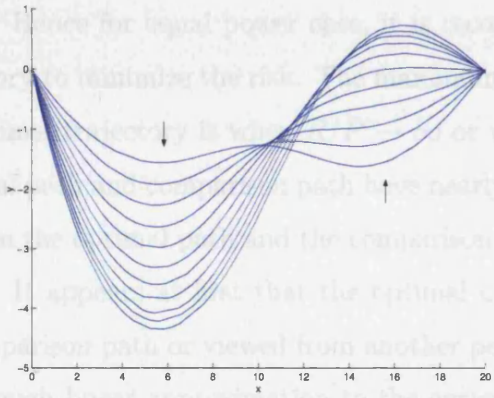


Figure 3.11: Optimal trajectories for unequal power radars for scenario 1

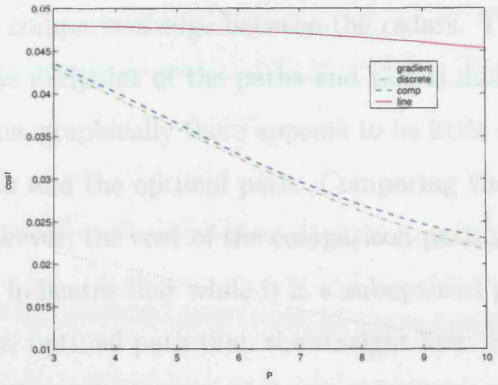


Figure 3.12: Cost of the optimal trajectories for equal power radars for scenario 1

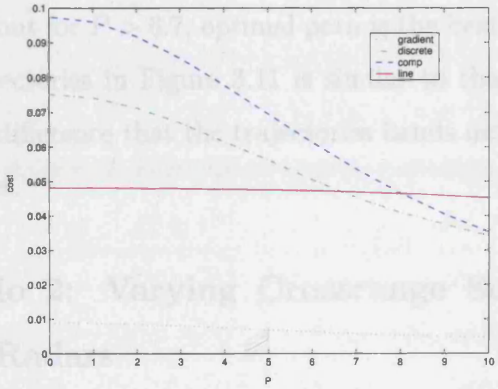


Figure 3.13: Cost of the optimal trajectories for unequal power radars for scenario 1

The costs decrease as P increases and this decrease is much steeper for the optimal trajectory. Hence for equal power case, it is recommended to follow the optimal trajectory to minimize the risk. The maximum value of the objective cost of the optimal trajectory is when $R/P \to \infty$ or when $P \to 0$. Since for this case, optimal path and comparison path have nearly equal cost but the relationship between the optimal path and the comparison path is not evident from this scenario. It appears at first that the optimal curve is a smoothed function of the comparison path or viewed from another perspective, the comparison path is a rough linear approximation to the curve. In the limit as P gets very large, however the comparison edge will become perpendicular to the optimal path and the comparison path intercepts will move to a single point,

the midpoint of the comparison edge between the radars. The only consistent common point is the midpoint of the paths and this is due to the symmetry of the problem. Thus, graphically there appears to be little similarity between the comparison path and the optimal path. Comparing the objective cost of the three paths, however, the cost of the comparison path is very close to the optimal path. This indicates that while it is a suboptimal path, it is a better approximation of the optimal path than the straight line. For case of unequal power radars when $P \leq 6.7$, the cost of the straight line path is less than the the optimal path cost due to gradient method and also it is less than comparison path cost. But for $P > 6.7$, the optimal path is the least risk path. Hence in this case it is recommended to follow straight line path for downrange separation of $P \leq 6.7$ but for $P > 6.7$, optimal path is the best choice. The trend of the optimal trajectories in Figure 3.11 is similar to that shown in Figure 3.10 with the only difference that the trajectories bends more near that radar having higher power.

3.8.4 Scenario 2: Varying Crossrange Separation Between Radars

In this case, the radars are kept at a fixed downrange separation P while the crossrange Q is progressively varied. Results similar to those observed in scenario 1 are expected and are tabulated in Table 3.3. As the crossrange $Q \rightarrow \infty$, the optimal path will approach a straight line. Indeed, this is observed in the optimal trajectories for this case as shown in Figures 3.14, 3.15. For this scenario, numerical difficulties preempted finding solutions as $Q \rightarrow 0$. This is likely to be due to the optimal path wanting to travel around the radars instead of between them. The path length of the optimal trajectories increases sharply for a small decrease in Q around 5.5. In reality, the vehicle would never follow this path, instead, it would travel around the radars at a much lower cost. From Figures 3.16, 3.17, inferences can still be drawn by observing the affects of varying Q. As Q is very small the cost to travel along the direct line path will be very high. This fact is also evident from the comparison path but the cost in

that case is slightly higher than the optimal. The optimal costs are very small whether calculated from the gradient method or using the optimal trajectory and the discrete approximation of cost. When Q is increased, the optimal cost J^*_{dis} will go to zero and the optimal path will become a straight line. In addition, the comparison path will also flatten to a straight line. The results of the first two scenarios follow our expectations of how the optimal trajectory should react to different radar geometries, and reinforce our understanding

Q	J^*_{grad}	J^*_{dis}	l^*	J_{comp}	l_{comp}	J_{line}	l_{line}
10	0.0108	0.0217	0.25×10^6	0.0231	28.2843	0.0454	20
9.5	0.0129	0.0258	0.26×10^6	0.0284	28.2452	0.0634	20
9	0.0154	0.0307	0.28×10^6	0.0355	28.1113	0.0920	20
8.5	0.0182	0.0363	0.29×10^6	0.0454	27.8539	0.1396	20
8	0.0213	0.0427	0.32×10^6	0.0606	27.4398	0.2250	20
7.5	0.0253	0.0506	0.47×10^6	0.0865	26.8328	0.3936	20
7	0.0300	0.0600	1.02×10^6	.0.1395	25.9973	0.7761	20
6.5	0.0344	0.0689	2.77×10^6	0.2801	24.9035	1.8517	20
6	0.0363	0.0727	12.12×10^6	0.8432	23.5339	6.2726	20
5.5	0.0449	0.0898	136.24×10^6	6.3788	21.8908	50.2545	20

Table 3.3: Calculated data for two equal power radars for scenario 2

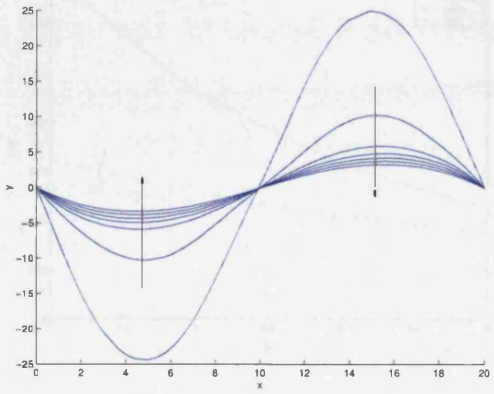


Figure 3.14: Optimal trajectories for equal power radars for scenario 2

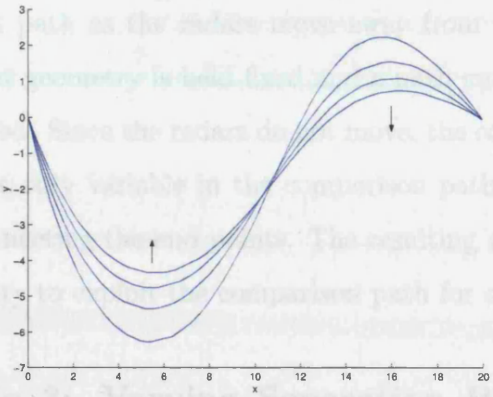


Figure 3.15: Optimal trajectories for unequal power radars for scenario 2

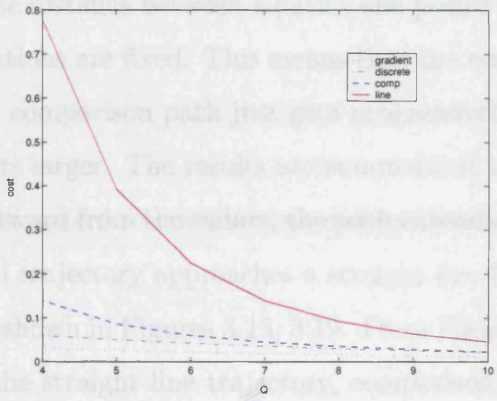


Figure 3.16: Cost of the optimal trajectories for equal power radars for scenario 2

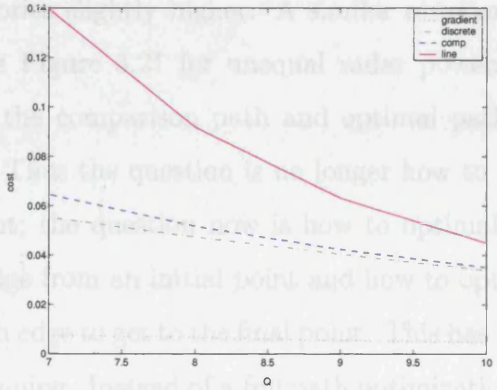


Figure 3.17: Cost of the optimal trajectories for unequal power radars for scenario 2

that the path will bend away from the radars when they are close and

approach the direct path as the radars move away from the end points. In scenario 3, the radar geometry is held fixed and a path end points separation is progressively varied. Since the radars do not move, the comparison edge will be constant and the only variable in the comparison path will be the length of the segments connecting the end points. The resulting optimal trajectories reveal an opportunity to exploit the comparison path for on-line utilization.

3.8.5 Scenario 3: Varying Separation Between Initial and Final Positions

For this scenario, the distance between a path's end points is iteratively varied while the radar locations are fixed. This means that the comparison path edge is constant and the comparison path just gets progressively longer as the end point separation gets larger. The results are summarized in Table 3.4. As the end points move outward from the radars, the path extends further and further out and the optimal trajectory approaches a straight line for a very small end point separation as shown in Figures 3.18, 3.19. From Figure 3.20, it is evident that the costs for the straight line trajectory, comparison trajectory and the optimal trajectory are nearly the same for a variation in R up to 8.4. After that there is a big increase in the cost of the straight line trajectory while the costs of the optimal and comparison trajectories remain almost equal, with the comparison trajectories slightly higher. A similar observation cane be made by looking into the Figure 3.21 for unequal radar powers. This very small difference between the comparison path and optimal path remains constant for all values of R. Thus the question is no longer how to get from the initial point the final point; the question now is how to optimally approach to the comparison path edge from an initial point and how to optimally depart from the comparison path edge to get to the final point. This has crucial implications for on-line path planning. Instead of a full path optimization being performed, utilizing valuable on-line system resources, one only needs to optimize the approach to and departure from the comparison edge. The total cost of the optimal trajectory at the discrete points are also drawn. This is very small

with compared to all other cost as already predicted. By further increasing R, the cost values for both optimal and comparison trajectories approach a constant value of 0.02.

Q	J^*_{grad}	J^*_{dis}	l^*	J_{comp}	l_{comp}	J_{line}	l_{line}
20	0.0108	0.0217	24.8883	0.0231	28.2843	0.0454	20
19.5	0.0110	0.0220	23.7390	0.0235	26.8701	0.0444	19
19	0.0111	0.0223	22.4924	0.0240	25.4558	0.0433	18
18.5	0.0113	0.0225	21.1851	0.0244	24.0416	0.0420	17
18	0.0114	0.0228	19.8742	0.0247	22.6274	0.0403	16
17.5	0.0114	0.0229	18.4999	0.0249	21.2132	0.0384	15
17	0.0115	0.0229	17.0428	0.0248	19.7990	0.0361	14
16.5	0.0113	0.0226	15.6273	0.0246	18.3848	0.0335	13
16	0.0111	0.0221	14.1197	0.0240	16.9706	0.0306	12
15.5	0.0106	0.0213	12.5921	0.0230	15.5563	0.0274	11

Table 3.4: Calculated results for two equal power radars for scenario 3

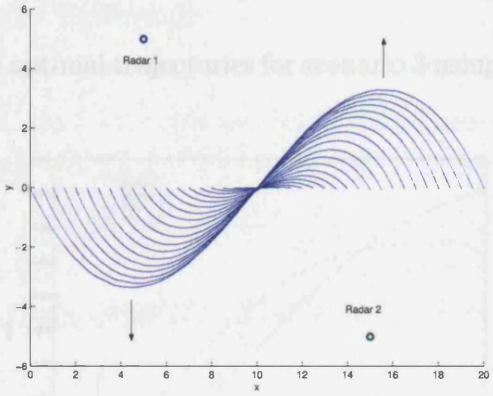


Figure 3.18: Optimal trajectories for scenario 3 using equal radar powers

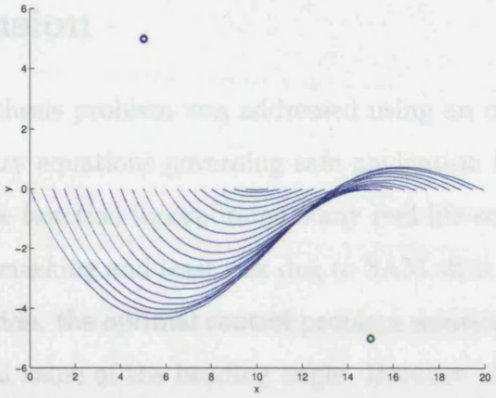


Figure 3.19: Optimal trajectories for scenario 3 using unequal radar powers

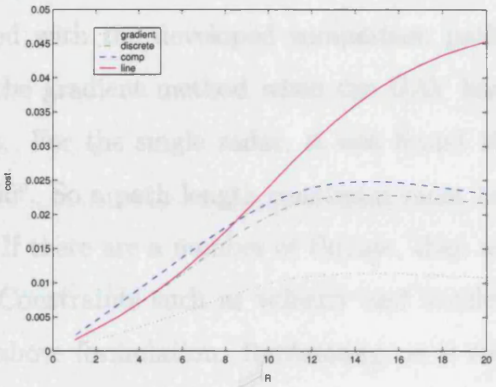


Figure 3.20: Cost of optimal trajectories for scenario 3 using equal radar powers

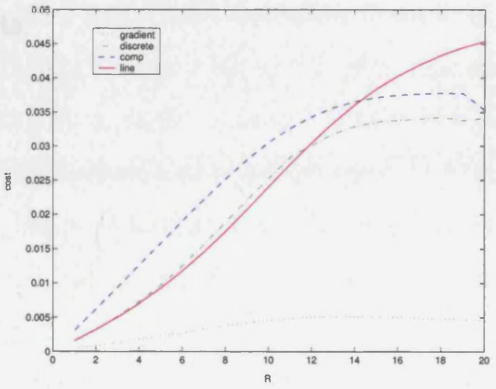


Figure 3.21: Cost of optimal trajectories for scenario 3 using unequal radar powers

3.9 Conclusion

The trajectory synthesis problem was addressed using an optimal control approach and necessary equations governing safe navigation for UAVs were derived. The objective function incorporates many real life scenarios: minimum flight time, terrain masking and least risk due to SAM sites. Using an adjoint-control transformation, the optimal control problem solution was reduced to a search for the initial value of the heading angle. Because of the complexity of the problem, it was simplified into two radar risk minimization problems, one with a single radar and the other with two radars. An analytical solution was obtained for the single radar case. The two radar case for different strength ratios was compared with the developed comparison paths using numerical technique such as the gradient method when the UAV has to sought a path between the radars. For the single radar, it was found that trajectories do not exist for $\theta_f \geq 60^o$. So a path length constraint must be introduced to recover the solution. If there are a number of threats, then an analytic solution becomes difficult. Constraints such as velocity and acceleration are hard to incorporate in the above formulation. Replanning, as is required in real time situations must also be considered.

In the next chapter, we will present a mixed integer linear formulation technique and formulate it in such a way that will be useful for real scenarios satisfying constraints.

Chapter 4

MILP and its Application in Flight Path Planning

4.1 Introduction

If the problem of path planning can be written as a linear program with mixed integer/linear constraints, then it can be solved using commercially available software AMPL/CPLEX. The optimization problem can easily translated into the AMPL modelling language [33]. The structure of the problem and constraints are written into an AMPL model file while the data is written to an AMPL data file. The data file can be easily edited directly or generated by simple Matlab scripts. The CPLEX optimizer is used to solve the problem [48]. A series of scripts in Matlab and AMPL allow the entire path planning problem to be invoked by a single command. Other Matlab scripts then plot the path and visualize the state and input sequence. A linear aircraft model is needed in this formulation. The next section describes a simple aircraft model for a point mass which will be used in the formulation. This chapter then describes the modelling of different mixed integer/linear constraints using binary variables and also presents simulation results using CPLEX as an optimizer.

4.2 Model of the Aircraft

The model used in section 3.4 is nonlinear and can not be used within MILP framework. As an alternate, the point mass dynamics of a UAV subject to two norm constraints form an approximate model for limited turn rate vehicles provided that the optimization favours the minimum time or minimum distance path [86]. The UAV dynamics are expressed as a simple point mass with positions and velocities $[x, y, v_x, v_y]^T$ as state variables and accelerations $[u_x, u_y]^T$ as control inputs:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} u_x \\ u_y \end{bmatrix}$$

or

$$\dot{\mathbf{s}} = A_c\mathbf{s} + B_c\mathbf{u} \tag{4.2.1}$$

The zero-order hold equivalent discrete time system is

$$\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_{i+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_k + \begin{bmatrix} (\Delta t)^2/2 & 0 \\ 0 & (\Delta t)^2/2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}\begin{bmatrix} u_x \\ u_y \end{bmatrix}_i$$

or

$$\mathbf{s}_{i+1} = A\mathbf{s}_i + B\mathbf{u}_i \tag{4.2.2}$$

Where i is the time step and Δt is the time interval between two time steps. The control input $[u_x, u_y]_i^T$ stays constant over each time interval Δt under the zero-order hold assumption.

4.3 Constraints to Avoid Radar Zones

The radar areas are modeled as rectangles with $(x_{min}^{rad}, y_{min}^{rad})$ and $(x_{max}^{rad}, y_{max}^{rad})$ as the coordinates of the lower left and upper right corner points of the zone.

At each time step i the position (x_i, y_i) of the vehicle must lie in the area outside of the radar zone which can be formulated as:

$$\begin{aligned} \forall i \in [1, \ldots, N] : x_i &\leq x_{min}^{rad} \\ \text{or} \quad x_i &\geq x_{max}^{rad} \\ \text{or} \quad y_i &\leq y_{min}^{rad} \\ \text{or} \quad y_i &\geq y_{max}^{rad} \end{aligned} \tag{4.3.3}$$

The or-constraints can be transformed to and-constraints by the introduction of binary slack variables[98]. Let b_{ik}^{rad} be a binary variable (0 or 1) and let Ω^{rad} be a large arbitrary positive number. Then the above constraints (4.3.3) can be replaced by the following mixed integer/linear constraints [89]:

$$\begin{aligned} \forall i \in [1, \ldots, N] : x_i &\leq x_{min}^{rad} + \Omega^{rad} b_{i1}^{rad} \\ \text{and} \quad -x_i &\leq -x_{max}^{rad} + \Omega^{rad} b_{i2}^{rad} \\ \text{and} \quad y_i &\leq y_{min}^{rad} + \Omega^{rad} b_{i3}^{rad} \\ \text{and} \quad -y_i &\leq -y_{max}^{rad} + \Omega^{rad} b_{i4}^{rad} \\ \sum_{k=1}^{4} b_{ik}^{rad} &\leq 3 \\ b_{ik}^{rad} &= 0 \text{ or } 1 \end{aligned} \tag{4.3.4}$$

If their are N_V vehicles and N_R stationary radars, equation (4.3.4) can be generalized as:

$$\begin{aligned} &\forall p \in [1, \ldots, N_V], \forall c \in [1, \ldots, N_R], \forall i \in [1, \ldots, N] : \\ &\qquad x_{pi} \leq x_{c,min}^{rad} + \Omega^{rad} b_{pci1}^{rad} \\ &\text{and} \quad -x_{pi} \leq -x_{c,max}^{rad} + \Omega^{rad} b_{pci2}^{rad} \\ &\text{and} \quad y_{pi} \leq y_{c,min}^{rad} + \Omega^{rad} b_{pci3}^{rad} \\ &\text{and} \quad -y_{pi} \leq -y_{c,max}^{rad} + \Omega^{rad} b_{pci4}^{rad} \\ &\qquad \sum_{k=1}^{4} b_{pcik}^{rad} \leq 3 \\ &\qquad b_{pcik}^{rad} = 0 \text{ or } 1 \end{aligned} \tag{4.3.5}$$

This radar avoidance technique is not restricted to rectangular planar geometry since any arbitrary shaped planar radar zone can be described by a surrounding

polygon of which the edges give rise to anti-collision constraints in both x and y coordinates. The extension to 3D motion with 3D radar zones is straightforward. We only need to formulate extra constraints in the z coordinate and a 3D radar area can be described by a surrounding polyhedron. A moving SAM unit with predefined motion can also be considered. The coordinates of the radar zone change at every time step according to its predefined motion. The straightforward approach is to define new forbidden regions at every time step corresponding to the positions of the moving SAM unit at that instant. For translational motion of rectangular zones, the format of binary constraints (4.3.4) remains the same. Rotational motion on the other hand and motion of non-rectangular zones yields constraints in both x and y coordinates.

For overlapping zones, a receding horizon approach may not find a feasible solution in the face of such hard constraints. In this case, the hard constraints can be transformed to soft constraints by the introduction of small variables as in the following:

$$
\begin{aligned}
&\forall p \in [1,\ldots,N_V], \forall c \in [1,\ldots,N_R], \forall i \in [1,\ldots,N]: \\
&\qquad x_{pi} \le x^{rad}_{c,min}(1-m^{rad}_{pci1}) + \Omega^{rad} b^{rad}_{pci1} \\
&\text{and} \quad -x_{pi} \le -x^{rad}_{c,max}(1-m^{rad}_{pci2}) + \Omega^{rad} b^{rad}_{pci2} \\
&\text{and} \quad y_{pi} \le y^{rad}_{c,min}(1-m^{rad}_{pci3}) + \Omega^{rad} b^{rad}_{pci3} \\
&\text{and} \quad -y_{pi} \le -y^{rad}_{c,max}(1-m^{rad}_{pci4}) + \Omega^{rad} b^{rad}_{pci4} \\
&\qquad \sum_{k=1}^{4} b^{rad}_{pcik} \le 3 \\
&\qquad\qquad 0 \le m^{rad}_{pci1},\ m^{rad}_{pci2},\ m^{rad}_{pci3},\ m^{rad}_{pci4} \le 1
\end{aligned}
\tag{4.3.6}
$$

where m^{rad}_{pci1}, m^{rad}_{pci2}, m^{rad}_{pci3}, m^{rad}_{pci4} are very small decision variables between 0 and 1. The idea is to reduce these variables to zero by incorporating them into the objective (cost) function. The problem formulation returns to the original setting when these $m's$ are zero. If it is not possible to reduce them to zero, i.e. the original hard constraints cannot be satisfied, the algorithm will have have the flexibility to generate solutions which violate these constraints as little as possible. An alternative formulation of the above constraints relevant to radar

avoidance is the following:

$$\begin{aligned}
&\forall p \in [1,\ldots,N_V], \forall r \in [1,\ldots,N_R], \forall i \in [1,\ldots,N]: \\
&\qquad x_{pi} - x_r^{rad} \geq R_r^{rad}(1 - m_{pri}^{rad}) - \Omega^{rad} b_{pri1}^{rad} \\
&\text{and} \quad x_r^{rad} - x_{pi} \geq R_r^{rad}(1 - m_{pri}^{rad}) - \Omega^{rad} b_{pri2}^{rad} \\
&\text{and} \quad y_{pi} - y_r^{rad} \geq R_r^{rad}(1 - m_{pri}^{rad}) - \Omega^{rad} b_{pri3}^{rad} \\
&\text{and} \quad y_r^{rad} - y_{pi} \geq R_r^{rad}(1 - m_{pri}^{rad}) - \Omega^{rad} b_{pri4}^{rad} \\
&\qquad \sum_{k=1}^{4} b_{prik}^{rad} \leq 3 \\
&\qquad b_{prik}^{rad} = 0 \text{ or } 1 \\
&\qquad 0 \leq m_{pri}^{rad} \leq 1
\end{aligned} \tag{4.3.7}$$

where N_R is the total number of radars and (x_r^{rad}, y_r^{rad}), R_r^{rad} are the position and radius of the r^{th} radar. So either (4.12.36) as discussed later, or (4.3.7) can be used to avoid danger zones.

4.4 Collision Avoidance Constraints

In the case of a fleet of UAVs, it is obviously desirable to consider collision avoidance in path planning. Collision avoidance between vehicles can be dealt with in a way similar to obstacle avoidance. At each time step every pair of vehicles p and q must be a minimum distance apart from each other in the x or/and y directions. If at the i^{th} time step, we let (x_{pi}, y_{pi}) and (x_{qi}, y_{qi}) be the positions of the vehicles p and q respectively and d_x^{col} and d_y^{col} are the safety distances in the x and y directions, then the collision avoidance constraint can be written as:

$$\begin{aligned}
&\forall i \in [1,\ldots,N],\ \forall p,\ q | q > p: \\
&\qquad |x_{pi} - x_{qi}| \geq d_x^{col} \\
&\text{or} \qquad |y_{pi} - y_{qi}| \geq d_y^{col}
\end{aligned} \tag{4.4.8}$$

The condition $q > p$ avoids duplication of the constraints on the positions. Alternatively we can write:

$$\forall i \in [1,\ldots,N],\ \forall p,\ q | q > p:$$

$$
\begin{aligned}
& x_{pi} - x_{qi} \geq d_x^{col} \\
\text{or} \quad & x_{qi} - x_{pi} \geq d_x^{col} \\
\text{or} \quad & y_{pi} - y_{qi} \geq d_y^{col} \\
\text{or} \quad & y_{qi} - y_{pi} \geq d_y^{col}
\end{aligned}
\tag{4.4.9}
$$

These constraints can be formulated as mixed integer/linear constraints by introducing appropriate binary variables b_{pqik}^{col} [89]:

$$
\begin{aligned}
& \forall i \in [1, \ldots, N], \forall p \in [1, \ldots, N_V], \forall q \in [p+1, \ldots, N_V] : \\
& \qquad x_{pi} - x_{qi} \geq d_x^{col} - \Omega^{col} b_{pqi1}^{col} \\
& \text{and} \quad x_{qi} - x_{pi} \geq d_x^{col} - \Omega^{col} b_{pqi42}^{col} \\
& \text{and} \quad y_{pi} - y_{qi} \geq d_y^{col} - \Omega^{col} b_{pqi3}^{col} \\
& \text{and} \quad y_{qi} - y_{pi} \geq d_y^{col} - \Omega^{col} b_{pqi4}^{col} \\
& \qquad b_{pqik}^{col} = 0 \text{ or } 1
\end{aligned}
\tag{4.4.10}
$$

Similarly these constraints can be converted to soft constraint by the introduction of small variables:

$$
\begin{aligned}
& \forall i \in [1, \ldots, N], \forall p \in [1, \ldots, N_V], \forall q \in [p+1, \ldots, N_V] : \\
& \qquad x_{pi} - x_{qi} \geq d_x^{col}(1 - m_{pqi1}^{col}) - \Omega^{col} b_{pqi1}^{col} \\
& \text{and} \quad x_{qi} - x_{pi} \geq d_x^{col}(1 - m_{pqi1}^{col}) - \Omega^{col} b_{pqi2}^{col} \\
& \text{and} \quad y_{pi} - y_{qi} \geq d_y^{col}(1 - m_{pqi2}^{col}) - \Omega^{col} b_{pqi3}^{col} \\
& \text{and} \quad y_{qi} - y_{pi} \geq d_y^{col}(1 - m_{pqi2}^{col}) - \Omega^{col} b_{pqi4}^{col} \\
& \qquad \sum_{k=1}^{4} b_{pqik}^{col} \leq 3 \\
& \qquad b_{pqik}^{col} = 0 \text{ or } 1 \\
& \qquad 0 \leq m_{pqi1}^{col},\ m_{pqi2}^{col} \leq 1
\end{aligned}
\tag{4.4.11}
$$

4.5 Speed and Acceleration Constraints

The maximum speed v_{max} is enforced by an approximation to a circular region in the velocity plane [84, 86]. The velocity vector is projected to different directions to obtain

$$
\forall m \in [1, \ldots, N_C^v], \forall i \in [1, \ldots, N], \forall p \in [1, \ldots, N_V] :
$$

$$\dot{x}_{pi}\cos(\frac{2\pi m}{N_C^v}) + \dot{y}_{pi}\sin(\frac{2\pi m}{N_C^v}) \leq v_{max} \tag{4.5.12}$$

The above constraints require that the velocity vector be inside a regular polygon with N_C^v sides circumscribed about a circle of radius v_{max}. A constraint on the minimum speed can be expressed in a similar way. However, it is different from the maximum speed constraint in that at least one of the constraints must be active instead of all of them:

$$\begin{gathered}\exists m \in [1,\ldots,N_C^v], \forall i \in [1,\ldots,N], \forall p \in [1,\ldots,N_V]: \\ \dot{x}_{pi}\cos(\frac{2\pi m}{N_C^v}) + \dot{y}_{pi}\sin(\frac{2\pi m}{N_C^v}) \geq v_{min}\end{gathered} \tag{4.5.13}$$

where N_C^v is the order of the discretization of the circle. Equation (4.5) is a non-convex constraint and can be written as mixed integer/linear constraints:

$$\begin{gathered}\forall m \in [1,\ldots,N_C^v], \forall i \in [1,\ldots,N], \forall p \in [1,\ldots,N_V]: \\ \dot{x}_{pi}\cos(\frac{2\pi m}{N_C^v}) + \dot{y}_{pi}\sin(\frac{2\pi m}{N_C^v}) \geq v_{min} - \Omega^v(1 - b_{pim}^v) \\ \sum_{m=1}^{N_C^v} b_{pim}^v \geq 1\end{gathered} \tag{4.5.14}$$

Similarly, the constraint for the upper bound on acceleration can be written as:

$$\begin{gathered}\forall m \in [1,\ldots,N_C^u], \forall i \in [1,\ldots,N], \forall p \in [1,\ldots,N_V]: \\ \ddot{x}_{pi}\cos(\frac{2\pi m}{N_C^u}) + \ddot{y}_{pi}\sin(\frac{2\pi m}{N_C^u}) \leq u_{max}\end{gathered} \tag{4.5.15}$$

4.6 Turning Rate Constraints

The minimum constraints on speed and acceleration involve binary variables which make the optimization hard. One way to get rid of these constraints is to assume that the UAV moves with constant speed but there is a minimum turning radius bound r_{min} which satisfies

$$r_{min} \leq r = \frac{v^2}{u} \tag{4.6.16}$$

This constraint turns out to be a constraint on the maximum magnitude of lateral acceleration as

$$u \leq \frac{v^2}{r_{min}} = u_{max} \tag{4.6.17}$$

where u is the magnitude of the acceleration vector and u_{max} is the maximum acceleration magnitude. To enforce minimum turning radius constraint in a linear framework, one can use only (4.6.17), where $u_{max} = \frac{v^2}{r_{min}}$.

4.7 Vehicle Capabilities and Time Dependency Constraints

The set of constraints to detect if a vehicle visits a waypoint can be written as

$$\begin{aligned} &\forall p \in [1,\ldots,N_V], \forall i \in [1,\ldots,N], \forall w \in [1,\ldots,N_W]: \\ &\quad x_{pi} - W^{visit}_{w,x} \leq \Omega^{visit}(1 - b^{visit}_{piw}) \\ &\quad x_{pi} - W^{visit}_{w,x} \geq -\Omega^{visit}(1 - b^{visit}_{piw}) \\ &\quad y_{pi} - W^{visit}_{w,y} \leq \Omega^{visit}(1 - b^{visit}_{piw}) \\ &\quad y_{pi} - W^{visit}_{w,y} \geq -\Omega^{visit}(1 - b^{visit}_{piw}) \\ &\quad b^{visit}_{piw} = 0 \text{ or } 1 \end{aligned} \tag{4.7.18}$$

The waypoints which must be visited by some vehicles with suitable capability are specified in the matrix W of order $N_W \times 2$ where $(W^{visit}_{xw}, W^{visit}_{yw})$ is the position of the w^{th} way point. A vehicle capability matrix K can be defined of order $N_V \times N_W$ where $K_{pw} = 1$ if vehicle p can visit the w^{th} way point, otherwise $K_{pw} = 0$. Time dependencies, forcing one way point to be visited after another, separated by some interval can be included in a matrix Δ. Each row of the matrix represents a time dependency and it has a column for each waypoint. Thus if there are N_D time dependencies, the matrix Δ is of order $N_D \times N_W$. A dependency is encoded -1 in the column corresponding to the first waypoint and $+1$ in the column for the second. The corresponding element in the vector T_D is the interval between the two visits and

$$\Delta T_W \geq T_D \tag{4.7.19}$$

where T_W is vector of the times of visit to each waypoint. If it is required that each way point is visited exactly once by a vehicle of suitable capabilities, then the constraint to enforce this is

$$\forall w \in [1,\ldots,N_W]: \sum_{i=1}^{N}\sum_{p=1}^{N_V} K_{pi} b^{visit}_{piw} = 1 \tag{4.7.20}$$

Time dependencies are enforced by the following constraint

$$\forall d \in [1, \ldots, N_D] \sum_{w=1}^{N_W} \Delta_{dw} \sum_{i=1}^{N} \sum_{p=1}^{N_V} ib_{piw}^{visit} \geq T_{D_d} \tag{4.7.21}$$

The summation $\sum_{i=1}^{N} \sum_{p=1}^{N_V} ib_{piw}^{visit}$ extracts the time of visit for the w^{th} waypoint.

4.8 Cost Function Selection

The objective function can be taken as the sum of two costs: a quadratic cost function and a cost to minimize the violation of constraints. We can minimize a quadratic function whose variables must satisfy the state space equation of the dynamical system (4.2.1) as follows

$$\min_{\mathbf{s},\mathbf{u}} \hat{J} = \min_{\mathbf{s},\mathbf{u}} \int_0^{\infty} (\mathbf{s}^T Q \mathbf{s} + \mathbf{u}^T R \mathbf{u}) dt \tag{4.8.22}$$

subject to

$$\dot{\mathbf{s}} = A_c \mathbf{s} + B_c \mathbf{u}$$

where $\mathbf{s} \in \Re^n$ is the state vector and $\mathbf{u} \in \Re^{n_u}$ is the control. The system is assumed to start from some initial state $\mathbf{s}_0$. Alternatively we can replace weighting matrices Q and R of the quadratic formulation by nonnegative weighting vectors $\mathbf{q}$ and $\mathbf{r}$ to give the following convex cost function:

$$\hat{J} = \int_0^{\infty} (\mathbf{q}^T |\mathbf{s}| + \mathbf{r}^T |\mathbf{u}|) dt \tag{4.8.23}$$

where $|\mathbf{s}|$ and $|\mathbf{u}|$ are vectors with non-negative components. When combined with the constraints this 1-norm formulation yields a mixed integer/linear program which is much easier to solve than the mixed/quadratic program that would be obtained using the quadratic cost. To solve this fuel optimal control problem numerically, one must discretise the system and use a finite time horizon T. The finite horizon T transforms into $N = T/\Delta t$ discrete time steps, where Δt is the sample time. The original optimization problem is thus transformed into

$$\min_{\mathbf{s}_i,\mathbf{u}_i} \hat{J} = \min_{\mathbf{s}_i,\mathbf{u}_i} \sum_{i=0}^{N-1} (\mathbf{q}^T |\mathbf{s}_i| + \mathbf{r}^T |\mathbf{u}_i|) \Delta t + f(\mathbf{s}_N) \tag{4.8.24}$$

subject to

$$\mathbf{s}_{i+1} = A\mathbf{s}_i + B\mathbf{u}_i \tag{4.8.25}$$

where A and B are the discrete system matrices. Neglecting the constant term $\mathbf{q}^T|\mathbf{s}_0|$, writing $f(\mathbf{s})$ as $\mathbf{p}^T|\mathbf{s}_N|\Delta(t)$ and dividing by Δt gives:

$$\min_{\mathbf{s}_i,\mathbf{u}_i} \hat{J} = \min_{\mathbf{s}_i,\mathbf{u}_i}(\sum_{i=1}^{N-1} \mathbf{q}^T|\mathbf{s}_i| + \sum_{i=0}^{N-1} \mathbf{r}^T|\mathbf{u}_i| + \mathbf{p}^T|\mathbf{s}_N|) \tag{4.8.26}$$

This is a convex, piecewise linear cost function that can be transformed into a linear form by introducing slack variables $w_{ij}(i = 1, \ldots, N,\ j = 1, \ldots, n)$ and $v_{ik}(i = 0, \ldots, N-1,\ k = 1, \ldots, n_u)$ to give

$$\min_{\mathbf{w}_i,\mathbf{v}_i} \hat{J} = \min_{\mathbf{w}_i,\mathbf{v}_i}(\sum_{i=1}^{N-1} \mathbf{q}^T\mathbf{w}_i + \sum_{i=0}^{N-1} \mathbf{r}^T\mathbf{v}_i + \mathbf{p}^T\mathbf{w}_N) \tag{4.8.27}$$

subject to

$$\begin{aligned} s_{ij} &\leq w_{ij} \\ -s_{ij} &\leq w_{ij} \\ u_{ik} &\leq v_{ik} \\ -u_{ik} &\leq v_{ik} \\ \mathbf{s}_{i+1} &= A_d\mathbf{s}_i + B_d\mathbf{u}_i \end{aligned} \tag{4.8.28}$$

where j and k denote the components of the state and input vectors respectively. The cost function can be modified to minimize the difference between the state and final state $\mathbf{s}_{pf}$. For multiple vehicles, the cost function for the p^{th} vehicle is given by

$$\hat{J}_p = \sum_{i=1}^{N-1} \mathbf{q}_p^T|\mathbf{s}_{pi} - \mathbf{s}_{pf}| + \sum_{i=0}^{N-1} \mathbf{r}_p^T|\mathbf{u}_{pi}| + \mathbf{p}_p^T|\mathbf{s}_{pN} - \mathbf{s}_{pf}| \tag{4.8.29}$$

The overall linear optimization problem for N_V vehicles becomes

$$\min_{\mathbf{w}_{pi},\mathbf{v}_{pi}} \sum_{p=1}^{N_V} \hat{J}_p = \min_{\mathbf{w}_{pi},\mathbf{v}_{pi}} \sum_{p=1}^{N_V}(\sum_{i=1}^{N-1} \mathbf{q}_p^T\mathbf{w}_{pi} + \sum_{i=0}^{N-1} \mathbf{r}_p^T\mathbf{v}_{pi} + \mathbf{p}_p^T\mathbf{w}_{pN}) \tag{4.8.30}$$

subject to

$$\begin{aligned} s_{pij} - s_{pfj} &\leq w_{pij} \\ -s_{pij} + s_{pfj} &\leq w_{pij} \\ u_{pik} &\leq v_{pik} \\ -u_{pik} &\leq v_{pik} \\ \mathbf{s}_{p,i+1} &= A_p\mathbf{s}_{p,i} + B_p\mathbf{u}_{pi} \end{aligned} \tag{4.8.31}$$

where s_{pfj} are the given final values of the j^{th} state. In order to keep the violation of constraints to a minimum we define another cost function that includes small variables $m's$ of the soft constraints with proper adjustment of weights w^{rad}, w^{col} as follows:

$$\min_{m^{rad}_{pcif}, m^{col}_{pqig}} \sum_{p=1}^{N_V} \tilde{J}_p = \min_{m^{rad}_{pcif}, m^{col}_{pqig}} \sum_{p=1}^{N_V} (w^{rad} \sum_{c=1}^{N_O} \sum_{i=1}^{N} \sum_{f=1}^{4} m^{rad}_{pcif} + w^{col} \sum_{q=p+1}^{N_V} \sum_{i=1}^{N} \sum_{g=1}^{2} m^{col}_{pqig}) \tag{4.8.32}$$

So the overall cost function will be the combination of the objective function of (4.8.32) and the later developed (4.12.37), as follows:

$$\min_{\mathbf{w}_{pi}, \mathbf{v}_{pi}, m^{rad}_{pcif}, m^{col}_{pqig}} \sum_{p=1}^{N_V} J_p = \min_{\mathbf{w}_{pi}, \mathbf{v}_{pi}, m^{rad}_{pcif}, m^{col}_{pqig}} \sum_{p=1}^{N_V} \hat{J}_p + \tilde{J}_p \tag{4.8.33}$$

4.9 Example 1

To demonstrate the idea clearly, we first examine a simple case of one vehicle and one obstacle. Consider a radar with a short range of 7 km situated at $(14, 14)$. This circular range can be enclosed in a square region having vertices $(7, 7), (21, 7), (7, 21), (21, 21)$. By doing this, the radar avoidance problem becomes an obstacle avoidance problem and we can apply the above formulation. The vehicle moves with certain constraints. These constraints are on the velocity and acceleration. The minimum and maximum magnitudes of velocity with which the vehicle can move are 100 m/s and 300 m/s, respectively. Similarly, the minimum and maximum accelerations are -10 m/s/s and 10 m/s/s, respectively. But for this case only the upper bounds for these constraints will be applied. The vehicle starts at $(0, 14)$ with speed $100m/s$ along the positive-x direction and ends at $(30, 14)$. When the time length is fixed at $N = 120$, where N is the total number of time steps and each step is of one second, then the path obtained as a solution of the MILP formulation is shown in Figure 4.1. The vehicle is unable to reach its destination within this time interval. The value of the cost using the cost function given in equation (4.8.30) is 3013492.086. The time taken by the solver to determine this optimal path is 35.5160 seconds. At this point by increasing N iteratively, it was found that for $N = 130$ the vehicle achieved its target as shown in Figure

4.2. A question that immediately comes to mind is why did the UAV adopt a route from below the obstacle. The answer is of course that the UAV could follow a route from above of the obstacle but due to the inclusion of the states in the cost function it adopted a route from below to minimize the cost. In this way not only do the states and control remain small but a unique solution is also obtained. The cost value for this is 2691453.035 which is smaller than the previous value but the computation time for this trajectory increases and is found to be 72.4680 seconds which is much higher (nearly double) than the simulation for $N = 120$. The corresponding graphs for velocity and acceler-

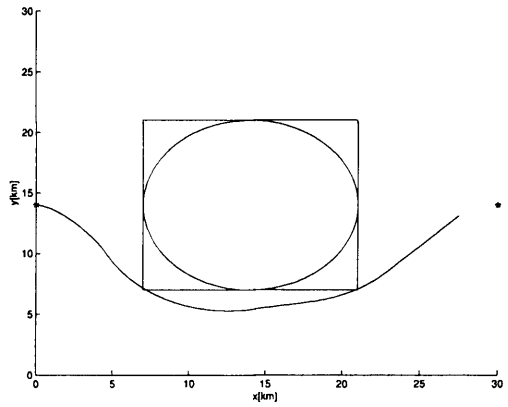


Figure 4.1: Single vehicle single obstacle path for $N = 120$ and without any constraints on the minimum velocity and acceleration

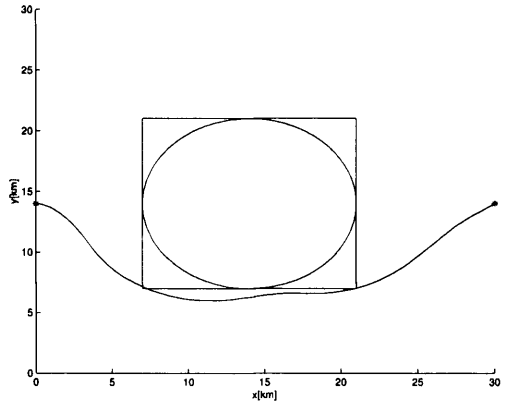


Figure 4.2: Single vehicle single obstacle path for $N = 130$ and without any constraints on the minimum velocity and acceleration

ation are shown in Figures 4.3 and 4.4 respectively. From Figure 4.3, it can

be seen that the speed of the vehicle for most of the time stays near to the maximum speed. For the fist 20 seconds, it almost increases linearly from 100 m/s to 300 m/s and then from 118 seconds to 130 seconds it decreases linearly. The speed at the target point was found to be $203.4341 m/s$. Figure 4.4 shows sudden variations in acceleration but all these variations in speed and acceleration are within specified limits in spite of the fact that minimum constraints for these values have not been applied.

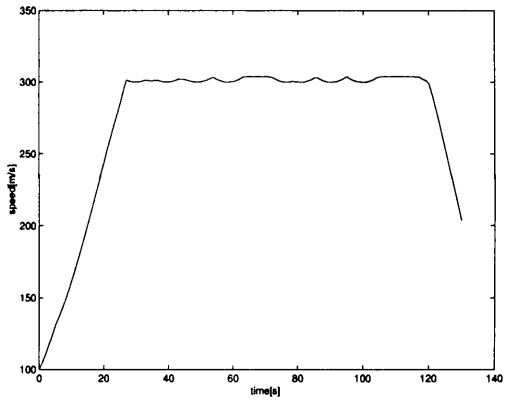


Figure 4.3: Single vehicle single obstacle speed map for $N = 130$ and without any constraints on the minimum speed and acceleration

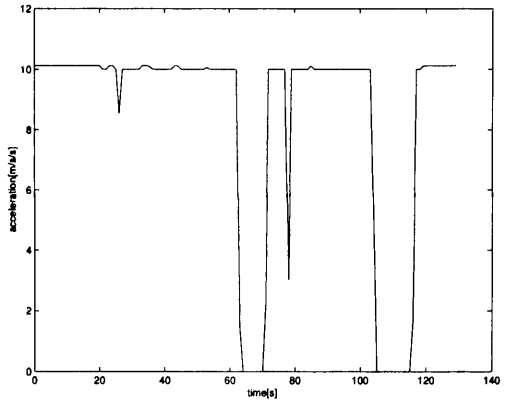


Figure 4.4: Single vehicle single obstacle acceleration map for $N = 130$ and without any constraints on the minimum speed and acceleration

In the next scenario two more short range SAM units are added which are located at $(30, 6)$ and $(30, 22)$. This will be helpful in seeing how much complexity is added when the number of radars is increased. Again for this case,

the lower bounds on velocity and acceleration are ignored. These constraints involve integer variables which will make the problem hard. The path obtained by minimising the cost given by (4.8.30) and also that for minimum time is shown in Figure 4.5. Again for the minimum time objective, the simulation is performed iteratively each time changing the time window given by N from some initial value until the destination is reached. The value of N is found to be 155 seconds while the computation time and cost are 652.2340 seconds, 2953218.281, respectively. Although the cost is almost the same as for one radar but the computation time increases enormously (almost 9 times). The speed map in Figure 4.6 shows that for this case the vehicle speed starts decreasing at nearly 73 seconds where the position is $(15.8586, 4.7793)$ which is a minimum point of the trajectory below the first obstacle. This indicates that the UAV prepares itself to pass though the narrow passage between the two following obstacles from the lower end point of the first obstacle. So in order to pass though this passage without violating the constraints, the UAV has to reduce its speed. Again at nearly 93 seconds, the speed starts increasing at a trajectory point of $(20.6737, 6.8219)$ which is the point that coincides with the right lower corner of the first obstacle. After that for the rest of the part of the trajectory, a similar variation in speed can be seen. On the other hand, the acceleration in Figure 4.7 shows sudden variations around 60 seconds but for the rest of the time it stays near to its maximum value. These variations occur when vehicle encounters the lower left corner of the first obstacle.

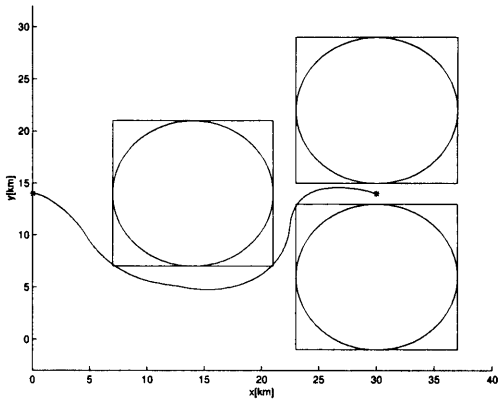


Figure 4.5: Single vehicle three obstacles path for $N = 155$ and without any constraints on the minimum velocity and acceleration

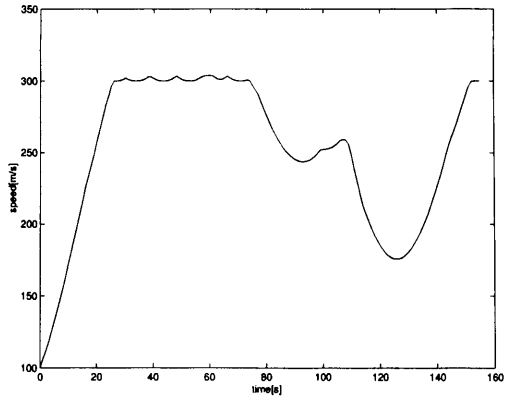


Figure 4.6: Single vehicle three obstacles speed for $N = 155$ and without any constraints on the minimum velocity and acceleration

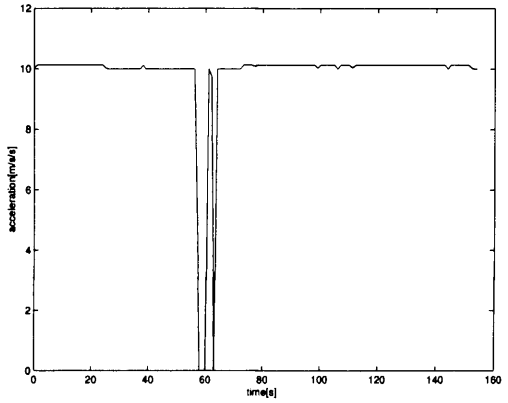


Figure 4.7: Single vehicle three obstacles acceleration for $N = 155$ and without any constraints on the minimum velocity and acceleration

4.10 MILP for Real Time Path Planning

The main difficulty in using MILP is the computation demand it requires and also we do not know the optimal planning horizon in advance. If there are more constraints, the process may be very slow even taking days to complete the optimization process. Hence for real time computation, a receding horizon approach can be used, in which the path is computed online by solving a MILP over a limited horizon at each time step. In this case, the path of the vehicle is composed of a sequence of locally optimal segments. At a certain time step, the MILP is solved for T future time steps, where the length T of the planning

horizon is chosen as a function of the available computation resources. Solving this local MILP provides the input commands for the T future time steps. However only a subset of these T input commands is actually implemented. The process is then repeated and a new set of commands is developed for the next time window. Usually the applied subset is restricted to the first control input, such that a new set of input commands is calculated at each time step.

4.10.1 Model Predictive Control or Receding Horizon Control

Originally developed to meet the specialized control needs of power plants and petroleum refineries, Model Predictive Control (MPC) technology can now be found in a wide variety of application areas including chemicals, food processing, automotive, aerospace, metallurgy, and pulp and paper [83]. MPC is a control strategy that explicitly uses a model of the system in order to predict system behaviour. This is then used to find the best control signal possible by minimizing an objective function. There are several advantages of using MPC to control a UAV; some of them are listed below:

- The concept is equally applicable to single-input single-output (SISO) as well as multi-input multi-output (MIMO) systems.
- MPC can be applied to linear and nonlinear systems.
- It can handle the constraints in a systematic way during the controller design.
- The controller is designed at every sampling instant so disturbances can easily be dealt with.
- Explicit use of a model to predict the system output at future time instants (also called horizon).
- Obtaining a control signal by minimizing an objective function.

The MPC strategy can be compared to driving a car. The driver knows the desired reference trajectory for a finite horizon, and by taking the car

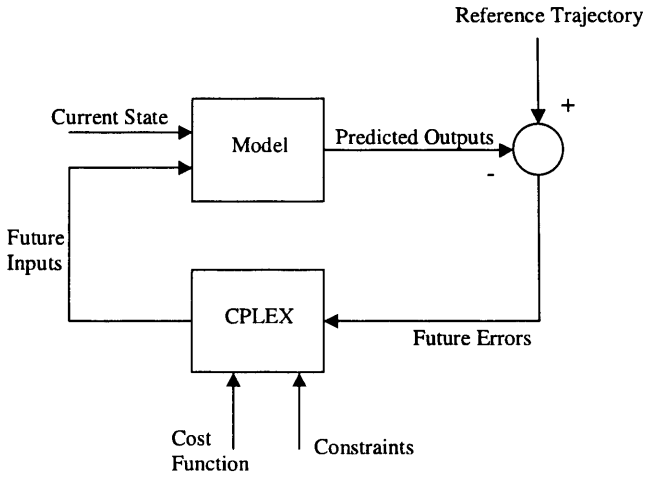


Figure 4.8: Model Predictive Control Scheme

characteristic into account, he or she decides which control actions to take in order to follow the desired trajectory. Only the first control actions (using the accelerator, brakes and steering) are taken at each instant and the procedure is then repeated. This can be described mathematically in more detail by the following steps:

1. The future outputs for a determined horizon N, called the prediction horizon, are predicted at each instant t using the system model. These predicted outputs $y(k+j|k)$ for $j = 1 \ldots N$ depend on $x(k|k)$, the current state and on future control signals $u(k+j|k), j = 0 \ldots N-1$.

2. The set of future control signals is calculated by optimizing a criterion in order to keep the process as close as possible to a reference trajectory $r(k+j|k)$. The criterion usually takes the form of a quadratic function of errors between the predicted output and the predicted reference trajectory. The control effort is also included in the objective function in most cases. An explicit solution can be obtained if the criterion is quadratic, the model is linear and there are no constraints, otherwise some optimization method like $CPLEX$ must be used.

3. The control signal $u(k|k)$ is sent to the system while the control signals $u(k+j|k), j = 1 \ldots N-1$ are rejected and step 1 is repeated with all states brought up to date. As the horizon moves forward and new information

becomes available, the $u(k+1|k+1)$ calculated at the next time step will be different from $u(k+1|k)$.

The basic structure of MPC is shown in Figure 4.8. If there is no model mismatch i.e. the model is identical to the process and there are no disturbances and constraints, the process will track the reference trajectory on the sampling instants.

4.10.2 Possible Infeasibility with Receding Horizon

When using the receding horizon approach, the problems of non-existence of feasible solutions may occur during the optimization procedure even though, in theory there are solutions to the whole problem. This is because the look ahead horizon is limited. The vehicle can be led to a critical state for which MILP has no solution at the next iteration. In other words, a feasible solution for future T time steps at time step i does not guarantee a feasible MILP at time step $i+1$. This can be further explained for the situation in which at the last time step of the planning horizon, the vehicle is moving at maximum speed, while its position is just outside an obstacle that has not been spotted yet. Since the position of the vehicle satisfies the anti-collision constraints, this situation corresponds to a feasible solution of the MILP. At the next time step, the obstacle is identified and the vehicle needs to brake or turn which however exceeds the constraints on acceleration or on the available manoeuvre space. Hence, a collision with the obstacle will result and MILP is unable to find a feasible solution in the next time step. Increasing the horizon will ease this kind of situation, but will raise computational demand.

4.10.3 Safe Feasible Mechanism

In the radar/SAM exposure minimization problem, there are no physical obstacles. Rather we have radars of various detection ranges. We may have, say, three types: a long range SAM unit with a nominal range of 65 km, a medium range SAM unit (25 km range) and a short range SAM unit (7 km range). We can approximately model these circular regions with squares of

length equal to the diameters of these circles. These radar ranges can overlap with one another. So if the path is totally blocked by these overlapping radars or due to the use of the receding horizon approach with hard constraints in (4.4.10), (4.3.5) and the cost (4.8.30), MILP may lead to infeasible solutions. But by using the soft constraints (4.3.7), (4.4.11) and the cost (4.8.33), we can always get a feasible solution. These violations are kept to a minimum by the use of the small variables (the $m's$) in the cost. If further reduction of these violations is required, we may model the threats as squares of flexible size, slightly greater than the actual fitted square. This increment can be taken as ten percent of the actual square. In this way, a vehicle can enter the radar zone but have to follow the safest possible path by optimizing this flexibility to a minimum.

4.11 Example 2

The scenario considered here has 10 radars shown as circles in Figure 4.9. Five radars are of medium range (25 km) centred at $(100, 100)$, $(125, 65)$, $(125, 135)$, $(50, 155)$, $(50, 45)$ while five radars are of short range (7 km) centred at $(42, 102)$, $(167, 182)$, $(167, 127)$, $(167, 37)$, $(167, 77)$. The initial positions of UAV1, UAV2, UAV3 are $(10, 10)$, $(10, 120)$, $(10, 180)$ respectively. The three UAVs start at the same time and move towards the common goal at $(170, 100)$ with the same speed of 200 m/s making an angle of $40°$ with the horizontal. In order to apply the previous formulation, the radars are modelled as rectangular flexible constraints enclosing the circular range. In the finite receding horizon scheme to prevent infeasibility, the vehicles can enter the threat zone keeping violations at a minimum. Therefore the modified formulation forces the UAV to leave the threat zone as soon as possible. The maximum violation will occur when the vehicle enters the threat zone at maximum speed perpendicular to the rectangular boundary at a point where this boundary line touches the circular boundary region. In such a situation, there is a chance that a vehicle may travel deep inside the threat zone before turning away. To reduce this deep penetration, an additional safety measure was taken by extending the

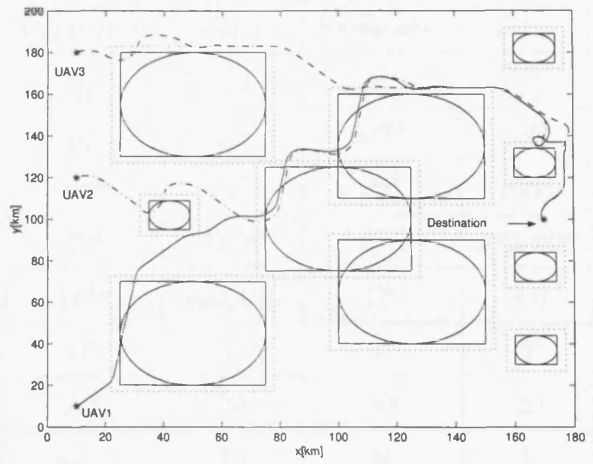
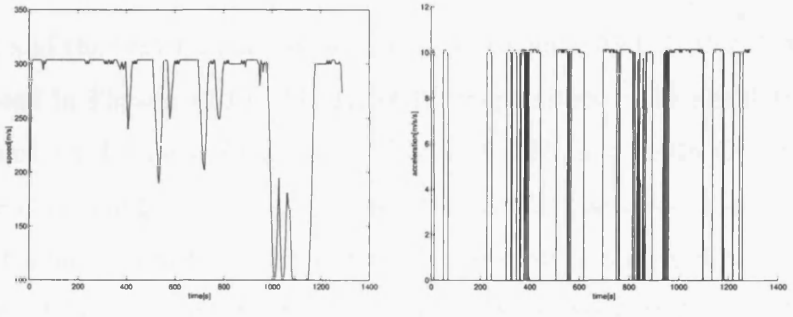


Figure 4.9: Optimal trajectories calculated using modified finite receding horizon control for three UAVs moving towards the same target with upper and lower bounds on speed and acceleration

rectangular boundary say ten percent. This is shown clearly in Figure 4.9 by dotted lines enclosing the region. The finite horizon T was chosen as 10 seconds which means at each iteration MILP finds the optimized solution for 10 seconds and hence a control sequence for the next 10 seconds is obtained but only the first current control input is implemented. The parameters used in the simulation are shown in Table 4.1. The parameters v_{min}, v_{max}, u_{max}, d_x^{col}, d_y^{col} were kept fixed. The other parameters are tuned to get the desired response.



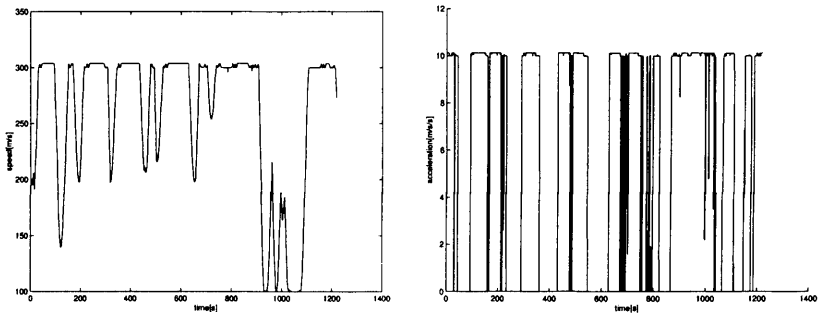
(a) Speed for UAV1 (b) Acceleration for UAV1

Figure 4.10: Speed and acceleration for UAV1

Trajectories obtained by solving this centralized problem are shown in Fig-

Parameter	Value	Parameter	Value
$\mathbf{q}_p^T$	$[1,1,1,1]$	$\mathbf{r}_p^T$	$[1,1]$
$\mathbf{p}_p^T$	$[1,1,1,1]$	w^{obs}	10^{10}
w^{col}	10^{10}	d_x^{col}	1000
d_y^{col}	1000	Ω^{col}	$800,000$
Ω^{obs}	$800,000$	Ω^{v}	900
Ω^{u}	50	N_C^v	20
N_C^v	20	N_C^u	20
u_{max}	10	δt	1
v_{min}	100	v_{max}	300

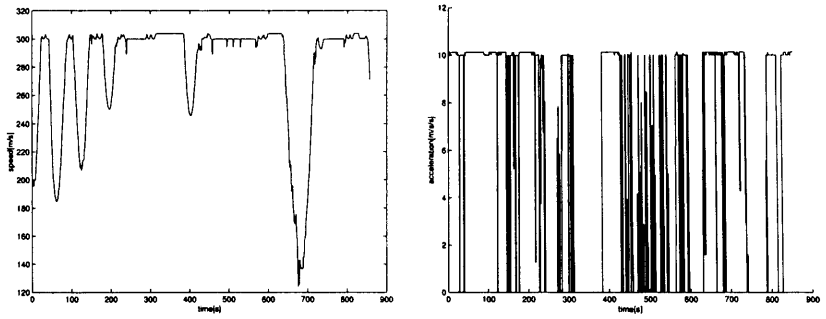
Table 4.1: Parameters used in the simulation



(a) Speed for UAV2 (b) Acceleration for UAV2

Figure 4.11: Speed and acceleration for UAV2

ure 4.9 and the velocity and acceleration profiles for UAV1, UAV2, UAV3 are illustrated in Figures (4.10), (4.11), (4.12) respectively. The simulation was performed on a machine with a CPU of 2.66 GHz and 1.048 Gb of RAM. The computation time for the full simulation is 27844 seconds (the time when the last vehicle, which is UAV1, reaches the destination) and total cost is $1.1899e+013$. The simulation times to destination for UAV2 and UAV3 are 27595 and 6386.3 seconds, respectively. The actual times taken by UAV1, UAV2, UAV3 to move on these trajectories are 1286, 1216, 856 seconds, respectively. By comparing these simulation and actual times, it is concluded that the simulation times are very high for use in online trajectory calculations.



(a) Speed for UAV3 (b) Acceleration for UAV3

Figure 4.12: Speed and acceleration for UAV3

It will be interesting to analyse the time taken to solve this finite receding horizon MILP problem at each time step. This is shown in Figure (4.14) where the peak variations in the MILP solution time are limited to the interval from 1030 to 1102 seconds where the maximum time taken is 5473 seconds, which occurs at the iteration time of 1036 seconds and at this time the trajectory points for UAV1 and UAV2 are shown in Figure (4.13). These points occur near the imaginary boundary of the short range radar which is the last obstacle before reaching the destination. It can be seen in Figure (4.13) that at this time UAV1 takes a full turn to correct its direction to reach the goal immediately after emerging out from the imaginary boundary of the short range obstacle. So in order to take the maximum turn, UAV1 has to reduce its speed to a minimum. On the other hand, at this time instant UAV2 is traveling exactly on the imaginary boundary with minimum speed. So the minimum speed constraint is active at the 1036 time step for both UAV1 and UAV2 and as this constraint is non convex involving integer variables, which make the problem hard. For other points of the peak simulation time interval $(1030 - 1102)$ seconds, the minimum velocity constraint is active for either UAV1 or UAV2 but not for both. Due to these huge variations in solution time for that particular interval, the variations for other parts of the Figure 4.14 are not visible. For this purpose Figures 4.15(a) and 4.15(b) are provided.

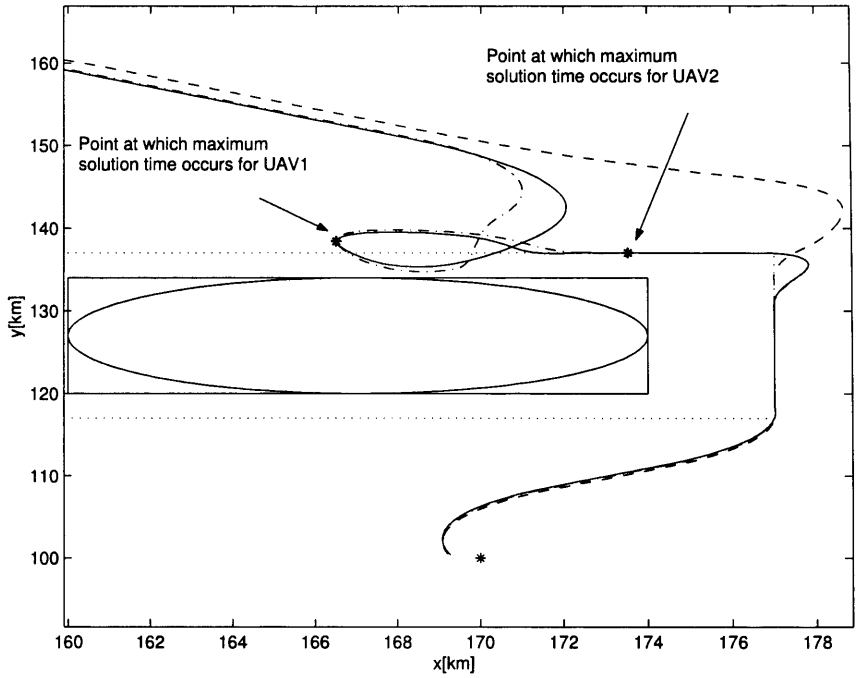


Figure 4.13: Points on the trajectory with maximum solution time for both UAV1 and UAV2

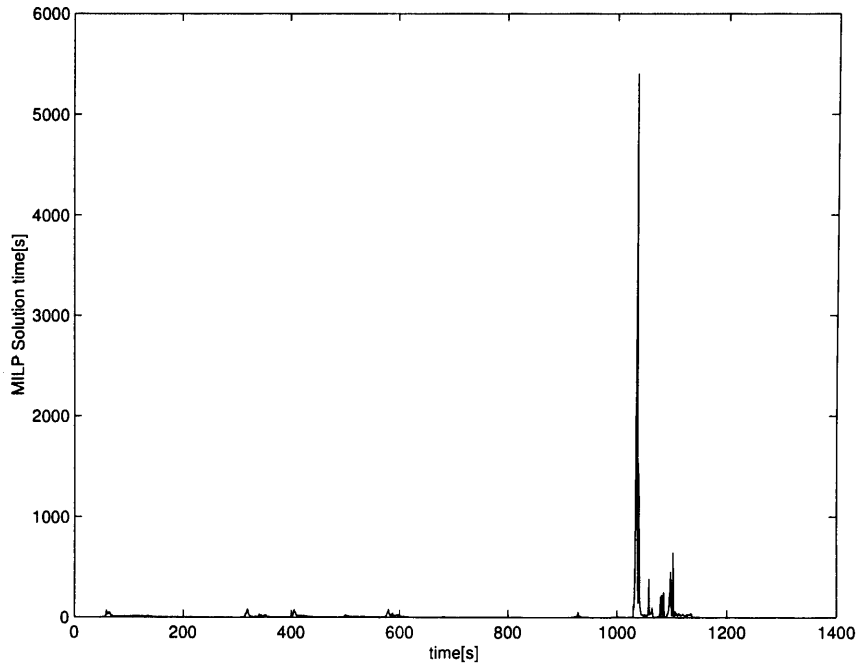
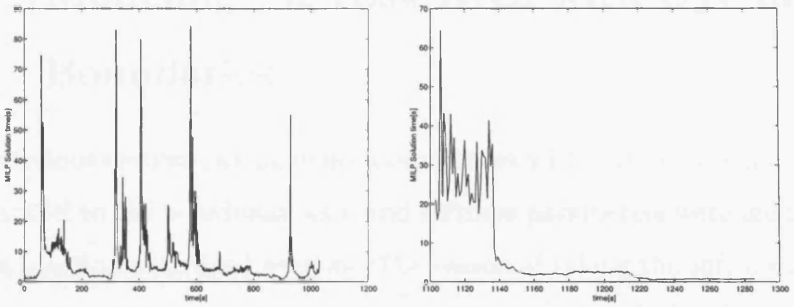


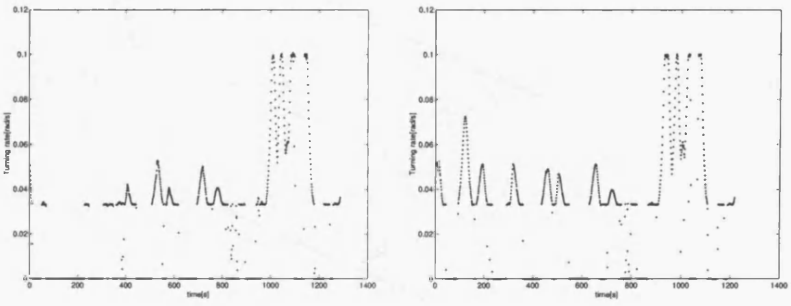
Figure 4.14: Time taken by MILP to solve the problem at each time step for full simulation



(a) For time values from 0 to 1029 seconds

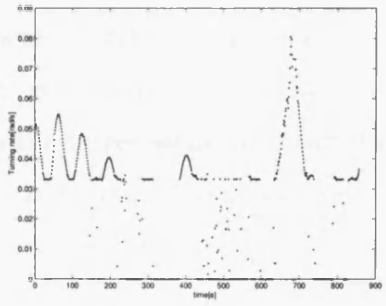
(b) For time values from 1103 to 1286 seconds

Figure 4.15: Time taken by MILP to solve the problem at each time step for two different time ranges



(a) UAV1

(b) UAV2



(c) UAV2

Figure 4.16: Turning rates for UAV1, UAV2 and UAV3

The turning or heading rate for UAV1, UAV2 and UAV3 are shown in Figure 4.16.

4.12 Modeling the Risk Area with Dynamical Boundaries

In the previous sections, we modelled the risk area with soft rectangular boundaries parallel to the coordinate axes and softness parameters were included in the cost function with fixed weights. The reason of taking the soft boundaries is described in those sections. From simulation, it has been observed that sometimes near the imaginary boundary of the risk area a UAV will loiter for a long time. This is because of the turning of a UAV off a wall again and again without finding a path to direct it towards the target either due to the limited horizon or due to the fixed weights on the softness parameters in the cost function described in section 4.8. This problem can be fixed by modelling the

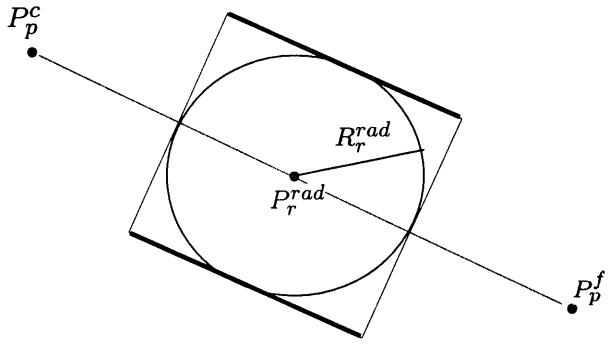


Figure 4.17: Dynamical boundaries for risk area

risk area with soft dynamic rectangular boundaries. At each simulation time step, two of the boundaries for each radar are taken parallel to the line of sight of the current point to the target while the other two boundaries are taken perpendicular to the line of sight as shown in Figure 4.17. Also the weights on the softness parameters in the auxiliary cost function can be chosen dynamically according to the distance of the current point to the target so that when a UAV is near to the boundary of the detected SAM unit, it will be directed towards the target by sliding along the walls of the boundary due to these different weights. The range of the radar warning receivers (RWR) mounted onboard on the UAV is assumed slightly higher than the range of the of the radar as defined in the design challenge of the GARTEUR Flight Mechanics

Action Group 14 [36]. This assumption is very crude and does not reflect the real situation. So in this section, it is assumed that a SAM site is considered to be known in position and range when its range comes within a distance of $v_{max} * T$ to the UAV, where T is the time horizon and v_{max} is the maximum velocity of the UAV. Consider a radar situated at a point $P_r(x_r, y_r)$ having nominal range R_r enclosed in a rectangular boundary as shown in Figure 4.17. The position of the p^{th} UAV at the current simulation time step is (x_p^c, y_p^c) and (x_p^f, y_p^f) is the target point for this vehicle. Therefore the equations of the boundary lines, shown bold in Figure 4.17 of a radar situated at (x_r^{rad}, y_r^{rad}), which are parallel to the line of sight are

$$(y_p^f - y_p^c)(x_p - x_r^{rad}) - (x_p^f - x_p^c)(y_p - y_r^{rad}) = \pm R_r^{rad} d_p^c \qquad (4.12.34)$$

Similarly, the equation of the boundary lines perpendicular to the line of sight are

$$(x_p^f - x_p^c)(x_p - x_r^{rad}) + (y_p^f - y_p^c)(y_p - y_r^{rad}) = \pm R_r^{rad} d_p^c \qquad (4.12.35)$$

where $d_p^c = \sqrt{(x_p^c - x_p^f)^2 + (y_p^c - y_p^f)^2}$ is the distance of the current point of the UAV from the target. In the situation of overlapped obstacles, the receding horizon algorithm may not work out a feasible solution because of such formulated hard constraints. For this consideration, the hard constraints can be transformed to soft constraints by the introduction of small variables. At each time step i the position (x_i, y_i) of the vehicle must lie in an area outside of the risk area. If their are N_V vehicles and N_R radar sites, then the general mixed integer linear form of the radar avoidance constraints [54] can be written as: $\forall p \in [1, \ldots, N_V], \forall r \in [1, \ldots, N_R], \forall i \in [1, \ldots, N]$:

$$\begin{aligned}
&(y_p^f - y_p^c)(x_{pi} - x_r^{rad}) - (x_p^f - x_p^c)(y_{pi} - y_r^{rad}) \\
&\leq -d_p^c R_r^{rad} m_{pci1}^{rad} + \Omega^{rad} b_{pgi1}^{rad} \\
&(y_p^f - y_p^c)(x_{pi} - x_r^{rad}) - (x_p^f - x_p^c)(y_{pi} - y_r^{rad}) \\
&\geq d_p^c R_r^{rad} m_{pci1}^{rad} - \Omega^{rad} b_{pgi2}^{rad} \\
&(x_p^f - x_p^c)(x_{pi} - x_r^{rad}) + (y_p^f - y_p^c)(y_{pi} - y_r^{rad}) \\
&\leq -d_p^c R_r^{rad} m_{pci2}^{rad} + \Omega^{rad} b_{pgi3}^{rad}
\end{aligned}$$

$$
\begin{aligned}
&(x_p^f - x_p^c)(x_{pi} - x_r^{rad}) + (y_p^f - y_p^c)(y_{pi} - y_r^{rad}) \\
&\geq d_p^c R_r^{rad} m_{pci2}^{rad} - \Omega^{rad} b_{pgi4}^{rad} \\
&\sum_{k=1}^{4} b_{pcik}^{rad} \leq 3 \\
&0 \leq m_{pci1}^{rad},\ m_{pci2}^{rad} \leq 1
\end{aligned} \tag{4.12.36}
$$

where m_{pci1}^{rad}, m_{pci2}^{rad} are small decision variables between 0 and 1. The idea is to increase these variables to 1 by incorporating them into the auxiliary objective (cost) function. The problem formulation returns to the original setting when these $m's$ are 1. If it is not possible to reduce them to 1, i.e. the original hard constraints cannot be satisfied, the algorithm will have the flexibility to generate solutions which violate these constraints as little as possible.

As discussed previously, one way to increase solvability of the problem while keeping the violation of constraints to a minimum is to soften the constraints by including small variables ($m's$) in the cost function. From this and the above discussion, the second component of the cost function will be modified and is

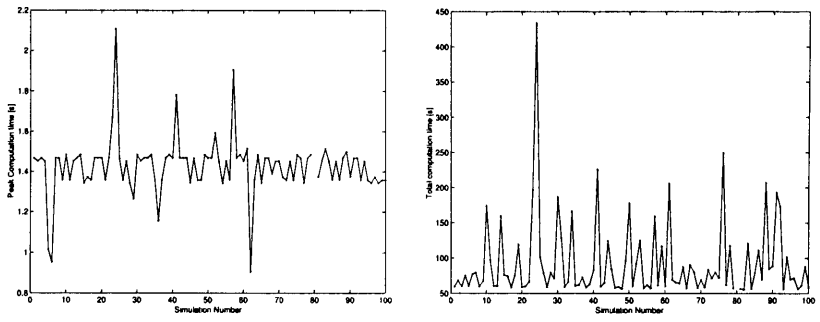
$$
\begin{aligned}
\min_{m_{pcif}^{rad}, m_{pqig}^{col}} \sum_{p=1}^{N_V} \tilde{J}_p = \min_{m_{pcif}^{rad}, m_{pqig}^{col}} \sum_{p=1}^{N_V} (w^{col} \sum_{q=p+1}^{N_V} \sum_{i=1}^{N} \sum_{g=1}^{2} m_{pqig}^{col} \\
- \sum_{h=1}^{2} w_h^{rad} \sum_{c=1}^{N_R} \sum_{i=1}^{N} \sum_{f=1}^{4} m_{pcif}^{rad})
\end{aligned} \tag{4.12.37}
$$

So the overall objective function is $J_p + \tilde{J}_p$ for the p^{th} vehicle. In equation (4.12.37), the weights ($w_1^{rad} = \alpha \times d_p^c$, $w_2^{rad} = \beta \times d_p^c$) are selected dynamically according to the distance of the current position of the UAV from the target and the constant α is given a smaller value than β. To avoid collision, the weight w^{col} is given a very large value. The simulation for one vehicle case was performed for 100 randomly created scenarios consisting of short range (7 km) and medium range (25 km) SAM units distributed over a 200 km $\times$ 200 km area. The random parameters are number, location and strength of the SAM units. The number of the SAM units were limited between 5 and 10. Since the focus was to make the technique useful for real time computation so an upper limit on computation time or optimisation process for CPLEX was set to 1.5 seconds. The justification for this is that large and difficult problems may

Parameter	Value	Parameter	Value
$\mathbf{q}_p^T$	$[1,1,1,1]$	$\mathbf{r}_p^T$	$[1,1]$
$\mathbf{p}_p^T$	$[1,1,1,1]$	w_1^{rad}	$3 \times d_p^c$
w^{col}	10^{10}	d_x^{col}	1000
d_y^{col}	1000	Ω^{col}	8×10^5
Ω^{rad}	8×10^{11}	Ω^v	9000
Ω^u	50	N_C^v	30
N_C^v	30	N_C^u	30
u_{max}	10	w_2^{rad}	$6 \times d_p^c$
v_{min}	100	v_{max}	300

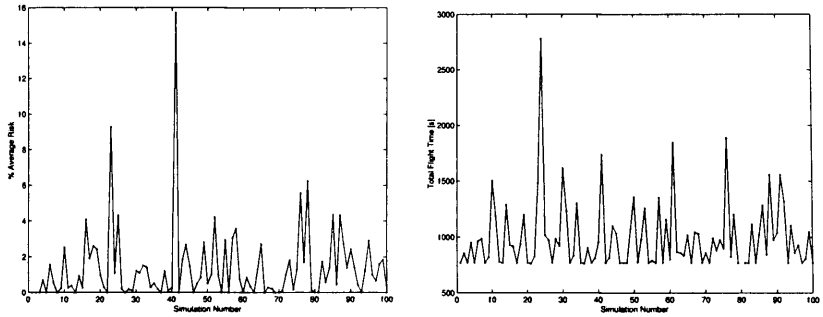
Table 4.2: Parameters used in the simulation using dynamic boundary formulation

take hours or days to prove for optimality and the optimal solution might have been obtained at very early stages of calculation. The parameters used in all simulations are summarised in Table 4.2. The measured quantities are success rate, peak computation time (maximum time to compute a way point), peak risk, average risk and total flight length. The results are shown in Figures 4.18 and 4.19. The horizon T is chosen as 5 seconds, i.e, the trajectory is evaluated for five time steps of 1 seconds duration each.



(a) Peak computation time (b) Total computation time

Figure 4.18: Peak and total computation time for 100 randomly generated data sets



(a) Average risk (b) Total flight time

Figure 4.19: Average risk and total flight time for 100 randomly generated data sets

The success rate was 99% i.e, the solution was obtained for 99 random scenarios. For 80^{th} scenario, the solution was not obtained within set time limits. Figure 4.18(a) shows a little violation of the set time limit for a few scenarios. The cause for these violations are not clear. This extra delay might be due to the internal setting of the commercially available solver CPLEX while communicating with AMPL/CPLEX from within Matlab environment. The risk was evaluated with the nonlinear probabilistic risk model described in the next chapter. As shown in Figure 4.19(a), the average risk for each trajectory while travelling to the selected path for most of the scenarios is reasonable except for the 41^{st} and 23^{rd} scenarios. The average risk for 41^{st} scenario is 16%. The highest values of the total distance travelled, total computation time, total flight time and peak risk occurs at 24^{th} scenario. This is due to the very high risk experienced by the UAV as it moves towards the target and due to that it starts loitering in order to find a safe path. This formulation is applied to the same multi-vehicle scenario as discussed in Example 2 and the simulation results are presented in Figure 4.20. From this Figure, we can see that the violations of the dangerous zones are very small when compared to Figure 4.9 and also it removes long time loitering of a UAV with the boundary. In this exercise the horizon T is chosen as 25 seconds, that is, the trajectory is evaluated for five time steps of 5 seconds duration each but only the first control input is implemented. The collision avoidance can be seen in Figure

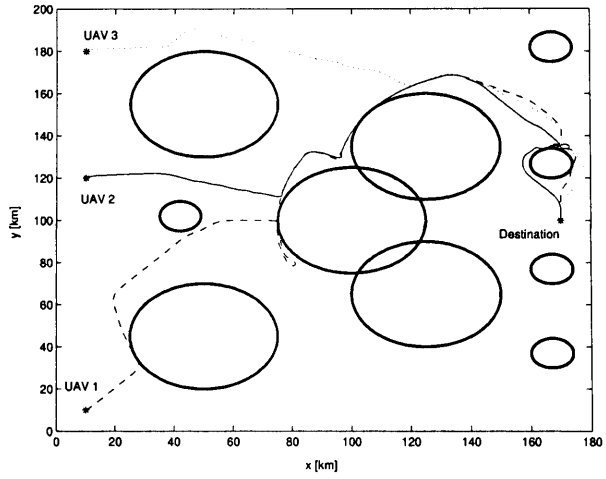


Figure 4.20: A multi-vehicle scenario and trajectories for three UAVs using dynamic boundary formulation

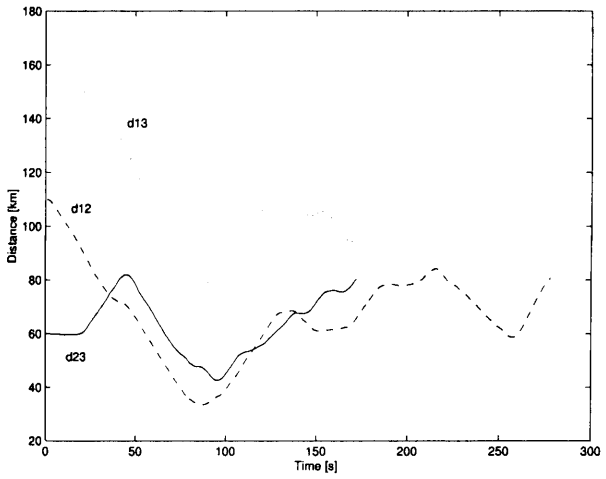


Figure 4.21: Distances between the UAVs during flight

4.21 which shows the distances between each two vehicles at different times.

4.13 Conclusion

A mixed integer linear problem was formulated for different constraints and applied to different case study examples. Using MILP, however, to design a whole trajectory with a planning horizon fixed at the goal is very difficult to perform in real time because the computational effort required grows rapidly with problem size. It was shown that this limitation can be avoided by using

a *receding planning horizon* in which MILP is used to form a shorter plan that extends towards the goal but does not necessarily reach it. The performance of a RHC strongly depends on the proper evaluation of the weights w_1^{rad} and w_2^{rad}. Care must be taken in selecting these weights when the feasibility of the path beyond the plan must be ensured. Robustness of receding horizon control against infeasibility is guaranteed by modelling the constraints as soft. The efficiency of the technique depends upon proper modelling of the mixed linear constraint, on the time horizon and also on proper adjustment of the weights on the pairs of the dynamic boundaries described in Section 4.12. The dynamic boundaries are used because of the turning of a UAV off a wall again and again without finding a path to direct it towards the target either due to the limited horizon or due to the fixed weights on the softness parameters in the cost function described in Section 4.8. The optimality can be increased by increasing the time window but by doing this the computational load will increase. A practical compromise between optimality and computational load is essential.

Due to the inherent complexity and probabilistic nature of the problem, a probabilistic technique will be presented in the next chapter that will model a nonlinear performance index to incorporate coupling between different parameters.

Chapter 5

A Probabilistic Framework for Path Planning of UAVs

5.1 Introduction

A three dimensional probabilistic approach for the path planning of UAVs is presented. This approach to local path planning is proposed for the three main reasons: first, the inevitable uncertainty of the measurements from the sensors; second, the intrinsic uncertainty of an unknown environment; and finally, the structure of reasoning of any intelligent system is naturally probabilistic. The novelty of the algorithm lies in its real time applicability due to a very low computational load in spite of the fact that it finds a path in three dimensions. The paths are locally optimal and are feasible for the UAV to follow by keeping the turn angle within certain maximum limits. Furthermore, vehicle collisions are avoided by maintaining a minimum distance between vehicles. Also UAVs are prevented flying at very low altitudes because of the danger of crashing into ground objects. Finally, since each UAV has limited fuel, a compromise can be made between risk and fuel consumption by limiting height and search angle.

5.2 Problem Formulation

5.2.1 Environment

The environment consists of a number of surface to air missiles (SAMs). A SAM fire unit is assumed to consist of one radar used for both surveillance and tracking and a number of missile launchers. The surveillance/tracking radar has a nominal detection range against UAVs. The fire units are integrated to form an integrated air defence system. The radar can be switched off, if an incoming anti-radiation missile is detected. The air defence units are randomly deployed in the operational area. Some of the surveillance radars are on alert using continuous transmission, while the other air defence units remain silent and serve as pop-up threats.

The UAVs fly in three dimensions with limitations on speed, acceleration and duration of flight. They are equipped with a radar warning receiver (RWR), which gives a bearing on an emitting radar with certain accuracy. The RWR has a range greater than the radar's detection range. To make the problem simpler, it is first assumed that minimum ideal communication exists between UAVs and each UAV calculates its own path using the information received from other UAVs and thus we have a decentralized control problem.

The problem is to find the safest path for a UAV from one point to another. Suppose the UAVs move with constant speed and $p^I(x, y, z)$ represents the risk at position (x, y, z) faced by the I^{th} UAV due to the SAMs. The safest path in going from point P_0 to P_T is a sequence of points in 3D obtained by minimizing the cost function

$$J = \int_{P_0}^{P_T} p^I(x, y, z)dt \tag{5.2.1}$$

over all points (x, y, z).

5.2.2 Risk Modelling

The overall UAV risk function is complicated due to the influence of different factors and can be modelled adequately in a probabilistic frame work as described below.

Probability of Hit For each defence unit (radar and SAM) aimed at a UAV there is a hit probability. Within the given range of SAM, this probability depends on the position of the UAV when the missile reaches it. This can be taken as a function of height (h) and distance (d) and is given by

$$\begin{aligned} p_k(h,d) =&(1-\text{softStep}(d, R_{s,m,l}, s_{k_1})).\text{softStep}(d, 0.1.R_{s,m,l}, s_{k_2}) \\ &.\text{softStep}(\arcsin(h/d), \gamma, s_{k_3}) \end{aligned} \tag{5.2.2}$$

where

$$\text{softStep}(x, x_0, s) = \frac{1}{2}\left(1 + \frac{x - x_0}{\sqrt{s^2 + (x-x_0)^2}}\right) \tag{5.2.3}$$

γ is the lower coverage angle of the radar and s_{k_i} is the softness of the step function as shown in the Figure 5.1. $R_{s,m,l}$ is the range of the missile which may be short, medium or long range. The hit probability for a long range SAM ($R = 65$ km) is shown in Figure 5.2 for $s_{k_1} = 12$, $s_{k_2} = 2.4$, $s_{k_3} = 0.1$, with numbers on different contours indicating the probability of hit.

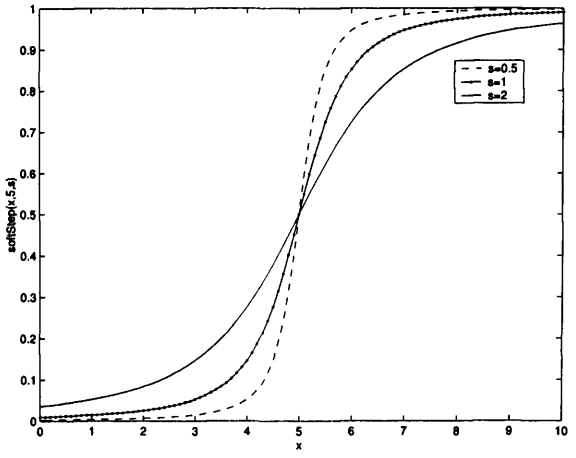


Figure 5.1: The function softStep(x,5,s) with $s = 0.5, 1, 2$

Probability of Destruction If a UAV is within the reach of M SAM sites, the hit probability is increased by possible cooperation such as alternating radar transmission or choice of launch site. This effect is modelled by evaluating the hit probability of all covering SAM sites $p_k^j(h,d)$ and using the relation

$$p_{des}(h,d) = 1 - \prod_{j=1}^{M}(1 - p_k^j(h,d)) \tag{5.2.4}$$

htb]

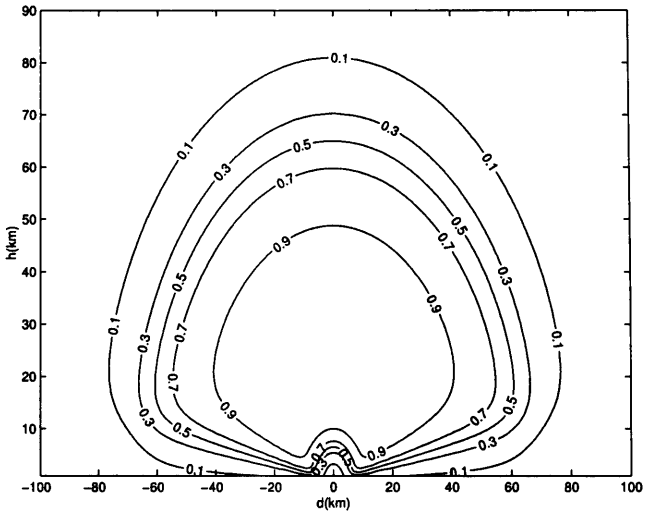


Figure 5.2: Hit probability for long range SAM

Probability of Crash When a UAV flies at very low altitude, there is a possibility of crashing with ground objects, like trees or hills. So in order to prevent all UAVs from flying around at zero altitude, this can be modelled as a crash probability by

$$p_{cr}(h) = 1 - \text{softStep}(h, h_{cr}, s_{cr}) \tag{5.2.5}$$

where h_{cr} is the nominal critical height and each UAV is forced to fly above this height for safety reasons. s_{cr} is the softness parameter of the above probability function which can be tuned according to the situation. This softness parameter is used to relax or strictly follow the critical height. If there is a hilly area, its value may be low, and in valleys it may have higher values to increase the softness because in this situation there may be objects like trees and buildings which may not be very high.

Probability of Survival

Due to small sizes, UAVs have limited fuel capacity. Typically, flight times of UAVs are 30 (*min*). The risk decreases with increasing height after a critical height but on the hand fuel decreases. So a compromise can be made between risk and fuel consumption by limiting the height.This effect can be modeled as survival probability p_{sur} similar to the crash probability.

Probability of Collision

When the mission involves a group of UAVs, risk of collision with other vehicles

is a function of the distance of the vehicle from other UAVs and can be modeled as

$$p_{co}(D) = 1 - \text{softStep}(D, d_{co}, s_{co}) \tag{5.2.6}$$

Where d_{co} is the safety distance to avoid collision.

Probability of Risk The overall risk probability can be calculated due to all the above factors as

$$p^I(h,d) = 1 - [(1 - p_{cr}) \prod_j (1 - p_k^j) \prod_{i \neq I} (1 - p_{co}^i)] \tag{5.2.7}$$

Where I represent the index of that particular UAV on which these calculations are being carried out. Since each vehicle has its own computing capabilities and decides its own path and this makes the problem as decentralized. The crash probability is shown in Figure 5.3.

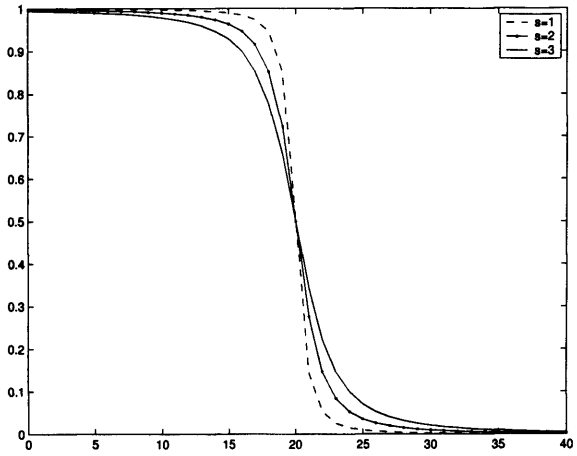


Figure 5.3: Different crash probabilities with $h_{tree} = 20$ and $s = 1, 2, 3$

5.3 Probabilistic Local Minimization

The algorithm is based upon a point search for local minima on a disc whose centre passes through the line of sight of the target from the current point and which is also perpendicular to that line. The radius of the disc is decided by the maximum search angle and which in turn can be decided by the maximum turn angle. The disc is divided into a suitable number of lines all passing through its centre as shown in Figure 5.4 below and local minima points are

searched for along these lines. The distance of the search disc from the current point is chosen according to the type and range of the search sensor mounted on the UAV and all points on the disc should be within the range of the search sensors. The search angle could be different for different search lines in the disk. This is because the maximum climb rate may be different in different directions. To find the coordinates of each point on the disc, it is necessary to specify a known reference line lying in the disc and passing through its centre. The reference line (RL) is selected to be parallel to the horizontal plane. For the two dimensional case, the search is limited to the reference line only but in three dimensions, we have to search the whole disc along various lines. To reduce the computational load, we can also limit our search to the reference line only, which finds a point in 3D in the inclined plane containing the line of sight and reference line and which changes with the line of sight as the search progresses. But since a minimum point can be found at a place which is not along the reference line, so for this purpose other lines are also searched. In the absence of the other searches, although the UAV will find a stealthy path but is limited to the reference line only.

Consider such a disc having its centre at the point $P_c(x_c, y_c, z_c)$ which is at a distance h from the current point $P_i(x_i, y_i, z_i)$ on the line of sight of target having coordinates $P_T(x_T, y_T, z_T)$ as shown in Figure 5.4. The equation of the plane containing the disc and $P(x, y, z)$ representing any point on the plane is given by

$$\overrightarrow{P_iP_T} \,.\, \overrightarrow{P_cP} = 0$$

$$\Rightarrow (x_T - x_i)(x - x_c) + (y_T - y_i)(y - y_c) + (z_T - z_i)(z - z_c) = 0$$

$$\Rightarrow a(x - x_c) + b(y - y_c) + c(z - z_c) = 0 \tag{5.3.8}$$

where

$$a = (x_T - x_i),\ b = (y_T - y_i),\ c = (z_T - z_i) \tag{5.3.9}$$

are the direction ratios of the line of sight of the target. Consider the j^{th} search line in the search disc making an angle of θ_j with the reference line having direction cosines $l, m, 0$. Let $P(x, y, z)$ be the point on the line at a distance r from P_c where r is less than or equal to R_j (the radius of the j^{th}

search line). Then we can write

$$\cos\theta_j = \frac{(x-x_c)l + (y-y_c)m + (z-z_c)0}{\sqrt{((x-x_c))^2 + (y-y_c)^2 + (z-z_c)^2}\sqrt{l^2+m^2+0^2}} \tag{5.3.10}$$

But

$$(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 = r^2 \tag{5.3.11}$$

and

$$l^2 + m^2 = 1 \tag{5.3.12}$$

Hence, equation (5.3.10) can be written as

$$(x-x_c)l + (y-y_c)m = r\cos\theta_j \tag{5.3.13}$$

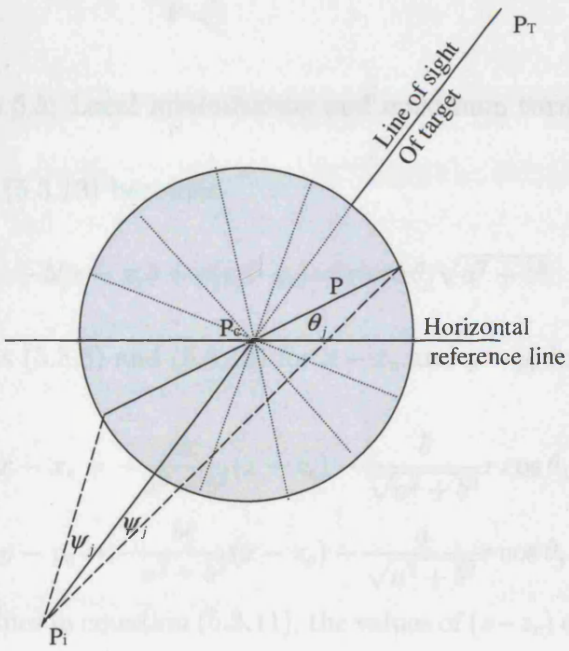


Figure 5.4: Search Disk

$$l.a + m.b = 0 \tag{5.3.14}$$

By solving equations (5.3.12) and (5.3.14), we will get two sets of values representing two parallel unit vectors in opposite directions. Select one set as

$$l = -\frac{b}{\sqrt{a^2+b^2}},\ m = \frac{a}{\sqrt{a^2+b^2}},\ n = 0$$

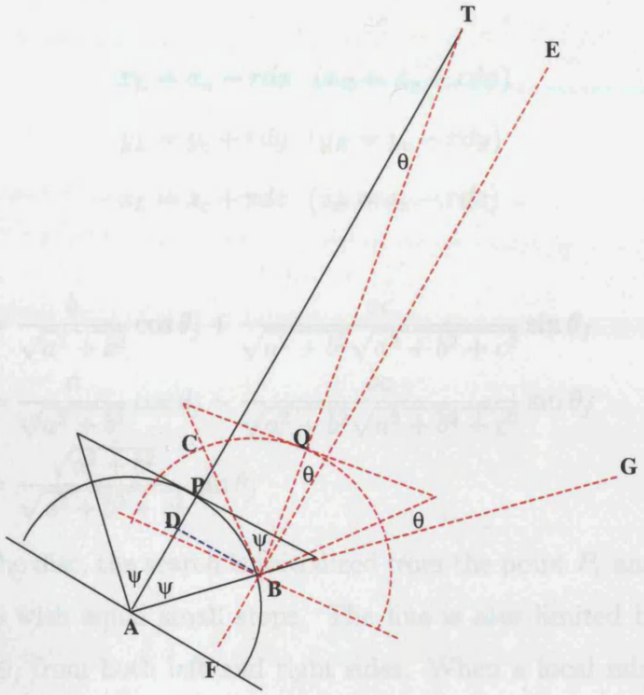


Figure 5.5: Local minimisation and maximum turn angle

Hence, equation (5.3.13) becomes

$$-b(x - x_c) + a(y - y_c) = r\cos\theta_j\sqrt{a^2 + b^2} \tag{5.3.15}$$

Solving equations (5.3.8) and (5.3.15) for $x - x_c$ and $y - y_c$ in terms of $z - z_c$, we have

$$x - x_c = -\frac{ac}{a^2 + b^2}(z - z_c) - \frac{b}{\sqrt{a^2 + b^2}} r\cos\theta_j \tag{5.3.16}$$

$$y - y_c = -\frac{bc}{a^2 + b^2}(z - z_c) + \frac{a}{\sqrt{a^2 + b^2}} r\cos\theta_j \tag{5.3.17}$$

Putting these values in equation (5.3.11), the values of $(z - z_c)$ can be calculated as

$$z - z_c = \pm\frac{\sqrt{a^2 + b^2}}{\sqrt{a^2 + b^2 + c^2}} r\sin\theta_j \tag{5.3.18}$$

So there will be two points $P_L(x_L, y_L, z_L)$ and $P_R(x_R, y_R, z_R)$ on either sides of the search line which are at a distance r from the point P_c and can written in the form

$$
\begin{aligned}
x_L &= x_c - rdx \quad (x_R = x_c + rdx) \\
y_L &= y_c + rdy \quad (y_R = y_c - rdy) \\
z_L &= z_c + rdz \quad (z_R = z_c - rdz)
\end{aligned}
\tag{5.3.19}
$$

where

$$
\begin{aligned}
dx &= \frac{b}{\sqrt{a^2+b^2}}\cos\theta_j + \frac{ac}{\sqrt{a^2+b^2}\sqrt{a^2+b^2+c^2}}\sin\theta_j \\
dy &= \frac{a}{\sqrt{a^2+b^2}}\cos\theta_j - \frac{bc}{\sqrt{a^2+b^2}\sqrt{a^2+b^2+c^2}}\sin\theta_j \\
dz &= \frac{\sqrt{a^2+b^2}}{\sqrt{a^2+b^2+c^2}}\sin\theta_j
\end{aligned}
\tag{5.3.20}
$$

For a line in the disc, the search is initialized from the point P_c and which goes along the line with equal small steps. The line is also limited by maximum search angle ψ_j from both left and right sides. When a local minimum point $P_m(x_m, y_m, z_m)$ is found after searching all lines, the UAV will move to that direction by a distance h. From simulation, it has been observed that by moving exactly to the local minimum point along the chosen safe direction, the vehicle may find itself in a danger zone in the next iteration. That is, although the point is safe, it may be very near to the threat, so it is better to cover a shorter distance (in our case h) to leave a safety margin. The next point of the path can be calculated as

$$
\overrightarrow{OP}_{i+1} = \overrightarrow{OP}_i + h\hat{n}_m
$$

where $\hat{n}_m$ is the unit vector in the direction of the local minima $P_m(x_m, y_m, z_m)$ from point P_i and is calculated by

$$
\hat{n}_m = \frac{[x_m - x_i, y_m - y_i, z_m - z_i]}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2}}
$$

In scalar form, the next decided point to move can be written as

$$
\begin{aligned}
x_{i+1} &= x_i + h\frac{x_m - x_i}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2}} \\
y_{i+1} &= y_i + h\frac{y_m - y_i}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2}} \\
z_{i+1} &= z_i + h\frac{z_m - z_i}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2}}
\end{aligned}
\tag{5.3.21}
$$

The process of local minimization for the case of a circular disc and the strong dependence of the maximum turn angle on the search angle can be seen from Figure 5.5.

From Figure 5.5, the maximum turn angle β is given by

$$\beta = \angle CBG = 2\psi + \theta \tag{5.3.22}$$

Also

$$\tan\theta = \frac{BD}{AT - AD} = \frac{AB\sin\psi}{AT - AB\cos\psi} = \frac{h\sin\psi}{d - h\cos\psi}$$

$$\beta = 2\psi + \arctan\frac{h\sin\psi}{d - h\cos\psi} \tag{5.3.23}$$

where h is the forward step size and d is the distance of the current point from the destination T. For constant h and ψ, the maximum turn angle only depends on d. When d is very large, then $\tan\theta \rightarrow 0 \Rightarrow \theta \rightarrow 0 \Rightarrow \beta \rightarrow 2\psi$. On the other hand β goes on increasing when a UAV approaches its targets. Suppose now that we want to find the optimal path between two points P_0 and P_T in 3D space.

5.3.1 Algorithm Inputs

- Range of the rectangular operational area in terms of minimum and maximum values of the x ,y and z coordinates.
- Radar (threat) locations in terms of x, y and z coordinates.
- Strength of threats e.g., short, medium and long range missile
- Starting and target points P_0 and P_T respectively.
- Risk threshold α. This threshold is in the sense that the risk evaluated at some point due to all radar sites using relation (5.2.7) is less than this value. The UAV is then considered to be in a safe region and one would like to continue in the same direction without searching the disk at that point. This user given value can be used to adjust the tolerable risk one can afford. By decreasing this value, very long but safe paths

result; where as increasing it gives short dangerous paths. Hence, this parameter serves as a compromise between threat avoidance and path length (fuel consumption).

- Forward step size h and lateral step angle $\delta\psi$. The forward step size is used to move in the selected direction while the lateral step angle is employed for search purposes along the lines in small steps.
- A vector $[\theta_j]_{j=1}^{jmax}$ of search directions
- A vector $[\psi_j]_{j=1}^{jmax}$ of maximum angles of search along each search line starting from a point on the line of sight of the target at a forward step size distance and then moving on either side of that point for each line. The angles should be less than 90° because otherwise the vehicle might then divert from its path. A longer but safer path results by choosing a higher value of these angles and vice versa
- Tolerance is used to terminate the algorithm. When the UAV is near to the target, then instead of converging, it begins to loiter. So this limit is set to avoid this phenomenon. When the vehicle is within this tolerance, then it is assumed that the target is achieved.

5.3.2 Algorithm Description

Step 1 Initialize the path vector with starting point and declare it the current point and set the indices $i = 1$, $j = 1$, $d\psi = \delta\psi$.

Step 2 Find the distance $\sqrt{(x_T - x_i)^2 + (y_T - y_i)^2 + (z_T - z_i)^2}$ of the current point to the target. If this distance is less than the tolerance limit, go to last Step 10. Otherwise go to next step.

Step 3 Find the direction ratios a, b, c of the line of sight of the target using (5.3.9).

Step 4 Find a point P_c on the line of sight at a distance h from the current point P_i using (5.3.21) (P_m should be replaced by P_T) and calculate the risk value p_c at this point using the relation (5.2.7).

Step 5 If this value of the risk probability is less than the threshold α, then there is no need to worry about the safety of this point. Therefore add the point to the list, set $i = i + 1$ and go to Step 3. Otherwise, declare it to be an expected point of the optimal path by assigning it to a point P_e and go to the next step.

Step 6 Set $r = h \tan d\psi$ and find two points P_L and P_R on the left and right side respectively of the point P_c at a search angle $\delta\psi$ along the j^{th} search line which makes an angle θ_j with the reference line using (5.3.20) and (5.3.19). Find the risk probabilities p_L and p_R at these points using the relation (5.2.7).

Step 7 If the risk p_e is less than or equal to the risks p_L and p_R, then go to step 10. Otherwise go to the next step.

Step 8 If the risk probability p_L (p_R) is less than the risk p_e and also it is less than or equal to the risk p_R (p_L), then P_L (P_R) is the minimal risk point among these points and there is a chance of getting a further low risk point on the left (right) side. Repeat the following:

$$x_e = x_L \quad (x_e = x_R)$$

$$y_e = y_L \quad (y_e = y_R)$$

$$z_e = z_L \quad (z_e = z_R)$$

$$d\psi = d\psi + \delta\psi$$

$$r = h \tan d\psi$$

$$x_L = x_L - rdx \quad (x_R = x_R + rdx)$$

$$y_L = y_L + rdy \quad (y_R = y_R - rdy)$$

$$z_L = z_L + rdz \quad (z_R = z_R - rdz)$$

until one of the following occurs:

- the search angle exceeds its maximum value which means $d\psi > \psi$
- a local minimum is found which means risk $p_e <$ risk p_L (p_R)

Step 9 If $j < j_{max}$, then assign point P_e to P_j and the risk p_e to p_j and set $j = j + 1$, $d\psi = \delta\psi$ and go to step 6. Otherwise find the minimum risk point $P_m(x_m, y_m, z_m)$ on the disk by comparing the minimum risk p_j found from all the search lines. Set $i = i + 1$ and find new safe path point using (5.3.21). Add this point to the path list and also set $j = 1$, $d\psi = \delta\psi$. Go to Step 2

Step 10 Output the result in the form of

- Optimal path which consists of way points.
- Total and average risk on this path

5.4 Simulation Results

The example scenario used here consists of 23 radar threats scattered in a 300 km $\times$ 300 km rectangular region of the xy-plane. The radars are considered to be medium range SAM units with a range varying from 15 km to 35 km. Table 5.1 shows their positions and ranges. For ease of demonstration, the approach will first be applied to a problem of finding a path in two dimensions and later it will be extended to the three dimensional case. For this purpose a plane parallel to the xy- plane at a height of 10 km was selected. The starting point of the UAV is $(10, 100, 10)$ and the destination is $(250, 240, 10)$. The parameters of maximum lateral search angle, lateral search angle step, path step, and stopping tolerance were first taken as fixed with values of are 60°, 5.7°, 1 km and 2 km respectively. The risk threshold parameter was varied by taking values $0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$. Figure 5.6 shows three different paths obtained for three different values of the threshold parameter α. Contours around each threat show different risk levels varying from 0.1 to 0.9 with the outer most showing a hit probability of 0.1 and the inner most a value of 0.9. By increasing the value of the risk threshold α the path becomes shorter and shorter and for $\alpha = 1$ it becomes a straight line. The variation of distance covered and average risk with threshold can be seen in Figure 5.7 and 5.8, respectively.

X(km)	Y(km)	Z(km)	Range(km)
142.00	270.84	0	15
135.32	241.36	0	20
248.66	49.88	0	25
118.17	156.23	0	25
215.44	170.76	0	30
170.00	140.00	0	20
26.32	133.05	0	30
109.89	90.76	0	35
230.00	120.00	0	20
284.93	167.38	0	15
4.27	178.85	0	15
244.86	293.13	0	20
66.57	211.11	0	20
156.62	279.87	0	20
214.01	68.41	0	25
134.89	51.66	0	25
180.00	20.00	0	30
220.00	230.00	0	30
70.00	130.00	0	30
80.00	260.00	0	25
166.00	190.00	0	35
280.00	80.00	0	35
60.00	30.00	0	15

Table 5.1: Radars locations and their ranges

The target assignment aspects of the algorithm can be explored by visiting more than one target in some specific order. Consider a UAV which starts at a position $(250, 240, 10)$ and intends to visit a set of targets $(110, 270, 10)$, $(260, 200, 10)$, $(213, 38, 10)$, $(130, 160, 10)$. We will label these points in the order given as $1, 2, 3, 4$ respectively. We are interested in finding a path to

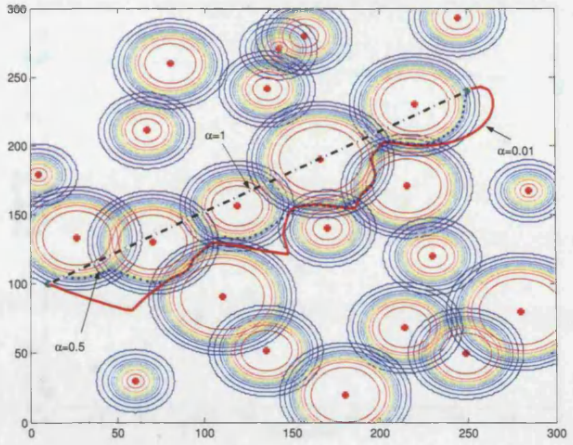


Figure 5.6: Paths obtained for thresholds, $\alpha = 0.01, 0.5, 1$

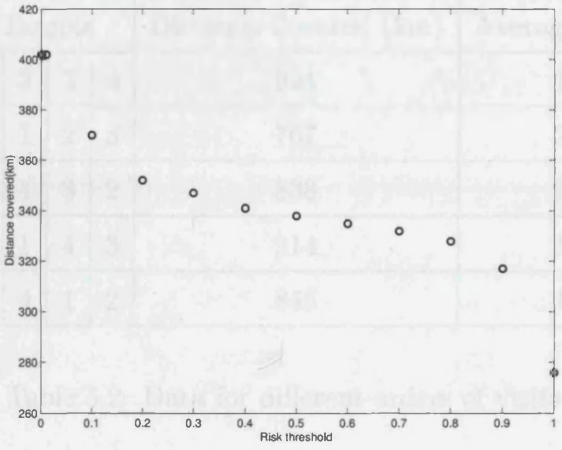


Figure 5.7: Variation of distance covered with threshold

visit these targets and the effect on the path and other parameters of different sequences. The results are summarized in Table 5.2 and the paths for fire target sequeces are shown in Figures 5.9 to 5.13. It is interesting to note that in the first order of visits when the vehicle is on its way to visit the second target after visiting first target, then at one time it comes within the range of three SAM units. Sometimes this situation cannot be avoided and one way to minimize the risk to a UAV during such situations is to pass as quickly as possible with maximum speed.

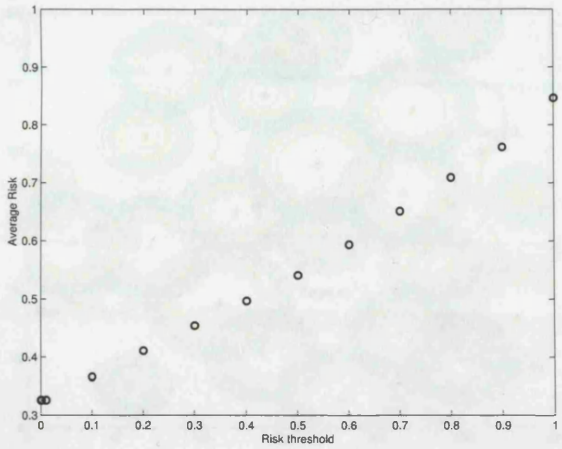


Figure 5.8: Variation of Average Risk with threshold

Order	Targets				Distance Covered (km)	Average Risk (%)
1	1	2	3	4	924	19.48
2	4	1	2	3	767	23.52
3	1	4	3	2	838	20.45
4	2	1	4	3	914	16.02
5	3	4	1	2	845	16.85

Table 5.2: Data for different orders of visits

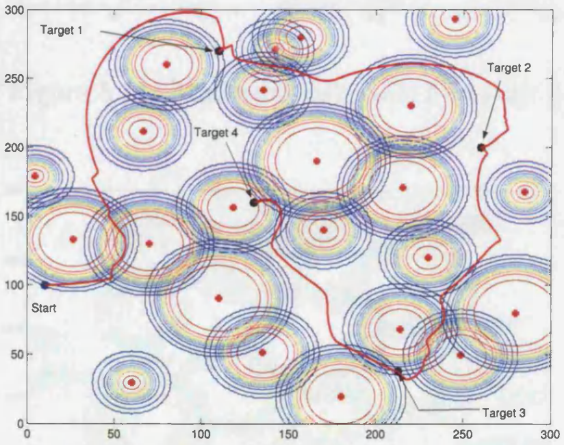


Figure 5.9: Minimum risk path for order 1

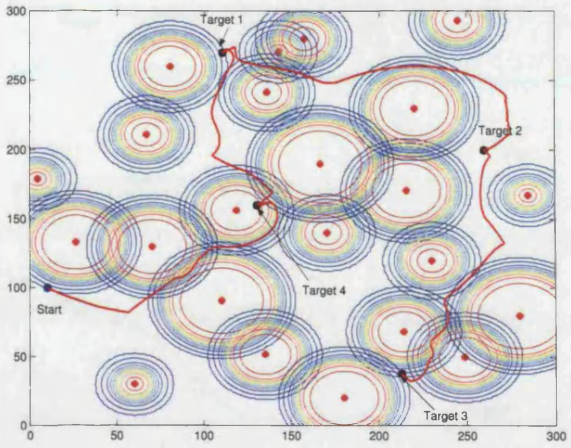


Figure 5.10: Minimum risk path for order 2

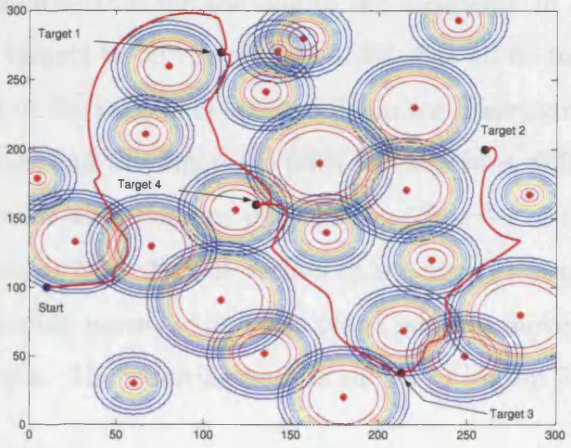


Figure 5.11: Minimum risk path for order 3

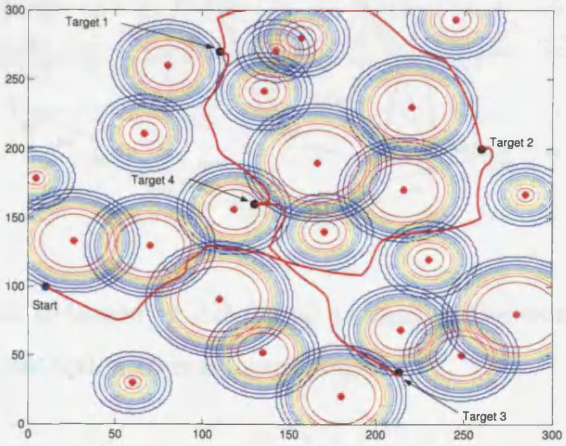


Figure 5.12: Minimum risk path for order 4

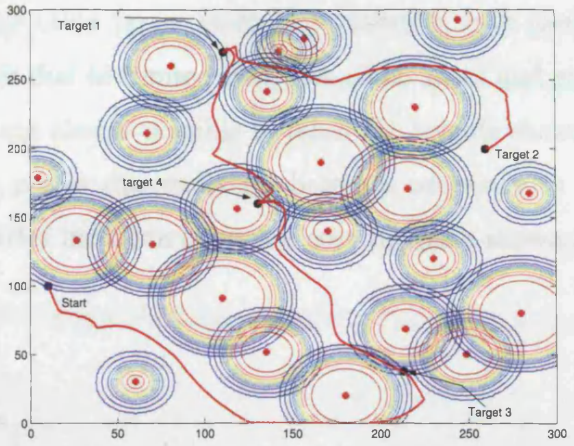


Figure 5.13: Minimum risk path for order 5

In order to demonstrate the working of the algorithm in $3D$, consider the case of multiple targets to be visited by a UAV that all lie in the same plane. The same height of the targets is selected again for demonstration purpose in 3 dimensional space and the way point path for targets at different heights can obtained in a similar way. For this purpose the 5^{th} order of target visits from Table 5.2 is selected. Now the full version of the algorithm is applied so that the UAV can not only move in the same plane but can move up and down to find a stealthy path. The resulting path is shown in Figure 5.14.

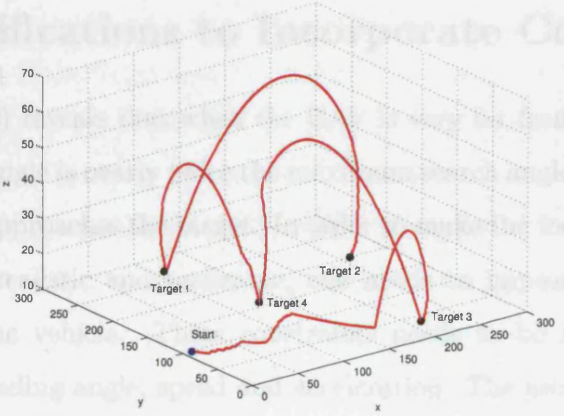


Figure 5.14: Path obtained in $3D$ for all targets in the same plane without considering any restriction to maximum height

On its approach to the third target, the UAV remained close to the horizontal plane but when it got near to the target it started moving up due to the

high risk. For the other target visits high altitudes were used to escape from danger. However due to limited fuel, low climb rates and sometimes timing constraint, it is not always possible to reach the heights shown in Figure 5.14. For this reason, a cost due to height becomes necessary to include into the cost function. After inclusion of such a cost the path shown in Figure 5.15 is obtained.

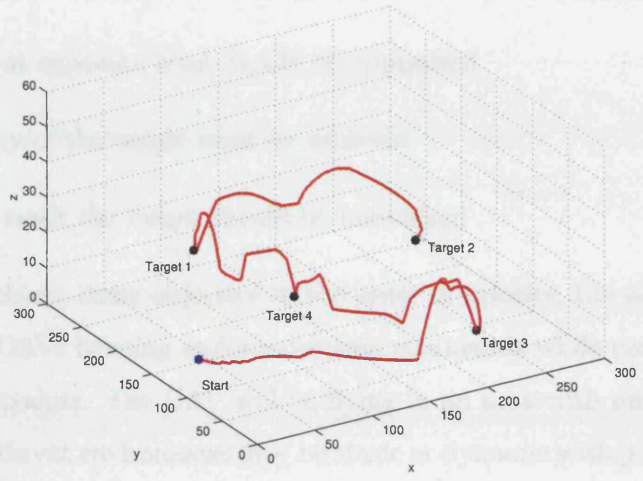


Figure 5.15: Path obtained in $3D$ for all targets in the same plane and considering a cost due to height

5.5 Modifications to Incorporate Constraints

Equation (5.3.23) reveals that when the UAV is very far from the target, the maximum turn angle is nearly twice the maximum search angle and it increases when the UAV approaches the target. In order to make the local minimization approach more realistic and realizable, one needs to impose the dynamical constraints of the vehicle. These constraints needs to be implemented for turn angle or heading angle, speed and acceleration. The proposed algorithm will be modified to include these constraints. The UAV cannot change its heading instantaneously. The rate of change of heading is limited by the vehicle dynamics. This can modelled by a maximum heading constraint ψ_{max} which is the maximum heading angle change the UAV can make in each simulation step. ψ_{max} is a function of the speed. With this strategy, the current UAV

heading will always lie along the symmetrical axis of a cone in 3D, which moves with the UAV. The UAV heading at the next time step will along the point of minima within this cone. The heading search procedure is illustrated for the three different cases as shown in Figures 5.16, 5.17 and 5.18. The strategy is designed taking into consideration the following four main objectives:

- restricted areas should be avoided
- the threat exposure level should be minimized
- proximity of the target must be achieved
- time to reach the target should be minimized

In order to achieve these objective in the order of priority, the algorithm will generate the UAVs heading and acceleration commands while considering the dynamic constraints. The UAV will be flying in an area with multiple threat sources. The threat environment may be static or dynamic with popup threats. So when the UAV is flying in an area with multiple threat sources with time varying properties, then probability of risk is a function of time as well as position. However, once the probabilistic risk map is constructed, it is not necessary to distinguish between moving and stationary threats, only the local value of the map is used to minimize the threat exposure. As mentioned above, the next position of the UAV can be varied by changing the heading of the UAV and the time to reach the next position can be varied by changing the speed of the UAV. If the UAV maintains its current speed V_c, the incremental distance that the UAV travels from its current position P_i to the next position P_{i+1} until the next computation time, is computed by the current speed and simulation time step Δt as shown in Figure 5.19. This incremental distance and the heading of the UAV along with the current position will determine the next position of the UAV. The speed variation is limited by the acceleration constraints because the UAV needs to reach the next position in a specified amount of time which is the simulation step time. The acceleration range that will not violate the speed constraints of the UAV is calculated as follows.

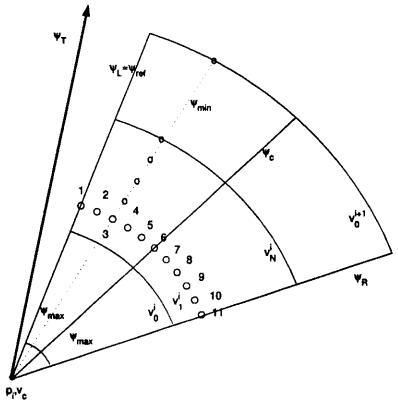


Figure 5.16: Local minimization scheme with constraints when the target is outside the search cone and is nearest to the left boundary of the cone

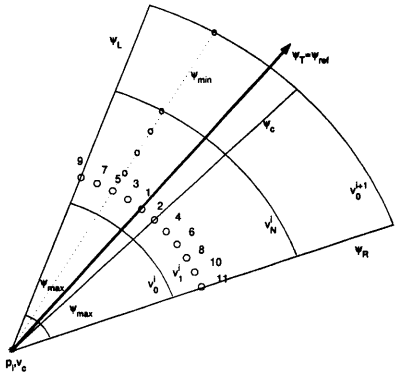


Figure 5.17: Local minimization scheme with constraints when the target is inside the search cone.

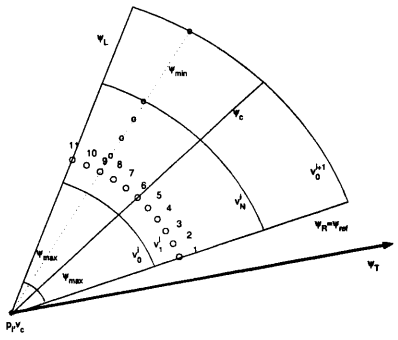


Figure 5.18: Local minimization scheme with constraints when the target is outside the search cone and is nearest to right the boundary of the cone

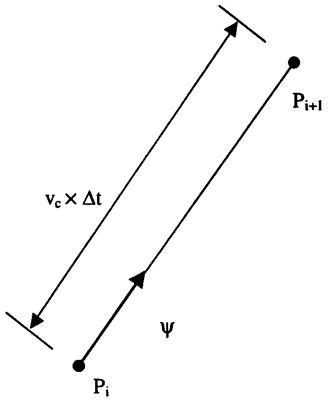


Figure 5.19: Incremental distance in one simulation step

Assuming that the UAV maintains a constant acceleration in the next simulation interval

$$a_{v_{max}} = \frac{v_{max} - v_c}{\Delta t} \tag{5.5.24}$$

which is the acceleration that will increase the UAV speed to its maximum value. Similarly

$$a_{v_{min}} = \frac{v_{min} - v_c}{\Delta t} \tag{5.5.25}$$

which is the acceleration that will reduce the UAV speed to its minimum value. So the acceleration range due to the speed constraints is defined by

$$R_{a_1} = [a_{v_{min}}, a_{v_{max}}] \tag{5.5.26}$$

The UAV acceleration should be in this range, because otherwise the UAV speed will be higher than v_{max} or lower than v_{min}, which is a direct violation of the speed constraints. But the given range of acceleration constraints is

$$R_{a_2} = [a_{min}, a_{max}] \tag{5.5.27}$$

Hence the admissible acceleration range is the intersection of these two ranges

$$R_a = [a_{lower}, a_{upper}] = R_{a_1} \cap R_{a_2} \tag{5.5.28}$$

which implies that

$$a_{lower} = \begin{cases} a_{v_{min}}, a_{v_{min}} \geq a_{min} \\ a_{min}, a_{v_{min}} < a_{min} \end{cases} \tag{5.5.29}$$

$$a_{upper} = \begin{cases} a_{v_{max}}, a_{v_{max}} \leq a_{max} \\ a_{max}, a_{v_{max}} > a_{max} \end{cases} \tag{5.5.30}$$

where a_{lower} is the lower limit and a_{upper} is the upper limit on the admissible acceleration range. The distance travelled in time interval Δt will be maximum if the UAV moves with acceleration a_{upper} and minimum if the acceleration is a_{lower}. So the admissible distance range is

$$R_d = [d_{lower}, d_{upper}] \tag{5.5.31}$$

where

$$d_{lower} = v_c \Delta t + 0.5 * a_{lower} \Delta t^2 \tag{5.5.32}$$

$$d_{upper} = v_c \Delta t + 0.5 * a_{upper} \Delta t^2 \tag{5.5.33}$$

Also by considering the acceleration range, the admissible speed range is obtained as follows:

$$R_v = [v_{lower}, v_{upper}] \tag{5.5.34}$$

where

$$v_{lower} = v_c + a_{lower} * \Delta t \tag{5.5.35}$$

$$v_{upper} = v_c + a_{upper} * \Delta t \tag{5.5.36}$$

Due to these considerations, the previous algorithm will be modified. For the 2D case, two searches are involved: a heading search and an acceleration search. The heading search is constrained on a circular arc shown by dicrete dots in Figures 5.16 to 5.17. These dots are labelled with numbers showing the order of search and are at a distance $d_{lower} + \delta d$ from the current position, where d_{lower} is the minimum distance for a UAV to reach. Search is not useful below the distance d_{lower} because a UAV with dynamical constraint must cover this distance in time interval Δt and the small distance δd is for UAV safety considerations. One objective is to go to the destination point in minimum time which can be accomplished if the UAV moves with maximum possible acceleration in Δt. The acceleration search starts from a_{lower} to a_{upper}. In other words, the search starts from d_{lower} and moves towards d_{upper} in small steps. The search finishes when the risk becomes greater than its value at the previous search.

5.6 Temporal Constraint

Driven by battle space management needs, real world threats can be much more complex than those modelled in the previous trajectory generation work. In an area that is dense with threats, path planning that relies primarily on threat zones avoidance may be insufficient to achieve the simultaneous goals of reaching to a destination way point in a high risk area and avoiding threats. On the other hand by modelling and managing the observability of the paths, path planning algorithm can be designed that have the potential to achieve these objectives. The detectability of an aircraft travelling near an enemy radar depends on more than just the distance to the radar, it also depends on the UAV attitude and configuration. This feature in the threat model introduces non-convexities and path dependencies as well as sharp gradients in the underlying optimization problems for trajectory generation. Therefore we propose temporal constraints or precedence constraints that include performing an activity only after another activity has been performed or performing simultaneous activities. The low observability routing problem could be considered to be temporal in nature by allowing periods of high observability interspersed with periods of low observability. This is desirable because of the way the enemy's systems work. Although it might not be possible to get close to an opponent's territories while maintaining low observability at all times; by strategically flying low observable paths for part of the time, it might be possible to drive the enemy's systems into a condition called lock-loss. This condition aborts the enemy's plans after a specified time of no detection. If an opponent detects a UAV, then an engagement is initiated, the only way the enemy will disengage is if there is a period of time (lock-loss time) in which the UAV may trigger the opponent into disengaging if high observability flight is constrained to occur for limited durations. Incorporating features into the path planner where high observability times are limited and interspersed with low observability times can be a beneficial strategy. So in regions where there are multiple radars, path planning incorporating risk and temporal constraints in this manner may be required to satisfy the requirements.

5.7 Comparison With the Other Strategies

The effectiveness of adjusting UAV acceleration as well as heading based on the current map information can be seen by comparing the proposed algorithm (PA) (as explained in the previous section) with two other strategies:

- Heading control with current map (HCCM): With this strategy, at each computation, the current probabilistic map is used to find the heading that would minimize the threat exposure level. The strategy uses the same algorithm as PA to conduct the heading search. Even if this strategy is provided with the current map it does not adjust its acceleration and the UAV maintains its initial speed throughout the mission.

- Heading control with initial map (HCIM): This strategy is the same as HCCM except it is provided with only the initial map of the area. Thus even if the probabilistic map is changing, it makes decisions for the commanded heading based on the initial status of the threats. Furthermore, it also commands the initial speed throughout the mission.

For comparison purposes, two threats are considered to be time-variant. One of the threats has a changing the concentration point, i.e. location. This threat is initially located at $(-2, 4)$ and moves along the positive y-axis as shown in Figures 5.21-5.30. The other threat is located at $(-20, 0)$ and has a changing area of effectiveness, i.e. the range of the SAM between 90 and 150 minutes. This threat initially has a range of 30 km. As the time increases from 90 minutes to 120 minutes, the range decreases linearly to 15 km and goes back to its initial value linearly between 120 and 150 minutes. The PA trajectory is shown with a solid line, the HCCM trajectory is shown with a dashed line and the HCIM trajectory is shown with a dashed-dotted line in Figures 5.21-5.30. For this simulation α and β are selected to be 0.0001 and 0.005 respectively which are also shown in Figures 5.21-5.30. As seen from Figures 5.21-5.24 all three strategies command the same trajectories up to 50 minutes. This can also be seen from Figure 5.20 where it is clear they have the same threat exposure level up to 50 minutes. When there is a high threat area approaching the UAV, the UAV using the PA starts to manoeuvre earlier than the others

to avoid the approaching threat. At 60 minutes the UAV with PA starts to manoeuvre to the right and flies with a speed 9% higher than its initial speed This is because there is a threat approaching the UAV. In other words, the threat exposure level is increasing in the direction the UAV is heading. Since PA recognises that the threat level will increase, it changes the heading and increases the speed to pass through the region before the probability in that region gets higher. Since HCCM uses only the current map, it recognises the approaching threat later than the PA does and starts to manoeuvre later. Also, it cannot adjust speed and therefore it is exposed to a higher level of threat than the PA. However, HCIM is exposed to the highest threat level, since it does not know anything about the time-variation of the map as it passes through the threat region, which is shown in Figures 5.21-5.24. It is also shown in Figure 5.20 that the threat exposure level increases significantly in HCIM as the threat approaches. HCCM has a smaller threat and PA the least. As seen from Figure 5.20 between 85 and 110 minutes, as the UAVs approach the second target position, there is another significant increase in the level of threat exposure. This is because the UAVs pass between two threat regions to get to the target as seen from Figures 5.25-5.28. Since the UAVs guided by HCCM and HCIM pass from this region before the UAV guided by PA, they are exposed to the threat region earlier. They are also expose to a higher level of threat than the UAV guided by PA.

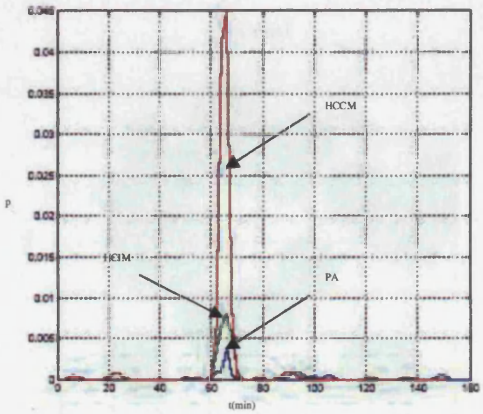


Figure 5.20: Threat exposure level along the trajectory of each strategy

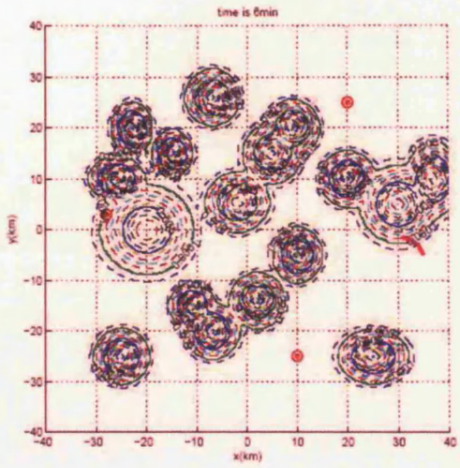


Figure 5.21: Comparison of the trajectories at time 6 min

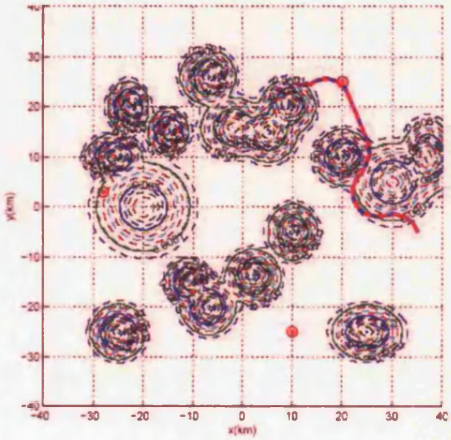


Figure 5.22: Comparison of the trajectories at time 50 min

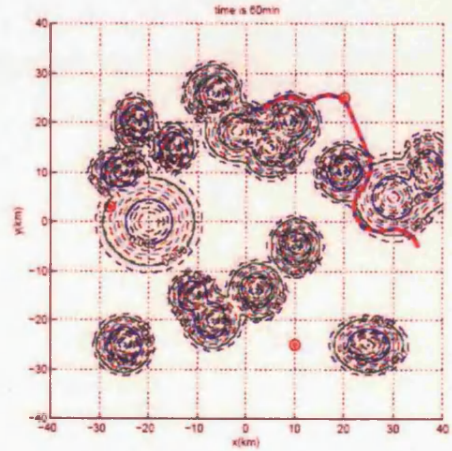


Figure 5.23: Comparison of the trajectories at time 60 min

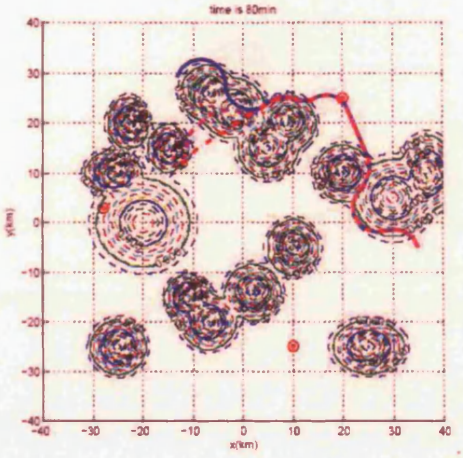


Figure 5.24: Comparison of the trajectories at time 80 min

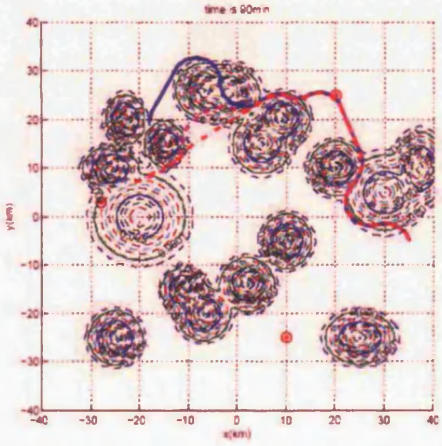


Figure 5.25: Comparison of the trajectories at time 90 min

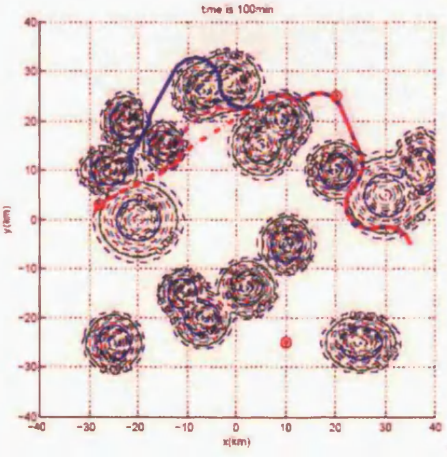


Figure 5.26: Comparison of the trajectories at time 100 min

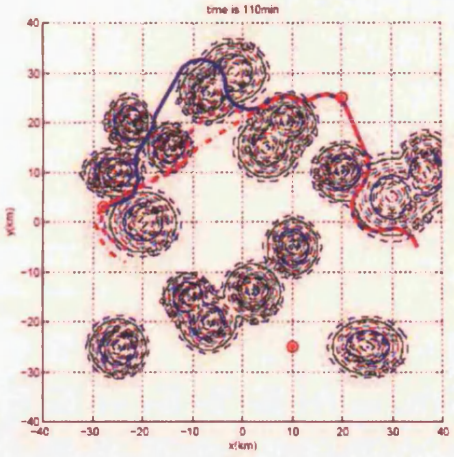


Figure 5.27: Comparison of the trajectories at time 110 min

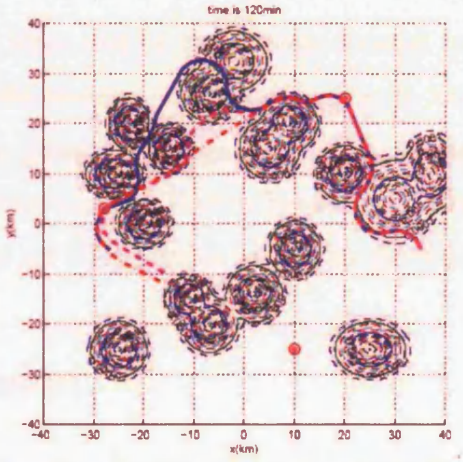


Figure 5.28: Comparison of the trajectories at time 120 min

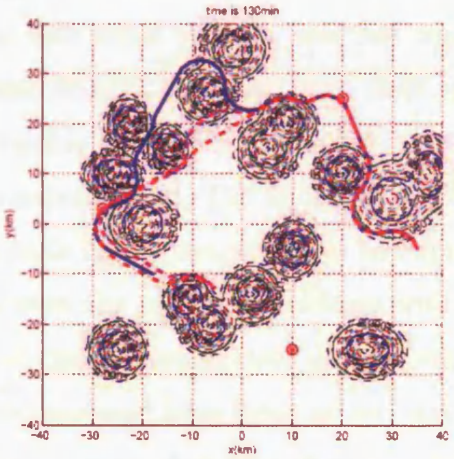


Figure 5.29: Comparison of the trajectories at time 130 min

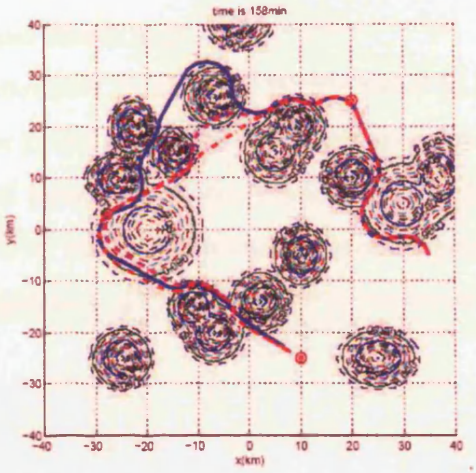


Figure 5.30: Comparison of the trajectories at time 158 min

This is because, since one of the threats located in front of the target changes its area of effectiveness, the threat exposure level decreases between 90 and 120 minutes. Thus, PA recognises that the threat exposure level will decrease in the direction that the UAV is heading and thus decreases the UAV speed significantly to get to the region where the probability gets smaller.

5.8 Conclusion

A real time three dimensional probabilistic approach for the path planning of autonomous vehicles has been presented. It is not only capable of finding a safe path but also takes into account real world constraints. The problems of collision avoidance with other vehicles, low fuel consumption and crash prevention with ground objects have also been dealt with. The paths are locally optimal and feasible for the UAV to follow by keeping the turn angle within some certain maximum limit. The algorithm is applied in decentralized mode, that is, each vehicle has its own processor and applies the algorithm to find its path. It addresses the collisions avoidance with consideration of the collision with other vehicles by keeping itself at some minimum distance from others. The UAVs are prevented from flying at very low altitudes because of the danger of crashing with ground objects. Since each UAV has some limited fuel, a compromise has to be made between risk and fuel consumption by

limiting the height and search angle.

The real time techniques discussed so far are based on local optimizations and perform well for planning within some limit area at each time step. If we have some global knowledge of the world then this information can be combined with the local optimization technique for improved optimality. In the next chapter, a graphical global technique is presented.

Chapter 6

Global Optimality of Flight Path

6.1 Introduction

Without complete knowledge of the environment an agent can only plan a path which is optimal with respect to the information at the time of planning. But if some global information is available, then it can be incorporated in the planning process for improved optimality. Hence the need for global planners. Deterministic approaches are used most often in global path planners. Two deterministic approaches are: the Voronoi diagram and the Visibility graph. In this chapter a hybrid approach is described which is based on the Voronoi graph, local optimization and grid search methods.

6.2 Voronoi Diagram Method

6.2.1 Voronoi Graph

This procedure, which is used in many different fields including computational fluid dynamics, computer graphics and statistics, begins with complete knowledge of the number and locations of the radar sites. Such a graph is constructed using Delaunay triangulation and its geometric dual, Voronoi polygons. For every triplet of radar sites, there exists a unique circle that passes through

all three. Consider only those triplets whose circle does not enclose any other radar site. The set of all such triplets is called the Delaunay triangulation and the centres of the circles are called Voronoi points. We may now construct a graph by defining the vertices as the Voronoi points. Edges are drawn to connect two Voronoi points if and only if their associated Delaunay triangle share an edge. By drawing all such edges, we construct the Voronoi diagram or graph. Figure 6.1 a shows Voronoi diagram for 30 threats spread over a 250×250 square unit area and Figure 6.2 shows its geometric dual. The edges of the Voronoi diagram have the property that they are equidistant from a pair of radar sites. One important property of the Voronoi polygon is that it is the perimeter of the set of all points in the plane closest to the Voronoi point containing it.

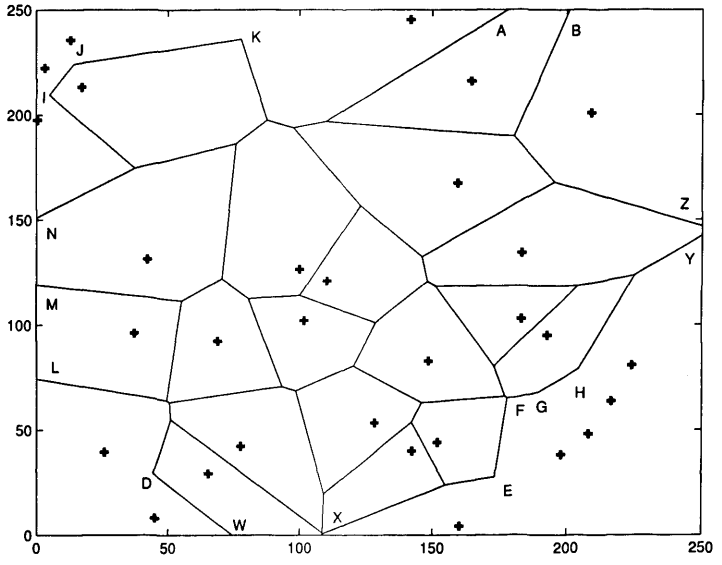


Figure 6.1: Incomplete Voronoi diagram for 30 threats

6.2.2 Matlab Code to Generate the Voronoi Graph

The construction of Voronoi graphs has been implemented in Matlab. However, Voronoi function (`voronoin.m`) in the Matlab is not complete in the sense that it does not divide the operational area completely and explicitly. As seen in Figure 6.1, it stops at some nodes such as the points D, E, F, G, H, I, J, K referred to as the infinity nodes in Matlab routine which are connected to infin-

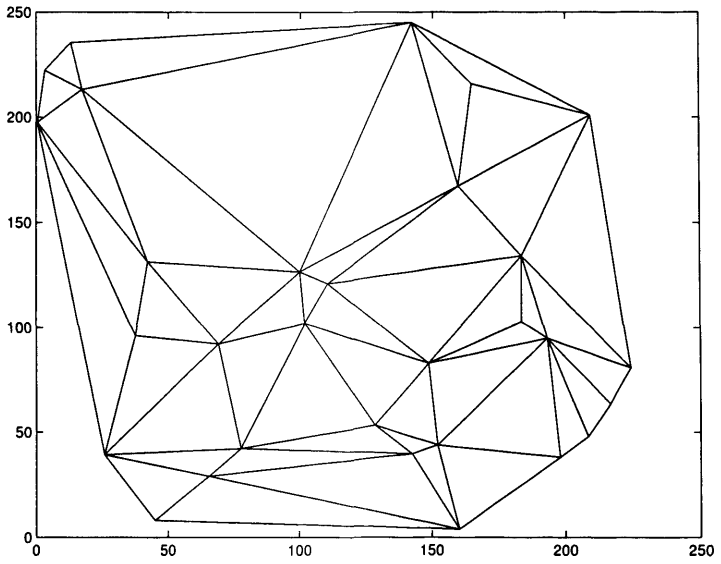


Figure 6.2: Delaunay Triangulation for 30 threats

ity. There should be straight lines from these points extending to the boundary of the area. On the other hand, the Matlab routine produces such nodes which are actually beyond the area considered. When the graph is restricted to the area under consideration, the intersection points A, B, L, M, N, X in Figure 6.1 between Voronoi edges and area borders are not included in the Voronoi node list. As can be seen the intersection points of the edges that have L, M intersection point with the boundary will be very far away from boundary. Since for Voronoi diagram path planning, the paths are restricted to line segments connecting the Voronoi points, so it would be impractical if not impossible for a UAV to travel along such long line segments.

6.2.3 Complete Voronoi Algorithm

Extra subroutines are hence needed and have been developed at Leicester (see Appendix A). These subroutines expand the node list by adding nodes on the borders for those segment connected to the infinity nodes and by inclusion of corner points. Furthermore, they remove those nodes outside the considered area and include instead the intersections of the corresponding edges and the border lines. Hence, the area is explicitly and completely partitioned into cells. The number of the cells is the same as that of the threats, and each threat

is located within one cell. The threat in a cell is the nearest one, among all threats, to any point in that cell. Several steps are taken in this completion of Voronoi graph coding. They are explained below.

We first describe what is meant by an infinity node. The infinity node is an imaginary node and is labeled as Node 1 in the Matlab code. Each cell has a list of nodes which form the perimeter of the polygon of the cell. A cell containing the infinity node in its node list indicates that its perimeter is not complete/closed. In other words, the corresponding threat is not yet completely separated from other threats. In order to separate this threat from others, at least one straight line, sometimes more lines, should be drawn from one of its nodes (the one either in front of or after the infinity node in the cell node list) to infinity.

The first step in our coding is to remove the infinity node from each cell's node list. This step consists of the following parts.

- Expand the operational area to include all finite Voronoi nodes (as interior points) to form a lager rectangular region.
- In a cell with the infinity node, identify the neighbouring nodes of the infinity node and search for other cells with the infinity node and with the same neighbouring node.
- Draw a middle line between the two corresponding threat points and record the intersections with the boundaries.
- Decide which intersection point is required and append it to the total list of Voronoi nodes.
- Modify the node lists of those two relevant cells and the connection map which shows links between Voronoi nodes.

The second step is to include the 4 corners of the expanded rectangle in the total Voronoi nodes. The following have to be applied to each corner point in turn:

- Check if the corner point is already in the total node list; if not, identify the threat point to which the corner is the nearest, i.e. the cell to which the corner point should belong.
- Insert the corner point at the right place in the node list of that cell.
- Append the corner point to the total node list and modify the connection map.

The third step deals with those nodes beyond the originally required operation (rectangular) region.

- In the total Voronoi node list, find out all the nodes which are beyond that required region and define them as the outside nodes.
- Using the connection map, decide all the Voronoi edges, and corresponding nodes, which link an outside node.
- Check if that edge intersects with the (required) boundary lines.
- Identify the cells which ought to have the intersections and modify the node lists of the cells.
- Append the intersections to the total Voronoi node list and modify the connection map.
- Delete all outside nodes from the total Voronoi node list and tidy up the connection map.

The fourth step is similar to Step 2, i.e. to include the corner points of the required region as Voronoi nodes. In the scenario considered here, a UAV flying along an edge is as far as it can be from the two nearest threats.

The complete Voronoi diagram for this selection of threat points is shown Figure 6.3. As can be seen from this figure, these subroutines expand the node list by adding nodes:

- $D1, E1, F1, G1, H1, I1, J1, K1$ on borders for the line segments connecting each other at infinity

- $A1, B1, D2, X1, L1, M1, N1$ for line segments connecting very far away
- Corner points S, T, U, V of the rectangular operational area making it a complete polygonal cell

It is worth while to note that node D is connected to two new nodes $D1, D2$ where $D1$ is the node for the line segment intersecting at infinity and $D2$ is the node for the line segment intersecting at a far away point. We tested the algorithm for different scenarios and found that it worked perfectly. Figures 6.4-6.7 show the working of the algorithm for the cases of 50 and 100 randomly created scenarios.

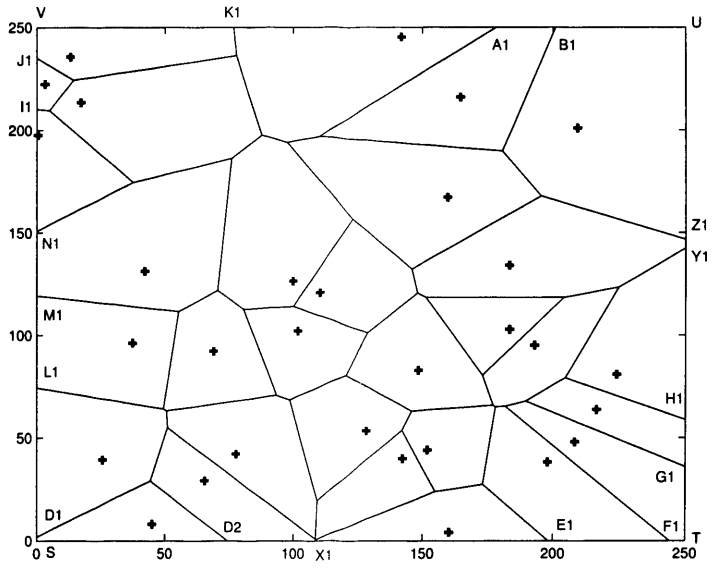


Figure 6.3: Complete Voronoi diagram for 30 threats

6.2.4 Extended Voronoi Graph

Once the complete Voronoi graph is constructed using threat locations, the next stage is to augment it with starting and end points of the vehicle. The augmented graph is called the extended Voronoi graph. A few ways can be used to link these two nodes with others. They can be simply connected to the nearest node [69], they can be linked to the nearest edge or additional edges are defined by the shortest distance lines from the starting (end) point to each

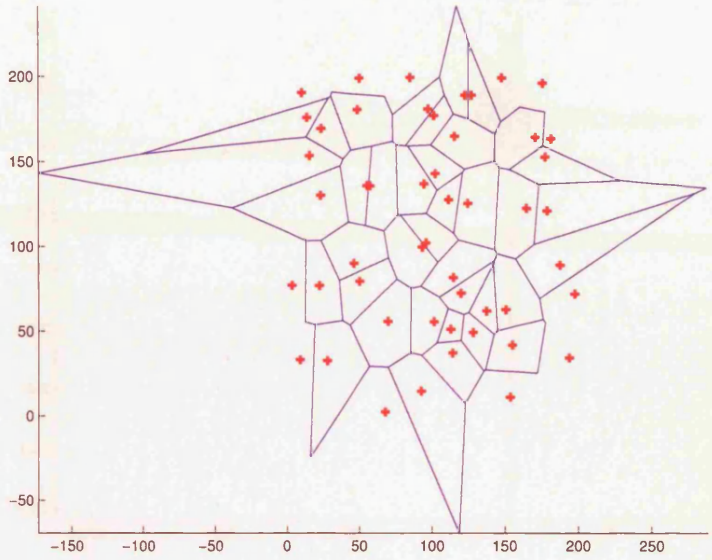


Figure 6.4: Incomplete Voronoi graph for 50 randomly generated threats scattered in 200 × 200 square unit area

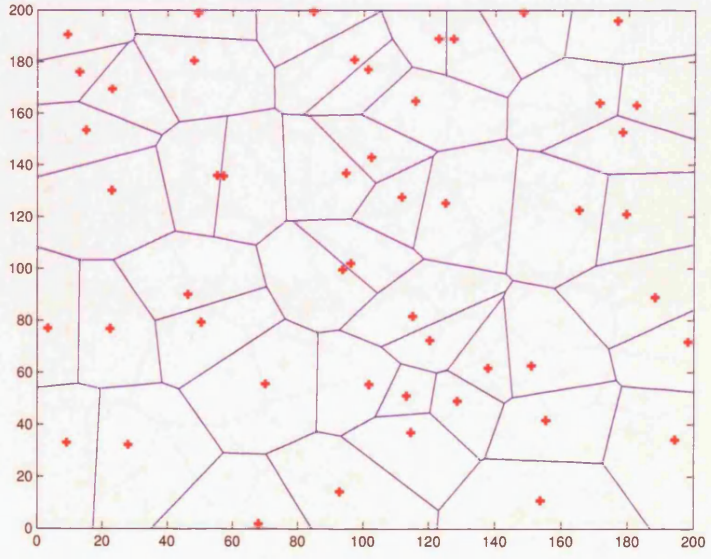


Figure 6.5: Complete Voronoi graph for 50 randomly generated threats scattered in 200 × 200 square unit area

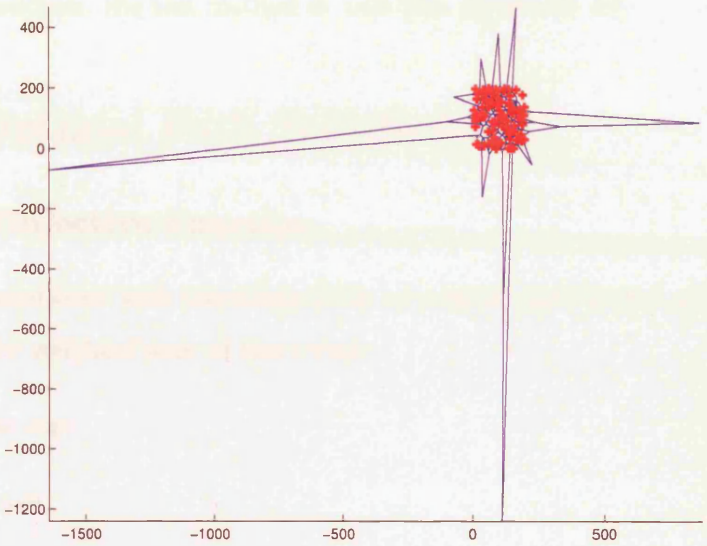


Figure 6.6: Incomplete Voronoi graph for 100 randomly generated threats scattered in 200×200 square unit area

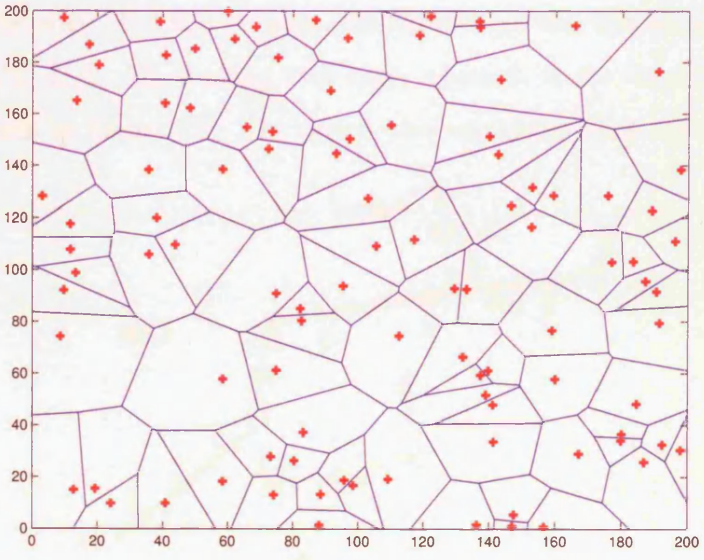


Figure 6.7: Complete Voronoi graph for 100 randomly generated threats scattered in 200×200 square unit area

edge of the Voronoi cell in which the starting (end) point is located. In the following sections, the last method is used (see Appendix A).

6.3 Optimal Path Selection

6.3.1 Objective Function

The cost associated with travelling along an edge of the Voronoi graph can be taken as the weighted sum of two terms:

- threat cost
- fuel cost

The threat cost is based on a UAV's exposure to enemy radar which is the strength of a UAV's radar signature and is proportional to $1/d^4$, where d is the distance to the threat. An exact threat cost calculation for travelling along an edge would involve the integration of the cost along the edge. In order to reduce the computational load, a simpler approach would be to calculate the threat cost at several locations along an edge while taking into account the length of the edge. For this purpose, the threat cost was calculated at three points: $L_i/6$, $L_i/2$, $5L_i/6$ along each edge, where L_i is the length of edge i. This is shown in Figure 6.8. The threat cost associated with the i^{th} edge is

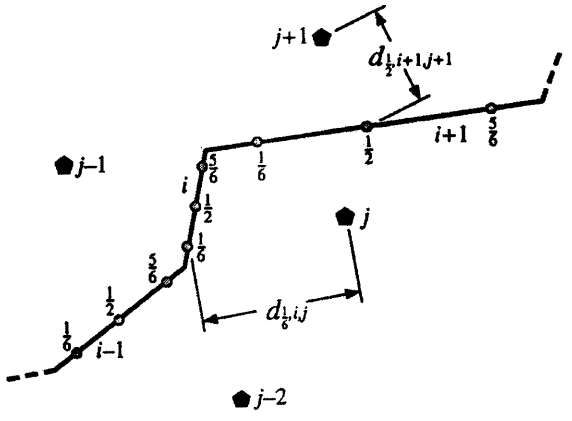


Figure 6.8: Threat cost calculation

given by the expression

$$J_{t,i} = L_i \sum_{j=1}^{N} \left(\frac{1}{d^4_{\frac{1}{6},i,j}} + \frac{1}{d^4_{\frac{1}{2},i,j}} + \frac{1}{d^4_{\frac{5}{6},i,j}} \right) \tag{6.3.1}$$

where N is the number of threats and $d^4_{\frac{1}{6},i,j}$ is the distance from the $1/6^{th}$ point on the i^{th} edge to the j^{th} threat. If the UAVs are flying at constant speed, the fuel required to travel (or time of travel) along one edge of the Voronoi diagram will be proportional to the length of the edge. The fuel cost associated with the i^{th} edge is given by

$$J_{f,i} = L_i \tag{6.3.2}$$

The total cost for travelling along an edge is the weighted sum of the threat and fuel costs

$$J_i = kJ_{t,i} + (1-k)J_{f,i} \qquad 0 \leq k \leq 1 \tag{6.3.3}$$

By selecting different values of k between 0 and 1, we can make a compromise between threat exposure and fuel consumption. A dynamic programming approach or Dijkstra's algorithm can be used to search the graph for an optimal path.

6.3.2 Optimization Algorithm

Once the Voronoi diagram is available. The optimal path can be searched using Dijkstra's algorithm. The algorithm proceeds by assigning labels to each node which may be temporary or permanent. A temporary label can be changed, whereas a permanent one cannot. For example, if node p has a permanent label (q, r), then r is the cost in going from ns to p which is $cost(ns, p)$ and q is the node on the shortest cost path $ns \rightarrow p$. If the label is temporary, then it has the same meaning but it refers only to the shortest path found so far. A shorter path may found later, in which case the label may become permanent. The functional form of the routine used is

$[splen, path] = OptAlg(n, ns, ne, nnz, D, row, col)$

The inputs of the algorithm are:

- n is total number of nodes

- ns is the starting point node number
- ne is the end point node number
- nnz is the number of non-zero elements of the connection matrix
- D is the cost vector of non-zero elements of the connection matrix ordered by increasing row index and increasing column index within each row
- row and col contain the row and column indices respectively of the nonzero elements of the connection matrix

The main steps of the algorithm are:

Step 1 Assign the permanent label $(0, 0)$ to the node ns and temporary labels $(0, \infty)$ to every other node. A zero in the first place indicates that no assignment has been made for the previous node position. Adjust initially the permanent node variable k as $k = ns$.

Step 2 Consider each node y connected to the node k with a temporary label in turn. Let the label at k be (p, q) and at $y(r, s)$. If $q + cost(k, y) < s$, then a new temporary label $(k, q + cost(k, y))$ is assigned to node y. Otherwise no change is made in the label of y. When all nodes y with temporary label adjacent to k have been considered, then move to the next step.

Step 3 From the set of temporary labels, select the one with the smallest second component and declare that label to be permanent. The node it is attached to becomes the new node k. If $k = ne$, go to the next step. Otherwise, go to step 2 unless no new node can be found. So this will be the case when the set of temporary labels is empty but $k \neq ne$, in which case no connected network exists between nodes ns and ne and the algorithm terminates.

Step 4 If the label of ne is (x, z), then z gives the total cost of the optimal path from ns to ne, while x gives the vector of nodes that links back to the previous node on the shortest path.

Dynamic programming-like algorithms can be applied as well. We describe such an algorithm below. The algorithm starts from the destination node P_e and works out the optimal path, with lowest cost, from each other node to P_e. This algorithm is particularly useful for the challenge problem described in the Introduction, because there will be several UAVs possibly starting from different positions and flying towards the same destination.

Similarly, two-component labels are assigned to all nodes. The algorithm takes the following steps.

Step 1 Assign the label $(0, 0)$ to the final node P_e and $(0, \infty)$ to all other nodes. Initialize the level variable $i = 1$, and define $\Phi_i = \{P_e\}$ and Φ_0 an empty set.

Step 2 For each node $p \in \Phi_i - \Phi_{i-1}$, define $\Psi_{i,p} = \{q\}$, such that q is a node directly connected to p. Define $\Psi_i = \bigcup \Psi_{i,p}$.

Step 3 For all $q(r, \beta) \in \Psi_{i,p}$, if $\alpha + cost(p, q) < \beta$, a new label $(p, \alpha + cost(p, q))$ is assigned to the node q; else, if $\beta + cost(q, p) < \alpha$, a new label $(q, \beta + cost(q, p))$ is assigned to the node p; where p has a label (n, α) and $cost(p, q)$ (or $cost(q, p)$) denotes the cost along the edge from p to q (q to p, though the two are of the same value in this problem). Repeat the above for all $\Psi_{i,p}$.

Step 4 Define $\Phi_{i+1} = \Phi_i \bigcup \Psi_i$. If $\Phi_{i+1} = \Phi_i$, the algorithm terminates; otherwise, set $i = i + 1$, goto Step 2.

When the algorithm finishes, the second component of the label of each node gives the optimal total cost to reach P_e from that node. From the first component of the label, a vector of all nodes which form the optimal path can be traced. If the final Φ_i is a strict subset of the node set, it indicates that not all nodes are connected to the final node P_e.

When used in finding an optimal path between a given pair of starting and end points, there is no obvious difference in terms of computational

efficiency between Dijkstra's algorithm and the dynamic programming approach (slightly modified version of above algorithm).

6.4 Way Point Generation using Developed Software

The software which uses the modified Voronoi diagram with a dynamic programming approach can be utilized to find the optimal path. The way point selection procedure is summarized below:

1. For a given set of threat coordinates, the modified Voronoi diagram is constructed, which gives its output in terms of nodes and the cells.

2. For a given pair of starting and end points, the extended Voronoi graph is evaluated.

3. A connection matrix having binary entries is constructed which gives the connection detail of the nodes i.e. which node is connected to which node. If two nodes are connected then the corresponding element in the matrix is one otherwise it is zero.

4. Each line segment of the graph has some associated cost. The cost is calculated using (6.3.1). A cost vector is constructed for the non-zero elements of the connection matrix.

5. The optimization algorithm described in Section 6.3.2 is applied to find out the optimal path and the associated risk.

The good points of the above algorithm are:

- Computationally efficient and the solution is available at once.

- The memory usage with this approach is very small.

- If the vehicle sensor detects some changes in the environment in the form of a popup threat, then a new diagram can be constructed and searched with the same procedure.

- It gives a global solution with respect to the graph.

The drawbacks of this approach are:

- Since the graph is constructed by considering radars of equal powers, the line segments formed are at equal distance from the pairs of radar sites and so the search is limited to these lines only. But in reality the radars may have different powers e.g.short range, medium range and long range. Hence the optimality of this approach is not global with respect to real scenario.

- When the initial and final nodes are connected to the nearest segment, then it may happen that this path could fall into a dangerous area

In order to best tune the path, the Voronoi approach can be combined with a local search technique and will be described in the next section.

6.4.1 Local Optimization

After a flight path has been selected using a Voronoi graph based method, local optimization and grid search techniques can be used to fine tune the path. Two local searches are involved. The first is to perturb the end-point of each sector of the selected path, except of course the last one of which the end-point is the required destination. Secondly, once the starting and end points of a sector have been decided, another local search will be employed along the straight line connecting these two points. The first kind of search is called "end-point" search and the second one "along-path" search.

Various methods can be used in these searches, for instance, methods based on gradients, simplex algorithms, or point-wise search. The important factor to be considered in adopting a search algorithm is the efficiency, due to the real-time environment. Another consideration is that original points and paths are preferred over new ones unless "significantly" lower risk is estimated.

In the end-point search, a search radius has to be decided first. In the example shown in Figure 6.9, the radius is calculated based on the distance between the end-point and its nearest threat and on the length of that line

segment ended at that end point. In the along-path search, the width of the search band should be similarly defined. For the sake of efficiency (e.g. for the calculation of waypoints in 3-dimensions) and consideration of flight constraints, a few more steps have been adopted in the algorithm code which, for example, eliminate intermediate points along a (nearly) straight-line path sector and/or skip over "sharp"turns.

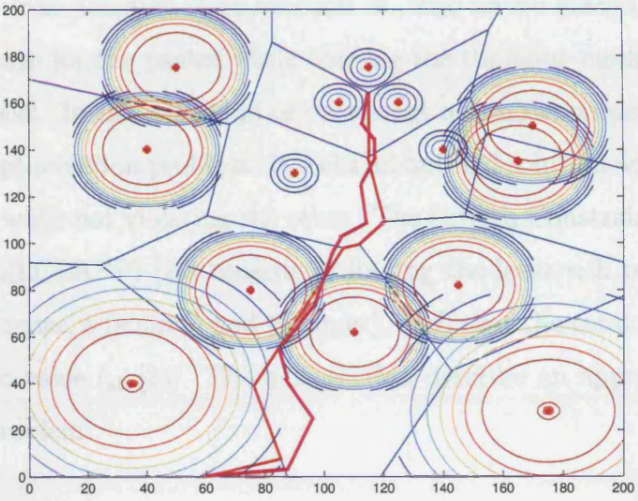


Figure 6.9: A selected Voronoi path (red) and the tuned path (pink) using local optimization

6.5 Constraint Optimization

When the number of threats is small, then the graph produced using the Voronoi method has few arcs. Since the path of the UAV is restricted to these arcs, it is then difficult to implement the path length constraint. One way to do this is to assume some fictitious threats along with real threats. These fictitious threats can be positioned in different ways. Here we will generate these randomly. Since the Voronoi approach is fast, for a reasonable number of fictitious threats, the computation can be run in real time. The path length constraint can be implemented as an upper bound on the selected path. The constraint minimum risk problem is also closely related to many other real life problems. For instance, in a routing query on a road system, people are always

interested in finding the shortest rout from one place to another. If the road system has cost associated with using each specific road, and the query requires the total cost of the selected route to be less than some cost limit, then this is a typical constraint optimization problem, which attempts to find a shortest path while the cost of travelling along the path does not exceed some specified value. Another example would be the packet sending problem in internet traffic. A packet is sent from location A to location B. The server always tries to find the shortest path for the packet while obeying the time-out mechanism of the internet protocol. In fact this kind of constraint optimization problem is a 2-dimensional optimization problem. It seeks an optimal solution with respect to one criterion, while not violating the other. The Length Constraint Least Risk Path Problem(LCLRPP) [24] consists of finding the least risk route between two specified nodes, s (source) and t (target), such that the total weight is less than a specific value l_{ub} [24]. The next section describe an approach to solve this kind of problem.

6.5.1 Dynamic Programming Approach to LCLRPP

In this section, a dynamic programming approach will be developed that will combine the label setting algorithm described in [24] with a preprocessing procedure to simplify the problem. There may be many ways to simplify the LCLRPP. Here the preprocessing idea is based on the fact that finding the least risk path from source node s to every node $i \in \mathcal{V}$ requires no more computational efforts than finding the shortest path from s to the target node t. Both need to perform Dijkstra's algorithm once. Similarly, finding the minimum risk path from every node $i \in \mathcal{V}$ to t requires no more extra cost than finding the shortest minimum risk path from s to t. Hence by applying the least objective (risk or length) calculation 4 times, we get the following for each node $i \in \mathcal{V}$.

$\mathcal{P}^l_{si}$ = least length path from s to i

$\mathcal{P}^l_{it}$ = least length path from i to t

$\mathcal{P}^r_{si}$ = least risk path from s to i

$\mathcal{P}^r_{it}$ = least risk path from i to t

If $\mathcal{P}^l_{st} > l_{ub}$ or simply there exists no path from s to t, then the problem is infeasible. So in order to have a feasible problem we need to introduce fictitious threats again. Further, if the length of the least risk path is less than the upper bound l_{ub}, this is the optimal solution, since it also satisfies the length constraint. Rather than these simple cases, the graph can be simplified by scanning through each node and each arc and dealing those which connect least length paths from s and t greater than the upper bound. This says that the length of the least length path from s to node i plus the length of the least length path from i to t is greater than the length limit, so we can ignore this node from the optimal path search i.e. we could remove this node i and all its incident arcs from the graph G. Also by the same argument as above an arc $< i, j >$ can be removed. Further, let $\mathcal{P}^l_{st}$ be the path that attains the least length, set $M = R(\mathcal{P}^l_{st})$. If a path has a cost greater than M, then this path will not be considered any more in the future optimal path search process, since the least length path $\mathcal{P}^l_{st}$ has a better risk. With this idea if the risk of a path connecting an arc with the least risk path from both ends exceeds M, then we can remove that arc.

After preprocessing, the next stage is to sort out the best path from the graph by using a set of labels on each node. Each label consists of a (risk, length) tuple which corresponds to a different path from the source node s to that node. The risk and length represent the total risk and total length of the path from s to that node respectively. In the set of labels of each node, no two labels have the same risk or the same length. Moreover, if label A has a larger risk than label B and A is stored, then the length in label A must be smaller than the length in label B. These can be defined formally as "dominant-relation"[24] more below.

Definition 1 *Let $(L^{\mathcal{P}}_{si}, R^{\mathcal{P}}_{si})$ and $(L^{\mathcal{Q}}_{si}, R^{\mathcal{Q}}_{si})$ be two different labels on node i, corresponding to two different paths $\mathcal{P}_{si}$ and $\mathcal{Q}_{si}$. We say $(L^{\mathcal{P}}_{si}, R^{\mathcal{P}}_{si})$ dominates $(L^{\mathcal{Q}}_{si}, R^{\mathcal{Q}}_{si})$ if and only if $L^{\mathcal{P}}_{si} \leq L^{\mathcal{Q}}_{si}$ and $R^{\mathcal{P}}_{si} \leq R^{\mathcal{Q}}_{si}$.*

Definition 2 *A label $(L^{\mathcal{P}}_{si}, R^{\mathcal{P}}_{si})$ is said to be efficient if it is not dominated by any other label at node i. In other words, at node i, there exists no path $\mathcal{Q}$ such that $L(\mathcal{Q}) \leq L^{\mathcal{P}}_{si}$ and $R(\mathcal{Q}) \leq R^{\mathcal{P}}_{si}$.*

The sorting procedure will find all the efficient labels for each node. The reason why we also keep track of these paths with larger risk but smaller length is that, any potential optimal path must first satisfy the length limit constraint. It might be the case that at node i, a path with smaller risk and larger length will violate the length limit when the procedure moves onto the next step, in which case the smaller risk does not help us at all. That is why we find all efficient labels instead of the least-risk labels. It starts from the source node, and expands the search process to the neighbouring nodes of the already searched nodes. In particular, the algorithm starts with no labels on any node, except for the label $\{(0,0)\}$ on source node s, then it extends the set of all labels on a node by extending the path along all outgoing arcs of that node. More specifically, when the procedure is operated on path $\mathcal{P}_{si}$ on a label $(L^{\mathcal{P}}_{si}, R^{\mathcal{P}}_{si})$ of node i, it considers each arc $< i,j > \in \delta^+(i)$ where $L^{\mathcal{P}}_{si} + l_{ij} + L_{jt} \leq l_{ub}$. Further, if $(L^{\mathcal{P}}_{si} + l_{ij}, R^{\mathcal{P}}_{si} + r_{ij})$ is not dominated by any existing labels of node j, then the algorithm extends path $\mathcal{P}_{si}$ via arc $< i,j >$ to node j. Next, we will describe the above in a systematic way in.

6.5.2 Length Constraint Optimal Path

Step 1 Let $M = \infty$ and $\mathcal{T}_R = \{(x_i^R, y_i^R, z_i^R, \alpha_i^R) : i = 1, \ldots, n_R\}$ be the set of n_R real threats such that (x_i^R, y_i^R, z_i^R) are position coordinates and α_i is the strength of the i^{th} threat.

Step 2 Define n_F, the number of fictitious threats with zero strength. Generate randomly the n_F fictitious threats $\mathcal{T}_F$ and set $\mathcal{T} = \mathcal{T}_R \cup \mathcal{T}_F$. Construct the complete Voronoi graph for $\mathcal{T}$ threats using the procedure described in section 6.2.3 and find the extended Voronoi graph by augmenting the starting and end points as detailed in section 6.2.4.

Step 3 Assign the risk cost r_{ij} (due to the real threats $\mathcal{T}_R$ only) and the length cost l_{ij} for each of the arc $< i,j > \in \mathcal{A}$ of the graph.

Step 4 Find the least length path from s to t using the algorithm given in section 6.3.2. If the length of this path is greater than the given upper

bound for the path length, then the problem is infeasible; go to Step 2. Otherwise if $R(\mathcal{P}^l_{st}) < M$, then set $M = R(\mathcal{P}^l_{st})$.

Step 5 Using the algorithm given in section 6.3.2, find the minimum risk paths from the source node s to all nodes $j \in \mathcal{V}\backslash\{s\}$. Let $\mathcal{P}^r_{sj}$ be the least risk path from s to j and the R_{sj} is the total risk such that $R_{sj} = R(\mathcal{P}^r_{sj}) = \sum r_{ij}$. If there is no path from source node s to sink node t, then the posed problem is infeasible, STOP. If $L(\mathcal{P}^r_{st}) \leq l_*$, then $\mathcal{P}^r_{st}$ is the optimal path and no need to proceed further, STOP.

Step 6 Using the algorithm given in section 6.3.2, find the minimum risk paths from the sink node t to all nodes $j \in \mathcal{V}\backslash\{s,t\}$. Let $\mathcal{P}^r_{jt}$ be the least risk path from j to t and the R_{jt} is the total risk such that $R_{jt} = R(\mathcal{P}^r_{jt})$.

Step 7 Using the algorithm given in section 6.3.2, find the minimum length paths from the source node s to all nodes $j \in \mathcal{V}\backslash\{s,t\}$. Let $\mathcal{P}^l_{sj}$ be the least length path from s to j and L_{sj} the total length such that $L_{sj} = L(\mathcal{P}^l_{sj}) = \sum l_{ij}$.

Step 8 Using the algorithm given in section 6.3.2, find the minimum length paths from the sink node t to all nodes $j \in \mathcal{V}\backslash\{s,t\}$. Let $\mathcal{P}^l_{jt}$ be the least length path from j to t and L_{jt} the total length such that $L_{jt} = L(\mathcal{P}^l_{jt})$.

Step 9 For all $j \in \mathcal{V}\backslash\{s,t\}$, check if $L_{sj} + L_{jt} > l_{ub}$ or $R_{sj} + R_{jt} \geq M$, then delete node j and all arcs incident on it.

Step 10 Similarly, for all arcs $< i,j > \in \mathcal{A}$, check if $L_{si} + l_{ij} + L_{jt} > l_{ub}$ or $R_{si} + r_{ij} + R_{jt} \geq M$, then delete $< i,j >$. Otherwise if $L(\mathcal{P}^r_{si}) + l_{ij} + L(\mathcal{P}^r_{jt}) \leq l_{ub}$, then $M = R_{si} + r_{ij} + R_{jt}$.

Step 11 If during Steps 9 and 10, the graph changes, then go to Step 5. Otherwise go to next Step.

Step 12 Set the initial labels for all nodes such that $\mathcal{L}_s = \{(0,0)\}$ and $\mathcal{L}_i = \Phi$ for all $i \in \mathcal{V} \setminus \{s\}$.

Step 13 Selection of the label to be extended:

If all labels have been marked, which means all efficient labels have been generated, then go to Step 16. Otherwise choose $i \in \mathcal{V}$ such that there is an unmarked label in $\mathcal{L}_i$ and $L_{si}^{\mathcal{Q}}$ is minimal, where $\mathcal{Q}$ is the path that attains this weight value.

Step 14 Extend label $(L_{si}^{\mathcal{Q}}, R_{si}^{\mathcal{Q}})$:

For all $(i,j) \in \delta^+(i)$ with $L_{si}^{\mathcal{Q}} + l_{ij} + L_{jt} \leq l_{ub}$ and $R_{si}^{\mathcal{Q}} + r_{ij} + R_{jt} < M$, if $(L_{si}^{\mathcal{Q}} + l_{ij}, R_{si}^{\mathcal{Q}} + r_{ij})$ is not dominated by $(L_{sj}^{\mathcal{K}}, R_{sj}^{\mathcal{K}})$ for any existing label at node j through path $\mathcal{K}$, then set $\mathcal{L}_j = \mathcal{L}_j \cup \{(L_{si}^{\mathcal{Q}} + l_{ij}, R_{si}^{\mathcal{Q}} + r_{ij})\}$.

Step 15 Mark label $(L_{si}^{\mathcal{Q}}, R_{si}^{\mathcal{Q}})$ and go to Step 13.

Step 16 Select from the label set $\mathcal{L}_t$ of the destination node, the label with the second component minimal and from this trace back the optimal path.

6.5.3 Time Complexity of the Algorithm

Assuming the graph generated has a feasible solution. Then the time complexity of the above algorithm is equal to the complexity of the Steps from 5 to 16. The time complexity of the preprocessing (Step 5 to Step 11) is $O(|\mathcal{A}| * |\mathcal{V}| * \log|\mathcal{V}|)$. It is not hard to see that the preprocessing will scan through all arcs and nodes and $|\mathcal{A}| = |\mathcal{V}|^2/2$ in the case of a complete graph. So the entire preprocessing will be repeated at most $|\mathcal{A}|$ times. In each step, if a node or an arc is removed, then we need to apply algorithm 1 again to update R_{si}, R_{ti}, L_{si}, L_{it} which takes $O(|\mathcal{V}| * \log|\mathcal{V}|)$. Hence in total, the preprocessing time complexity is $O(|\mathcal{A}| * |\mathcal{V}| * log|\mathcal{V}|)$. Also the time complexity of the LCLRPP procedure (Steps 12 to 16) is $O(|\mathcal{A}| * l_{ub})$, where $|\mathcal{A}|$ is the size of the arc set and l_{ub} is length limit. This is quite a loose bound. The idea is that the algorithm starts extending the labels from the source node. At each node i, the algorithm considers all outgoing arcs, so accumulatively it goes through all arcs. Note that each arc will be used in Step 14 for at most l_{ub} times, since we only consider non-negative weights (lengths) and by

the definition of *dominant* and *efficient* labels, no two labels at one node have the same weight. So if we use a bucket data structure to sort the labels in increasing order of weight, the algorithm will each time pick a label at node i of increasing weights. Hence there shall be at most l_{ub} labels in each node, which means each outgoing arc at node i will be used on Step 14 for at most l_{ub}. Moreover, by using a bucket data structure, the least length label will be always at the top of the bucket, so step 13 is essentially a constant operation. Hence, altogether the time complexity is $O(|\mathcal{A}| * |\mathcal{V}| * log|\mathcal{V}| + |\mathcal{A}| * l_{ub})$.

6.6 Comparison of Different Approaches

In order to compare the different approaches, simulations were performed for one hundred randomly created scenarios in virtual flight environment. The results from these simulations were compared for different parameters to evaluate their effectiveness for real time use.

6.6.1 Flight Environment

- Height was fixed at 2 km.
- Operational area was taken 200×200 km^2.
- Initial and final positions for UAV $(20, 20, 2)$ and $(180, 180, 2)$ respectively.
- Number of SAM sites $5 \rightarrow 10$ (randomly generated).
- Range of missiles: 7 or 27 km (randomly generated).
- Point mass dynamics.
- If risk probability is less than 0.08, it is considered as 0.
- The magnitude of velocity and control inputs are bounded by [100 300] m/s and [−10 10] m/s^2, respectively.

6.6.2 Measured Quantities

The following quantities were measured and compared for each method.

- Success rate
- Maximum time to compute a waypoint
- Peak risk
- Total flight length
- Total flight time
- Average risk

6.6.3 Methods Compared

- M_1: Tuned Voronoi approach
- M_2: Visibility line approach
- M_3: Probabilistic local minimization
- M_4: MILP
- M_5: Bouncing approach [56]

Parameters	Positive	←		→	Negative
Threat Modelling	M_3	M_1	M_5	M_2	M_4
Safety	M_5	M_2	M_4	M_3	M_1
Fast convergence	M_3	M_1	M_2	M_4	M_5
Simplicity	M_5	M_3	M_2	M_1	M_4
Global Optimality	M_2	M_1	M_4	M_3	M_5
Fast computation	M_3	M_1	M_2	M_4	M_5
Flexibility	M_4	M_3	M_5	M_1	M_2

Table 6.1: Summarized comparison results

M_1	M_2	M_3	M_4	M_5
100/100	100/100	100/100	99/100	98/100

Table 6.2: Success rate for different methods

Results for different parameters are summarized in Table 6.1. Among the hundred random simulations, the 23rd scenario was chosen as representative (Figure 6.11) and trajectories for this scenario for the above mentioned methods are shown in Figures 6.12, 6.13, 6.14, 6.15, 6.16. MILP does not use the exact threat model but rather uses the nominal range of the SAM to find the next way point and encloses the nominal range into rectangular region. Due to the highly nonlinear nature of the exact cost function, MILP defines an

auxiliary cost function that uses these rectangular threat zones. The other methods use the exact risk models but in the case of M_1 and M_2, the exact risk function is used to find the optimal path and this is restricted to a search on the Voronoi and Visible edges, respectively. Looking at vehicle safety, the bouncing method finds the safest path by bouncing against the virtual risk boundaries again and again as shown in Figures 6.19 and 6.20. So if there is no gap between the risk boundaries, it will not find a path. Also M_3 and M_4 for most of the cases found the safest path. The Voronoi method sometimes gives a path that passes through highly risky areas as shown in Figures 6.19. Other methods find trajectories at medium risk level. Bouncing is the simplest method that consists of two phases (see Figure 6.10). The first corresponds to a part of the optimization phase in which a UAV tries to find an optimal flight path over a pre-defined period of flight time along which an underlying objective, e.g., distance to a target point is minimized. This can be readily done by solving a mathematical programming problem or performing a simple search. This phase repeats itself and continues until a UAV confronts obstacles or threats where the second phase begins. In second phase, operational area is considered to be consists of cells instead of points. The size of the cell of the cell directly impacts on the closeness to the original threat model and the speed of the convergence to the aimed target. This cell approach also needs the concept of risk probability of a cell and which is calculated as the average of the probabilities of several representative points within that cell. The next direction to proceed is determined by seeking the most desirable cell among neighboring cells around the current position. In this phase,the optimality part regarding minimizing distance to aimed target is ignored and a UAV is asked to follow the surface of the forbidden region with the same direction which was set when the most recent transition from the first to second phase occurs. The transition from the second to first phase occurs when the minimization of the distance to the target point becomes possible again and the current cell is visited the first time for this transition purpose. Results using this approach are as shown in Figures 6.16, 6.17. When the risk threshold was fixed at 0.08, UAV did find the way to direct it towards the target for the 23rd

scenario after many bounces and hence a long time was consumed as shown in Figure 6.16 but when this threshold was relaxed to 0.1, a path becomes available as shown in Figure 6.17. MILP is the most complex method including the binary variables and the complexity of the other methods is evident by the placement order in Table 6.1. If global optimality is concerned, then Visibility and Voronoi are preferable while the others are based on local searches. For real time applications, fast computation is usually required, then probabilistic local minimization has been proved to be the fastest way point generator as shown in Figure 6.18. Flexibility means how easy the methods can incorporate constraints on velocity and acceleration. MILP (by using binary variables) and probabilistic methods can easily accommodate these constraints. Also in the case of the Voronoi approach, path length constraints can be imposed as described in section 6.5.2 but for the Visibility method, this is is still an issue. Probabilistic local minimization seems to be the fastest convergent technique while the flight time for the bouncing algorithm is the highest. Table shows the success percentage of all methods; MILP and bouncing do not give an optimal solution for two scenarios in the specified time.

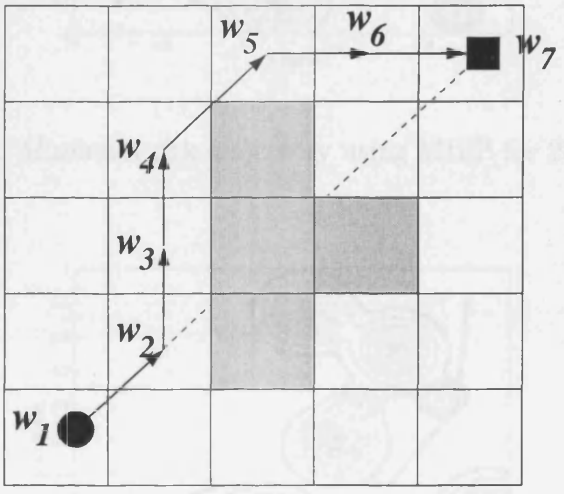


Figure 6.10: Typical waypoints w_i's generated from the bouncing algorithm: w_1 and w_7 are given initial and targets points, respectively. The intervals $[w_1, w_2]$ and $[w_5, w_7]$ correspond to the first phase, and the interval $[w_2, w_5]$ the second phase. The shaded cells denote obstacles.

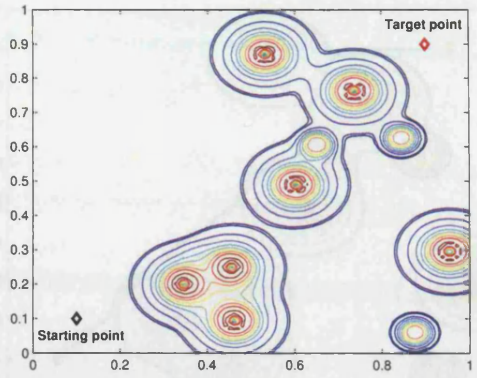


Figure 6.11: 23^{rd} scenario

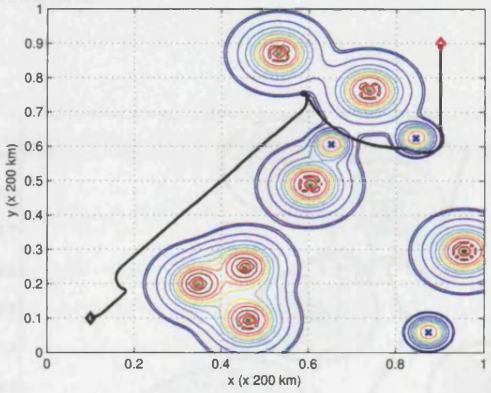


Figure 6.12: Minimum risk trajectory using MILP for 23^{rd} scenario

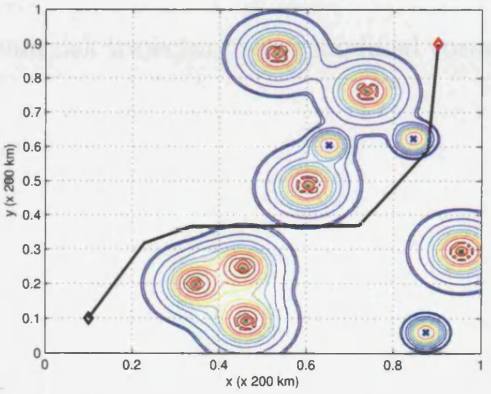


Figure 6.13: Minimum risk trajectory using visibility approach for 23^{rd} scenario

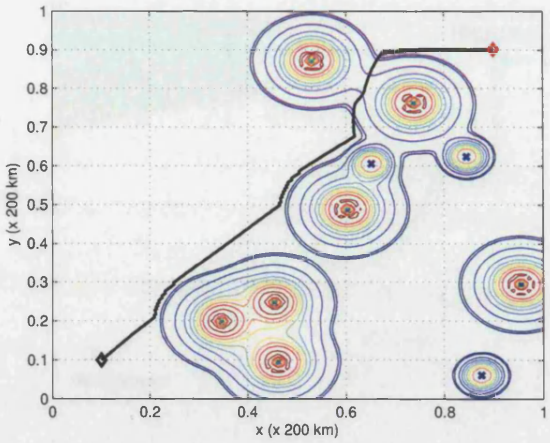


Figure 6.14: Minimum risk trajectory using probabilistic local minimization approach for 23^{rd} scenario

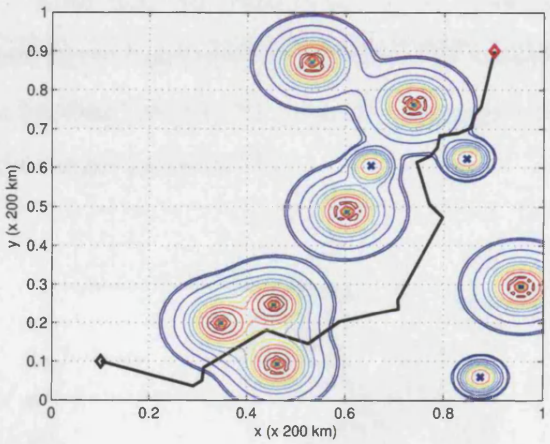


Figure 6.15: Minimum risk trajectory using modified voronoi approach for 23^{rd} scenario

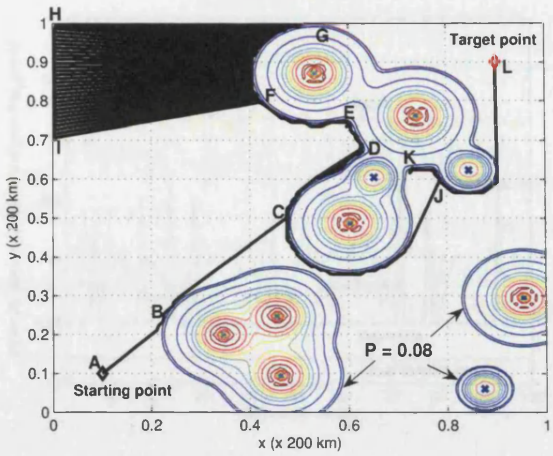


Figure 6.16: Minimum risk trajectory using bouncing technique for 23^{rd} scenario when risk threshold was set at 0.08. The path starts from "A" and ends at "L" via "B", "C", "D", "E", "F", "G", "H", "G", "I", "F", "E", "D", "C", "J", "K" and "J". Note that the path from "G" to "I" involves the numerous transitions between phase I and phase II. As the UAV approaches "F" from "I", it finally turns its heading towards "E", not "G", because this choice minimizes the distance to the target point at "F".

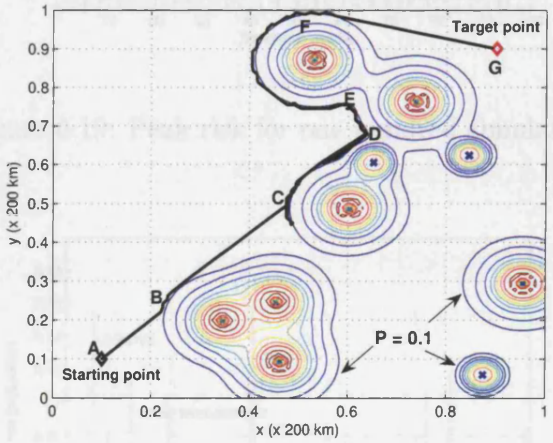


Figure 6.17: Minimum risk trajectory using bouncing technique for 23^{rd} scenario when risk threshold was set at 0.1. As oppose to the previous case, there exist another safe path passing through "F" as a result of increasing risk threshold.

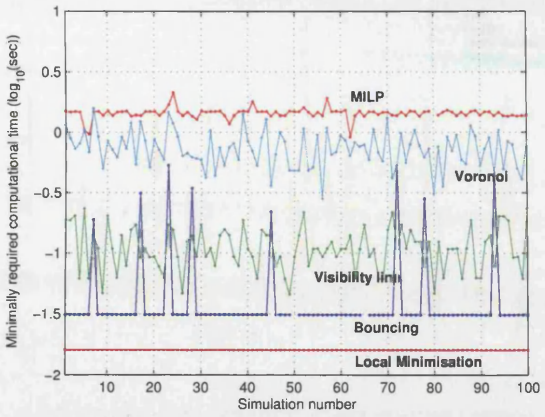


Figure 6.18: Peak computation time for one hundred simulations

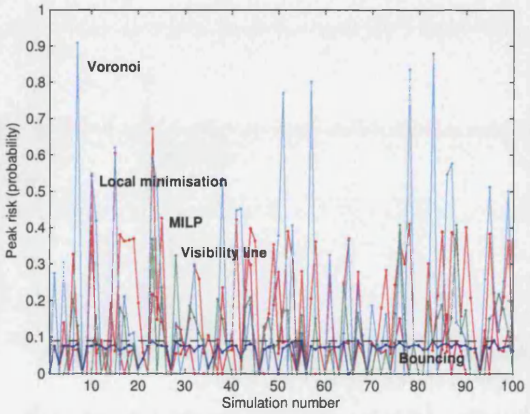


Figure 6.19: Peak risk for one hundred simulations

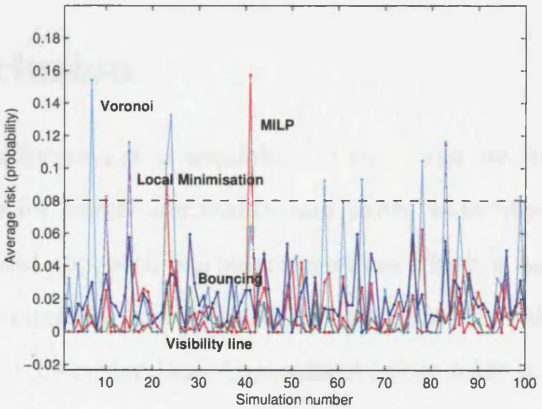


Figure 6.20: Average risk for one hundred simulations

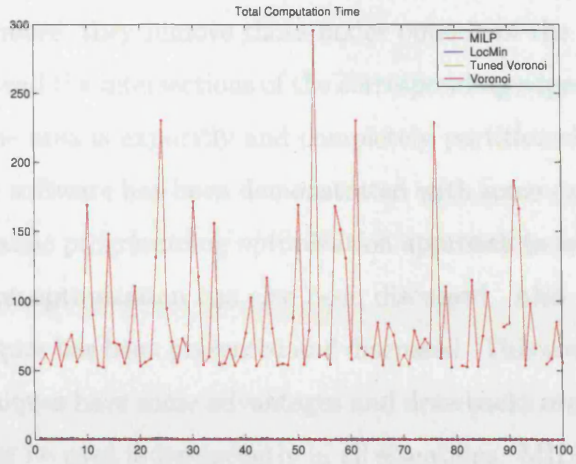


Figure 6.21: Total computation time for one hundred simulations

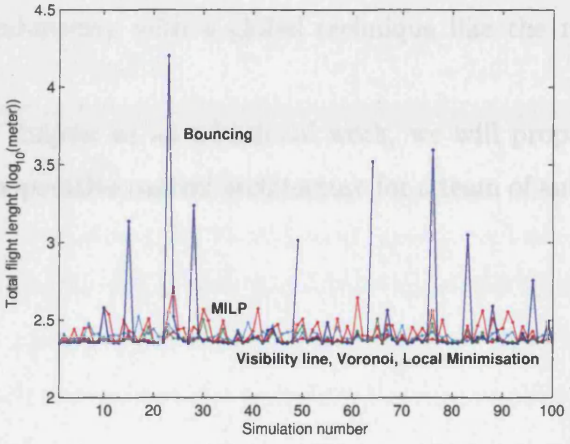


Figure 6.22: Flight time for one hundred simulations

6.7 Conclusion

If some global information is available, then it can be incorporated to the planning process for better optimality and hence sometimes a global planner is needed. A hybrid approach has been presented which is based on the Voronoi graph, local optimization and grid search methods. A software package has been developed at Leicester (see Appendix A) that adds extra subroutines to modify the Voronoi code. These subroutines expand the node list by adding in nodes on borders connecting the infinity nodes and by the inclusion of corner

points. Furthermore, they remove those nodes outside of the considered area and include instead the intersections of the corresponding edges and the border lines. Hence, the area is explicitly and completely partitioned into cells. The efficiency of the software has been demonstrated with some example scenarios that uses a dynamic programming optimization approach to select the optimal path. Constraint optimization has also been discussed. Also a comparison of different techniques has been presented and discussed. This comparison reveals that most techniques have some advantages and drawbacks over the others and hence should not be used independently in all situations. MILP is very flexible in adopting the constraints on velocity and acceleration but for large problems in a centralized mode, it may become intractable for real time use. Therefore, it should be used in a decentralized fashion to reduce the computational effort and also in combination with a global technique like the modified Voronoi graph.

In the next chapter as an additional work, we will propose a systematic decentralized cooperative control architecture for a team of autonomous UAVs.

Chapter 7

A Decentralized Cooperative Control Architecture

7.1 Introduction

7.1.1 Motivation

Advances in computation, sensors, and communication have provided the enabling technologies for achieving cooperative control of multiple vehicle systems. However, research in the cooperative control of UAVs has been limited. Much of the work reported in the literature focuses on close formation flight of multiple vehicle systems. The dynamics of these formations are tightly coupled either by control laws governing the formation behaviour [3] or by a combination of aerodynamics and formation control laws [82, 105]. Because these systems are coupled dynamically, they can be analysed as a single large scale system. Unlike formation flight problems, cooperative path planning and resource allocation problems usually involve UAVs that are physically independent of one another, although they may be coupled dynamically by their cooperative control algorithms. The scaling problem is also apparent in the field of unmanned aerial vehicles. In applications where path planning and coordination of a large fleet of autonomous vehicles is required, centralized solutions quickly become computationally intractable. Moreover, in these applications, the planning problem typically needs to be resolved multiple times, as new in-

formation about the environment is often gathered while the mission unfolds. Thus, a decentralized receding horizon (or model predictive control) planning strategy seems a natural approach to solving the multi-vehicle trajectory generation problem. One such method is proposed in [92], where static obstacles and other moving agents are accounted for by potential functions. Although computationally attractive, the use of potential functions does not guarantee safety due to collision with other UAVs because vehicles are captured using soft constraints in the cost function. In [49], an alternative algorithm based on an iterative bargaining scheme is given. However, as the iteration might converge to an infeasible equilibrium, again only soft safety guarantees exist. Several important UAV cooperative control problems can be formulated as resource allocation problems. This includes target assignment problems [7, 74], cooperative classification problems[18] and cooperative search problems [91]. The majority of the cooperative path planning problems considered in the literature involve timing or sequencing of UAVs for arrival at targets or other specified locations. A cooperative control strategy for UAV rendezvous was presented in [68]. Cooperative path planning is also employed in cooperative search and cooperative classification problems [19]. There is also an extensive literature on multi-robot systems [4, 5, 23, 26, 34, 63, 77].

7.1.2 Problem Description

Here we propose a theoretical cooperative control architecture for a team of N heterogeneous UAVs that are initially engaged on a mission through known target locations in the presence of dynamic threats. Each UAV plans its own trajectory using a receding horizon strategy based on mixed integer linear programming (MILP). Several different types of targets may be considered. M target locations are suspected a priori with a certain probability, while the others are initially unknown. The problem is decomposed into the following subproblems:

- cooperative target assignment
- path planning

- feasible trajectory generation
- trajectory following

During the mission, at each target location, the UAVs perform Confirm, Attack and Loss Assessment tasks . Other target and threat locations are detected gradually during flight towards a known target, while the tasks are determined in real-time by the actions of all UAVs and their results (e.g., sensor readings), which makes the task dynamics stochastic. The tasks must, therefore, be allocated to UAVs in real-time as they arise. Each class of UAVs has its own sensing and attack capabilities with respect to the different target types, so the need for appropriate and efficient assignment is paramount. This produces a simple, flexible, scalable and inherently decentralizable method for task allocation. Every UAV while on its way also monitors the changes in the environment and accommodates this information in planning its path. We incorporate the effect of various decision parameters, target distributions, and the UAV team characteristics in our approach.

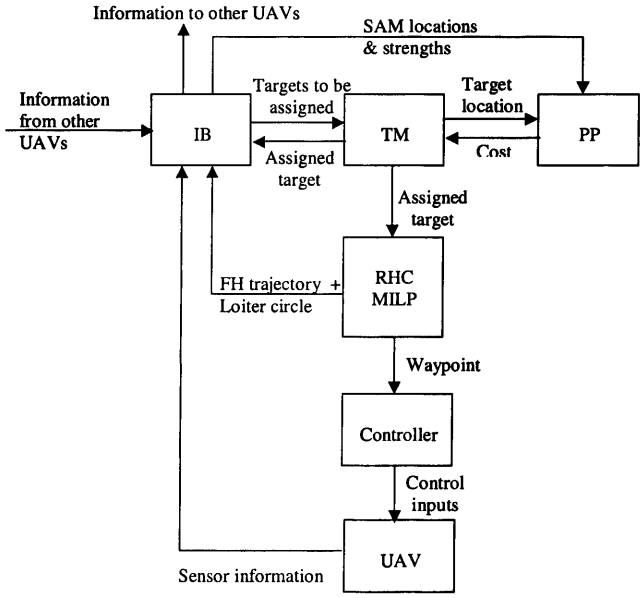


Figure 7.1: Cooperative control architecture

7.2 Proposed Architecture

Figure 7.1 shows the proposed architecture. At the lowest level of the architecture is the physical UAV. The target manager (TM), path planner (PP), and information base (IB) work together to assign a target for the UAV. Sensor measurements from the current planning and other UAVs go to the information base, which also broadcasts its own data to other UAVs. The TM receives from IB the available targets to be assigned and sends target locations one by one to the PP. In turn PP sends TM the approximate cost of achieving the target. The path planner receives location and strength of the SAM sites from IB and make use of the modified Voronoi diagram with fictitious threats to find out the minimum risk path. The optimum path is selected using the dynamic programming approach described in Section 6.3.2, which make use of the cost function given in equation (6.3.3). This cost function takes into account both risk and distance to the target and the compromise between these two can be adjusted by the parameter k which varies from 0 to 1. After the target has been selected, the next stage is the feasible trajectory generation parameterized with time. By feasible, we mean that in the absence of disturbances and modelling errors, an input trajectory causes the UAV to fly the trajectory without violating its velocity and heading rate constraints. We assume that each UAV is equipped with a trajectory tracking controller. This is accomplished using a receding horizon control (RHC) strategy employing a mixed integer linear programming (MILP) technique that finds a trajectory over a finite horizon ending in a loiter pattern. The trajectory generation process makes use of soft dynamic threat zone modelling developed in Section 4.12 but with hard constraints for collision avoidance. The trajectory is calculated in a sequential order after target assignment and is detailed in section 7.4. The trajectory generator outputs a flag that indicates when a new waypoint path is needed. Several events may necessitate a new path. First, the UAV may successfully complete its path. Second, a pop-up threat may be detected in the environment. Third, because of disturbances, the tracking error may become unacceptably large. The IB shown in Figure 7.1 facilitates communication be-

tween different UAVs. Each UAV implements a separate target manager, path planner, trajectory generator and tracking controller. Therefore, the decisions reached by these functional blocks must be synchronized among the different UAVs. The primary role of the IB is to ensure synchronization. The design of communication managers for multiple cooperating autonomous vehicles has been discussed in [38] for UAVs, and [96] for autonomous underwater vehicles.

7.3 UAV State

The state, $S_{i,t} = [s'_{i,t}\ \pi'_{i,t}\ \pi_{i,t}]'$, of the ith UAV, at time t has two parts:

- A physical state $s_{i,t} = [p'_{i,t}\ v'_{i,t}]'$: including information on its
  1. position $p_{i,t}$ at time t
  2. speed $v_{i,t}$ at time t
  3. heading $\psi_{i,t}$ at time t
- A functional state $\pi_{i,t} = [id_{i,t}\ \kappa'_{i,t}\ \partial_{i,t}\ \rho_{i,t}\ J_{i,t}]'$: including the
  1. identity $id_{i,t}$ of the task if any
  2. location of the specific task (if any) $\kappa_{i,t}$ to which the UAV is committed or has bid for
  3. nature of the task $\partial_{i,t}$ which can take values from the set $\{confirm, attack, loss\ assessment\}$
  4. the corresponding commitment status $\rho_{i,t}$ which can take values from the set $\{open, competing, comitted\}$ indicating whether the UAV has no commitment (open), has bid on a task or been associated with one (competing), or is assigned to a task and, possibly, is performing it (committed)
  5. the UAV's expected cost for performing this task $J_{i,t}$

  The functional state of an open UAV has null values in its other fields. The ignore tasks require no commitment, and correspond to an open functional state.

7.4 Sequential Trajectory Planning

Each UAV plans its trajectory individually using a receding horizon strategy based on mixed integer linear programming (MILP). A first order discrete zero order hold system is used to capture the dynamics and kinematics of the vehicle (see Section 4.2). At each time step, a dynamically feasible trajectory for each aircraft that terminates in a loiter pattern is calculated by taking into account: risk due to SAM sites and also risk due to collision with vehicles. Dynamical soft constraints are used to model radar zones while the loiter pattern guarantees safety due to collision with vehicles. Conflicts between multiple UAVs are resolved in a sequential, decentralized fashion, in which each UAV takes into account the latest trajectory and loiter pattern of the other UAVs. Besides maintaining feasibility, if the problem is too complex to be solved within the time constraints of a real- time system, this approach also provides an a priori safe rescue solution consisting of the previous trajectories and individual loiter patterns.

Each UAV individually computes its trajectory towards a destination waypoint, accounting for the intentions of the other UAVs and also minimizing risk due to SAM units. Since information on the latter is gathered online and changes as they update their own trajectories, each UAV adopts a receding horizon planning strategy: a new segment of the total path towards the destination is computed at each time step by solving an optimization problem over a limited horizon of length T. The cost function to be minimized is a measure of time or fuel and observability or risk. The solution to the optimization problem provides the trajectory points and corresponding input commands to the aircraft for the next T time steps. However, only the first of these input commands is actually implemented, and the process is repeated at the next time step. As such, new information about the state and actions of the (other) UAVs can be accounted for at each iteration. Assume that $s_{i,f} = [p'_{i,f}\ v'_{i,f}]'$ is the destination state vector of the ith UAV. It consists of a final position $p_{i,f}$ and a corresponding speed vector $v_{i,f}$ with respect to an inertial coordinate frame. This waypoint is assigned by the target manager to

the UAV. Given the state $s_{i,t}$ at a certain time step t, the trajectory resulting from solving the path planning problem towards $x_{i,f}$ consists of a sequence of $(T+1)$ states $s_{i,t+k},\ k = 0, \ldots, T$, and a corresponding sequence T inputs $u_{i,t+l},\ l = 0, \ldots, (T-1)$. The plan starting at time step t must be computed during time step t - 1, i.e. when the UAV is on its way to $x_{i,t}$. The latter state is part of the previous plan, which we assume to be accurately tracked. As such, the UAV will be in the predicted state $x_{i,t}$ when the next plan is executed. It implies that each UAV can reliably assume that all other UAVs are exactly following their trajectories as planned. Including robustness to uncertainties in the latter is a topic of future research. Safety of the vehicle is concerned with two things: collisions with other vehicles and danger from SAM units. Since a SAM unit may have variable range and there is no physical boundary, safety due to this means following a trajectory that is of minimum risk. The prevention of collisions with other vehicles is most important. Hence a UAV will be in a safe state, if from that state, there exists a known dynamically feasible trajectory to a sequence of states ending in a circular loiter pattern that is collision free [90], has minimum risk and in which the vehicle can remain for a long time to avoid collision with other vehicles.

Conflict Area Let $\mathcal{A}_{i,t}$ be $\subset \Re^2$ the subset of the inertial space which represents the area in which the dynamically feasible trajectory lies that start at $s_{i,t}$ and ends in a feasible loiter pattern where $\mathcal{T}_{i,t} \cup \mathcal{C}_{i,t+T} \subset \mathcal{A}_{i,t}$.

Conflict Set The UAV i is involved in a conflict with UAV $j \neq i$ at time step t, if $\mathcal{A}_{i,t} \cap \mathcal{A}_{j,t} \neq \emptyset$ and the set $\mathcal{B}_{i,t}$ of all such UAVs is called a conflict set.

Plan The sequence of trajectory points starting at time step t and the coordinates of the rectangle that is aligned with the inertial coordinate frame and surrounds the loiter pattern $\mathcal{C}_{i,t+T}$ is called the plan of the aircraft i and is denoted by $\mathcal{D}_{i,t}$.

Planning Order Each UAV is given the same upper bound Δt on the time interval during which the UAV must solve its optimization problem and

the central information base will determine the orders $\mathcal{O}_t$ of the UAVs in which the conflicts are resolved.

The conflict area $\mathcal{A}_i$ of the UAV i can be taken as a circle whose radius is the sum of the maximum distance that can be travelled over the length of the planning horizon including a margin for parking circle and is $\sqrt{(T\Delta t v_{max})^2 + 4r_{max}^2}$, where r_{max} is the maximum achievable radius of the parking circle. Initially at $t = 0$, we assume $\mathcal{A}_{1,0} \cap \mathcal{A}_{2,0}, \ldots, \cap \mathcal{A}_{1,0} = \phi$. Now the path planning algorithm given in can be described as:

Step 1 Start at time t.

Step 2 Send the next predicted position $p_{i,t+1}$ to the central information base and in response receive the conflict set $\mathcal{B}_{i,t+1}$, the planning order and planning starting time $t_{ord(i),s}$

Step 3 If conflict set $\mathcal{B}_{i,t+1}$ is non-empty, then broadcast the current plan $\mathcal{D}_{i,t}$ to all UAVs in $\mathcal{B}_{i,t+1}$ and go to Step 5 else go to next Step.

Step 4 Solve the receding horizon optimization problem for the i^{th} UAV. If an acceptable solution is found within time δt, let the new plan $\mathcal{D}_{i,t+1} = \mathcal{T}_{i,t+1} \cup \mathcal{C}_{i,T+t+1}$ else let $\mathcal{D}_{i,t+1} = \mathcal{D}_{i,t} \setminus p_{i,t}$. Go to last Step.

Step 5 At time $t_{ord(i),s}$, solve the optimization problem by taking into account the latest plans for all UAVs j whose order in the conflict set is less than the order of the i^{th}UAV and taking the earlier plans for all UAVs k whose order is greater than the order of the i^{th} UAV.

Step 6 If an acceptable solution is found at time $t_{ord(i)} + \delta t$, let the new plan be $\mathcal{D}_{i,t+1} = \mathcal{T}_{i,t+1} \cup \mathcal{C}_{i,T+t+1}$ else let $\mathcal{D}_{i,t+1} = \mathcal{D}_{i,t} \setminus p_{i,t}$. Go to next Step.

Step 7 During $[t_{ord(i)} + \delta t, t_{ord(i)} + \delta t + \delta t_{comm}]$ broadcast $\mathcal{D}_{i,t+1}$ to all UAVs k whose order is greater than the i^{th} UAV.

Step 8 End by time $t + 1$ and repeat.

The trajectory at each iteration for each UAV is constrained to terminate in a loiter pattern (parking circle) by taking into account dynamic soft constraint

for radar zone modelling developed in Section 4.12 and hard constraints for collision avoidance. Further the hard constraints for collision avoidance consists of two types of constraints: rectangular constraints for each finite horizon trajectory point (Type 1 collision avoidance constraints) and rectangular constraints surrounded by the corresponding parking circles (Type 2 collision avoidance constraints).

7.4.1 Type 1 collision avoidance constraints

Each trajectory point $p_{j,k}$ of UAV $j \in \mathcal{B}_{i,t}$ is considered an obstacle that is present at time step k in the planning horizon, if that point lies in the conflict area $\mathcal{A}_i$ of the planning UAV i. The time steps for which $p_{j,k} \in \mathcal{A}_i$ are $\bar{\mathcal{T}}_j \subseteq \bar{\mathcal{T}} = [0, \ldots, T]$. These points are considered square obstacles of dimension $d_s = 2(max(v_{max}\Delta t, d_{safe}) + v_{max}\Delta t)$, where d_{safe} is the required safety distance around each UAV. The lower left corner of the waypoint obstacle is then given by $(x^{type1}_{min,jk}, y^{type1}_{min,jk}) = (x_{jk} - \frac{d_s}{2}, y_{jk} - \frac{d_s}{2})$ and the upper right corner by $(x^{type1}_{max,jk}, y^{type1}_{max,jk}) = (x_{jk} + \frac{d_s}{2}, y_{jk} + \frac{d_s}{2})$. The constraint for this type can be formulated as described in equation (4.4.10) of Section 4.4.

7.4.2 Type 2 collision avoidance constraints

Again this is a hard constraint formed by enclosing the parking circles within a tight square of side $2 * R$, where R is the radius of the parking circle which is given by $R = \frac{v_T^2}{F_c}$. As can be seen in Figure 7.2, this radius is decided by the magnitudes of the terminal velocity v_T and the maximum centripetal force F_c for a unit mass UAV.

The position vector of a point at an angle θ on the right circle can be written as

$$p_\theta^{right} = p_c^{right} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} (p_T - p_c^{right}) \tag{7.4.1}$$

where p_c^{right} is the centre of the right circle. Also if $\vec{v}_T^{\perp} = [-v_{y_T}\ v_{x_T}]^T$ is the

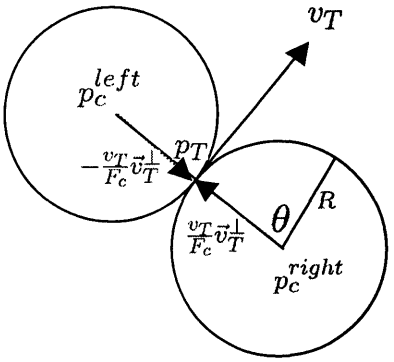


Figure 7.2: Parking or loiter circles for the UAV

orthogonal complement of $\vec{v}_T = [v_{x_T}\ v_{y_T}]^T$, then

$$\begin{aligned} p_T - p_c^{right} &= \frac{v_T^2}{F_c}\hat{v}_T^{\perp} \\ &= \frac{v_T}{F_c}\vec{v}_T^{\perp} \end{aligned} \tag{7.4.2}$$

Using this in (7.4.1), we have

$$p_\theta^{right} = p_T - \frac{v_T}{F_c}\vec{v}_T^{\perp} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \frac{v_T}{F_c}\vec{v}_T^{\perp} \tag{7.4.3}$$

or

$$\begin{bmatrix} x_\theta^{right} \\ y_\theta^{right} \end{bmatrix} = \begin{bmatrix} x_T \\ y_T \end{bmatrix} - \frac{v_T}{F_c}\begin{bmatrix} -\dot{y}_T \\ \dot{x}_T \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \frac{v_T}{F_c}\begin{bmatrix} -\dot{y}_T \\ \dot{x}_T \end{bmatrix} \tag{7.4.4}$$

$$x_\theta^{right} = x_T - \frac{v_T}{F_c}\dot{y}_T(\cos\theta - 1) - \frac{v_T}{F_c}\dot{x}_T\sin\theta \tag{7.4.5}$$

$$y_\theta^{right} = y_T + \frac{v_T}{F_c}\dot{x}_T(\cos\theta - 1) - \frac{v_T}{F_c}\dot{y}_T\sin\theta \tag{7.4.6}$$

Similarly for a left parking circle

$$x_\theta^{left} = x_T + \frac{v_T}{F_c}\dot{y}_T(\cos\theta - 1) + \frac{v_T}{F_c}\dot{x}_T\sin\theta \tag{7.4.7}$$

$$y_\theta^{left} = y_T - \frac{v_T}{F_c}\dot{x}_T(\cos\theta - 1) + \frac{v_T}{F_c}\dot{y}_T\sin\theta \tag{7.4.8}$$

Let the parking circle of each UAV j having conflict with the planning UAV i be constrained within a square with lower left corner $(x_{min,j}^{type2}, y_{min,j}^{type2})$ and upper right corner $(x_{max,j}^{type2}, y_{max,j}^{type2})$. Select equally spaced discrete points N_P on the parking circle of the UAV i such that θ_s is the sampling angle. The above

dimensions include continuous segments of the parking circle of the UAV i between its discrete sample points. So the constraint to avoid parking circles of UAV j involved in conflict with the planning UAV i can be modelled using equation (4.4.10) by the introduction of binary variable b^{LR} which select either the left or right circle, $\forall j$ and $\forall k \in [1, \ldots, N_P]$ such that

$$\begin{aligned}
x_{i_T} - \frac{v_T}{F_c}\dot{y}_{i_T}(\cos k\theta_s - 1) - \frac{v_T}{F_c}\dot{x}_{i_T}\sin k\theta_s &\leq x^{type2}_{min,j} + \Omega^p b^p_{jk1} + \Omega^p b^{LR} \\
-x_{i_T} + \frac{v_T}{F_c}\dot{y}_{i_T}(\cos k\theta_s - 1) + \frac{v_T}{F_c}\dot{x}_{i_T}\sin k\theta_s &\leq -x^{type2}_{max,j} + \Omega^p b^p_{jk2} + \Omega^p b^{LR} \\
y_{i_T} + \frac{v_T}{F_c}\dot{x}_{i_T}(\cos k\theta_s - 1) - \frac{v_T}{F_c}\dot{y}_{i_T}\sin k\theta_s &\leq y^{type2}_{min,j} + \Omega^p b^p_{jk3} + \Omega^p b^{LR} \\
-y_{i_T} - \frac{v_T}{F_c}\dot{x}_T(\cos k\theta_s - 1) + \frac{v_T}{F_c}\dot{y}_{i_T}\sin k\theta_s &\leq -y^{type2}_{max,j} + \Omega^p b^p_{jk4} + \Omega^p b^{LR}
\end{aligned} \tag{7.4.9}$$

$$\begin{aligned}
x_{i_T} + \frac{v_T}{F_c}\dot{y}_{i_T}(\cos k\theta_s - 1) + \frac{v_T}{F_c}\dot{x}_{i_T}\sin k\theta_s &\leq x^{type2}_{min,j} + \Omega^p b^p_{jk1} + \Omega^p(1 - b^{LR}) \\
-x_{i_T} - \frac{v_T}{F_c}\dot{y}_{i_T}(\cos k\theta_s - 1) - \frac{v_T}{F_c}\dot{x}_{i_T}\sin k\theta_s &\leq -x^{type2}_{max,j} + \Omega^p b^p_{jk2} + \Omega^p(1 - b^{LR}) \\
y_{i_T} - \frac{v_T}{F_c}\dot{x}_{i_T}(\cos k\theta_s - 1) + \frac{v_T}{F_c}\dot{y}_{i_T}\sin k\theta_s &\leq y^{type2}_{min,j} + \Omega^p b^p_{jk3} + \Omega^p(1 - b^{LR}) \\
-y_{i_T} + \frac{v_T}{F_c}\dot{x}_T(\cos k\theta_s - 1) - \frac{v_T}{F_c}\dot{y}_{i_T}\sin k\theta_s &\leq -y^{type2}_{max,j} + \Omega^p b^p_{jk4} + \Omega^p(1 - b^{LR})
\end{aligned} \tag{7.4.10}$$

7.5 Map Dynamics

As u_i moves in the environment towards the assigned target, it performs an action, $a_{i,t}$. A canonical action set is denoted by D, which includes sensor readings, firing of munitions and null actions. Since the UAVs have limited weapons, so the firing of munitions is done on the high valued targets and is rarely performed on threats. The threats are avoided by selecting a minimum risk path based on the threat probability map. If the action is a sensor reading, it returns an observation value, $b_{i,t}$, which is a stochastic quantity that is used to update the threat map $\mathcal{M}^{thr}(x, y, t)$, the target occupancy map $\mathcal{M}^{tar}(x, y, t)$ and the task status map $\mathcal{M}^{tas}(x_{\tau_j}, y_{\tau_j}, t)$ $\forall$ current targets $\tau_j, j = 1, \ldots, M$ located at (x_{τ_j}, y_{τ_j}). If multiple UAVs collect observations at the same time, updates due to their actions are applied sequentially. This determines the up-

dates of the different maps at (x, y) through a possibly stochastic map update function.

7.5.1 Derivation of the Map Update Equation

Consider the case where a UAV takes a measurement in cell (x, y) at time t. Let A be the event that a target is located in cell (x,y) and b_t is the binary sensor reading taken by the UAV, where $b_t = 1$ indicates target detection and $b_t = 0$ for no detection. Further, B_{t-1} is the vector of all sensor readings for cell (x, y) by all UAVs taken up to time $t-1$, that is before time t.

Based on the above definitions, $P(A|B_{t-1})$ is the probability of target existence at time $t-1$ and $P(A|B_{t-1}, b_t)$ is the updated probability after obtaining the new reading b_t. So

$$P(t-1) = P(A|B_{t-1}) \tag{7.5.11}$$

$$P(t) = P(A|B_{t-1}, b_t) \tag{7.5.12}$$

We assume that the sensor measurements in any cell are conditionally independent given the state of the cell, i.e.

$$P(b_1, b_2, \ldots, b_n|A) = \prod_{i=1}^{n} P(b_i|A) \tag{7.5.13}$$

Now

$$P(A|B_{t-1}, b_t) = \frac{P(B_{t-1}, b_t|A)P(A)}{P(B_{t-1}, b_t)} \tag{7.5.14}$$

$$P(\bar{A}|B_{t-1}, b_t) = \frac{P(B_{t-1}, b_t|\bar{A})P(\bar{A})}{P(B_{t-1}, b_t)} \tag{7.5.15}$$

Dividing equation (7.5.14) by (7.5.15), we can write

$$\begin{aligned}
\frac{P(A|B_{t-1}, b_t)}{P(\bar{A}|B_{t-1}, b_t)} &= \frac{P(B_{t-1}, b_t|A)P(A)}{P(B_{t-1}, b_t|\bar{A})P(\bar{A})} \\
&= \frac{P(B_{t-1}|A)P(b_t|A)P(A)}{P(B_{t-1}|\bar{A})P(b_t|\bar{A})P(\bar{A})} \\
&= \frac{\frac{P(A|B_{t-1})P(B_{t-1})}{P(A)}P(b_t|A)P(A)}{\frac{P(\bar{A}|B_{t-1})P(B_{t-1})}{P(\bar{A})}P(b_t|\bar{A})P(\bar{A})} \\
&= \frac{P(A|B_{t-1})P(b_t|A)}{P(\bar{A}|B_{t-1})P(b_t|\bar{A})}
\end{aligned} \tag{7.5.16}$$

Taking $\alpha = \frac{P(b_t|A)}{P(b_t|\bar{A})}$ and solving (7.5.16) for $P(A|B_{t-1}, b_t)$

$$\begin{aligned} P(A|B_{t-1}, b_t) &= \alpha \frac{P(A|B_{t-1})}{P(\bar{A}|B_{t-1})} P(\bar{A}|B_{t-1}, B_t) \\ &= \alpha \frac{P(A|B_{t-1})}{1 - P(A|B_{t-1})} [1 - P(A|B_{t-1}, B_t)] \end{aligned} \quad (7.5.17)$$

or

$$\left[1 + \alpha \frac{P(A|B_{t-1})}{1 - P(A|B_{t-1})}\right] P(A|B_{t-1}, b_t) = \alpha \frac{P(A|B_{t-1})}{1 - P(A|B_{t-1})} \quad (7.5.18)$$

$$P(A|B_{t-1}, b_t) = \alpha \frac{P(A|B_{t-1})}{1 + (\alpha - 1)P(A|B_{t-1})} \quad (7.5.19)$$

Hence, when the UAVs search sensors report a target present in cell (x, y), i.e. $b_i(x, y, t) = 1$, the update is

$$P_{(x,y)}(t+1) = \alpha \frac{P_{(x,y)}(t)}{1 + (\alpha - 1)P_{x,y}(t)} \quad (7.5.20)$$

When the UAVs search sensors report that there is no target in the cell at (x, y), i.e., $b_i(x, y, t) = 0$, then the update will be according to the equation

$$P_{(x,y)}(t+1) = \frac{1 - P_{(x,y)}(t)}{1 + (\alpha - 1)P_{(x,y)}(t)} \quad (7.5.21)$$

7.6 Tasks for UAVs

We categorize the tasks for UAVs into two main classes when it starts moving towards the assigned target.

7.6.1 Primary Task

This task is performed by each UAV at all time.

Detection: UAV, u_i, makes a sensor reading $b_i(x, y, t) = 1$ if the sensor detects a target or threat and $b_i(x, y, t) = 0$ if it does not. Each UAV is assumed to be equipped with proper identification algorithms that help in differentiating between a target and threat. The sensor is assumed to be imperfect and the detection accuracy of the sensor is characterized by

$$\alpha = \frac{P(b_t|A)}{P(b_t|\bar{A})}$$

where A is the event that a target or threat is actually located in the cell being scanned. The threat map $\mathcal{M}^{thr}(x,y,t)$ and target map $\mathcal{M}^{tar}(x,y,t)$ are updated based on a Bayesian formulation

$$P_{(x,y)}(t+1) = [1-b_i(x,y,t)]\frac{1-P_{(x,y)}(t)}{1+(\alpha-1)P_{(x,y)}(t)} + b_i(x,y,t)\frac{\alpha P_{(x,y)}(t)}{1+(\alpha-1)P_{(x,y)}(t)} \tag{7.6.22}$$

A cell with the search status transitions to ignore if the probability falls below the resolution threshold, p_r, and to confirm if the probability exceeds the suspicion threshold, p_{tar} for target detection.

7.6.2 Secondary Tasks

These tasks are only performed at the target locations.

Confirm: The cuing of a confirm task at cell (x,y) indicates that a UAV with the appropriate sensors should move towards the cell and scan it. All cells with a priori suspected targets are initialized with the confirm task and given a target occupancy probability of p_s. The confirm task is functionally identical to detection, and the probabilistic map update function is the same as given in equation (7.6.22). However, unlike detection, it is assignable to UAVs with the appropriate expertise. The sensors used may also be different in the two cases. The cell transitions to search if its detection probability falls below p_s (as a result of failure to confirm suspicions), and to attack if it exceeds the certainty threshold, p_{att}. UAVs don't perform confirmation tasks for threats and only update the threat map based on sensor data. An object is considered a threat if its threat probability exceeds the threshold p_{thr}.

Attack: The attack status indicates that an appropriately armed UAV should proceed to the location and attack the target with the correct munition. In this case the updated probability for target existence $P^{tar}_{(x,y)}(t+1)$ is the same as the probability of target existence at time step $t+1$ given that the target has been attacked. And which is turn equals the probability that the target is present at (x,y) at time step t and not destroyed given that the target has been attacked. This can be calculated from the product of two probabilities: probability of target presence at (x,y) at time step t and the probability that

target not destroyed given that the target has been attacked. The attacking UAV then changes the probability for target existence $P^{tar}_{(x,y)}(t+1)$ at that location to:

$$\begin{aligned} P^{tar}_{(x,y)}(t+1) &= P^{tar}_{(x,y)}(t)[1 - prob(\text{target destroyed} \mid \text{target attacked}) \\ &= P^{tar}_{(x,y)}(t)(1-p_D)] \end{aligned} \tag{7.6.23}$$

where $0 \leq p_D \leq 1$ is the probability that the target is destroyed in the attack. Different types of UAVs can have different values of p_D for different target types. If the updated $P^{tar}_{(x,y)}(t+1)$ becomes less than the exit threshold, p_E, the cell transitions to status LA.

Loss assessment: The purpose in the loss assessment (LA) task is to verify that the PTaE has indeed fallen below p_E. Like detection and confirm, this is a purely observational task, and uses the same update equation (7.6.22). If the result of the update produces $P^{tar}_{(x,y)}(t+1) \geq p_E$, the cell transitions back to attack provided it still has weapons (otherwise it returns to base). If $P(x,y,t+1) < p_E$, the cell position is assumed to be target free and the UAV heads towards the next assigned target.

Ignore: This status applies to cells that are known a priori to harbour no targets or where the absence of targets has been confirmed. Figure 1 shows the transitions between states using an automaton formulation.

7.7 Target Assignment

Suppose that there are N UAVs u_i, $i = 1, \ldots, N$, M targets α_j, $j = 1, \ldots, M$ and L threats β_k, $k = 1, \ldots, L$. These targets and threats are known initially while more can be discovered while flying towards a known target. The new target or threat is discovered within the limited sensor range. The aim is to destroy as many targets as possible while moving on minimum risk paths to avoid threats. The UAVs will be gathering information online while heading towards a pre assigned target and to perform confirm, attack and loss assessment tasks on the target location such that the overall group cost is optimized. We consider UAVs drawn from two classes: Target recognition (TR) UAVs;

and attack (A) UAVs. The TR UAVs are best suited for the confirm, attack and loss assessment tasks, while A UAVs are needed for attack. All UAVs are assumed to have equal capability to detect a target or a SAM site. UAVs of any class X have a class-specific expertise vector, $[\xi_j^X]$, with respect to the four tasks, $T_j, j = 1, \ldots, 4$, in the task set. The expertise of class X UAV u_i for task T_j is, therefore, denoted by ξ_{ij}^X. All ξ_{ij}^X are between 0 and 1. In keeping with the capability designations, we set: $\xi_1^{TR} = \xi_1^A, \xi_2^{TR} > \xi_2^A, \xi_3^{TR} < \xi_3^A, \xi_4^{TR} > \xi_4^A$. For this type of decision problem, the standard approach is to create an objective function that encodes the desired objectives of the decision problem. An optimal solution is one that minimizes this objective function. The complexity of the problem for finding an optimal solution increases with the number of targets. For agents with targets, we must search through possible assignments. The target and task assignment process is done initially at time $t = 0$ and is repeated again whenever a new threat or target is discovered. Whenever the target assignment becomes necessary, it is done in a sequential order before the trajectory generation process for that vehicle as described earlier. Each vehicle is assigned only one target at a time. Multiple target assignments to one UAV will be a topic of further research. A cost function is evaluated sequentially for each UAV (that has not been assigned any target in the current assignment) with respect to all suspected target locations. The target which has the smaller value of objective function is assigned to that UAV. The sub-objectives for each assignment are:

Objective 1 Maximize the number of vehicles prosecuting each target (to maximize survivability). This objective can be represented by assigning a value to the team size for each assigned target. A number can be identified that encodes how much better a larger team is than a smaller team. A monotonically increasing function that increases dramatically between 1 and 3 team members, indicating that a minimally acceptable team consists of two members. So if m_j is the number of vehicles already assigned to a target j, then the following objective will determine the

cost for assigning $m_j + 1$ UAVs to that target

$$J_1(m_j) = \frac{1}{2}\left(1 + \frac{m_j - 2}{\sqrt{s_{J_1}^2 + (m_j - 2)^2}}\right) \tag{7.7.24}$$

where s_{J_1} is the softness parameter of the above function. The choice depends upon the designer but a reasonable value is 2.

Objective 2 Maximize the number of targets visited. This heuristic drives the team towards visiting as many targets as possible. This is modelled in the objective function so that each target should be visited by at least one UAV

$$J_2(m_j) = \frac{1}{2}\left(1 - \frac{m_j - 1}{\sqrt{s_{J_2}^2 + (m_j - 1)^2}}\right) \tag{7.7.25}$$

As can be observed from the above equation, after $m_j = 1$, J_2 decreases rapidly with the number of UAVs. For different targets, this objective function may have different values.

Objective 3 Maximize the effectiveness for target service. This means that an appropriate UAV either TR class or A class or both should proceed to the location for the best demanded action. The probability of target existence at that location also affects the choice of the UAV class and is modelled as

$$J_3(p_j, \xi_{ij}^X) = (1 - p_j) * \exp(-\xi_{ij}^X) \tag{7.7.26}$$

Objective 4 Minimize the path length to the target using

$$J_4 = d_j \tag{7.7.27}$$

where d_{ij} is the normalized distance of the UAV i to the target j.

Objective 5 Minimize the threat exposure. The hit probability has been described earlier and can be used here as the objective function

$$J_5 = P_{hit} \tag{7.7.28}$$

All objectives except the second are myopic objectives whereas second is a team objective. For each task, the team must try to use UAVs best suited to

it. The values for all the sub-objectives described above lie between 0 and 1 and can be combined into a single objective as

$$J = \min_{j \in T_j}(1 - J_1) * J_2 * J_3 * J_4 * J_5 \tag{7.7.29}$$

J is evaluated for each target and the target with the minimum value of J is selected. All UAVs have instantaneous and noise-free access to a centralized information base (IB), which comprises the following items:

1. The target map $\mathcal{M}^{tar}(x, y, t)\ \forall(x, y)$.
2. The threat map $\mathcal{M}^{thr}(x, y, t)\ \forall(x, y)$.
3. The task status map $\mathcal{M}^{tas}(x_{\tau_j}, y_{\tau_j}, t)\ \forall$ current targets $\tau_j, j = 1, \ldots, M$ located at (x_{τ_j}, y_{τ_j}) and id_{τ_j} is the identity of the task to be performed at target τ_j i.e., whether it is confirm ($id_{\tau_j} = 2$), attack ($id_{\tau_j} = 3$), or loss assessment ($id_{\tau_j} = 4$).
4. The UAV state vector, $S(t) = \{S_i(t)\}\ \forall u_i$.

7.8 Working Procedure

Each UAV reads and updates the IB at each step. Initially each cell has some probability of existence for a target or threat which varies from 0 to 1. The cells which have a probability for target existence greater than the threshold p_{att} are needed to be treated as a priority. These cells are believed to be without any doubt for target existence and so confirmation is not necessary there. Therefore the first task to be performed there is attack and after that other classes of UAV can come forward for loss assessment. Similarly the threshold for threat (SAM site) existence at any location (x, y) is p_{thr}. The cells where targets and threats are not suspected or impossible have $P^{tar}(x, y, 0) \leq p_s$ and $P^{tar}(x, y, 0) \leq p_s$ respectively and these cells have ignore status. The UAVs' initial positions are also given. All UAVs initially have open status. When a UAV has no task to choose (which happens when all the targets are neutralized or the UAV is ineligible for all available or associated assignable tasks), it can join a UAV which has a committed status. After all the targets are neutralized, the UAVs

come back to their base. Each UAV reports its best choice to the central information base. UAVs that have small value of the performance index J are assigned to its preferred task while other UAVs compete for the remaining choices in a sequential order until every UAV has an initial assignment. When two UAVs prefer the same task, the conflict is resolved in favour of the UAV with the smaller distance. Thus, initial assignment is purely cooperative and semi-greedy. After the initial assignment, each UAV moves towards its assigned or associated task, updating both the probability of target and threat existence in each cell it passes. When it reaches its assigned target, it performs the task and updates all the maps there and the UAV's status reverts to open. Each new assignable task is cued with an available status. At all times, all open and competing UAVs are considered for all available and associated tasks. The UAVs are processed in an order determined by the central hub. The process continues until all locations have an ignore status or some time threshold is met.

7.9 Conclusion

A decentralized cooperative control architecture has been proposed which updates target maps, threat maps in a probabilistic way using a Baysian formulation. The tasks are assigned and conflicts are removed in a sequential way. Each UAV designs its trajectory using a receding horizon formulation of the MILP that ends in a loiter pattern. A novel objective function has been developed that helps in assigning tasks to UAVs. Theoretically this architecture looks promising but extensive simulation needs to be done to validate it experimentally. Future work could incorporate learning and adaptation at the UAV and team levels so that decision-making can improve with experience and individual UAVs can develop specialized expertise. Also, including dynamics in the expertise vector for each UAV could be helpful to model losses in UAV capabilities in the event of munition use or damage.

Chapter 8

Conclusions and Suggestions for Further Research

In this thesis, high level control issues for autonomous air vehicles have been addressed. In particular, autonomous vehicle trajectory planing has been explored and several techniques developed and compared with an emphasis on real time use. First, the problem was formulated using an optimal control approach and necessary equations were derived for safe navigation of UAVs. The objective function incorporates many real life scenarios: minimum time, terrain avoidance and least risk due to SAM sites. Because of the complexity of the problem, it was simplified into smaller problems: single radar and two radar exposure minimization problems, with the the intention finding analytical solutions. An analytical solution was obtained for the single radar case. The two radar case for different strength ratios was compared with the developed Voronoi paths using numerical techniques such as the gradient method. For the single radar, it was found that trajectories do not exist for $\theta_f \geq 60^o$. So a path length or time constraint must be introduced to recover the solution. If there are a number of threats, then analytic solutions becomes difficult. Constraints such as velocity and acceleration are hard to incorporate in the above formulation as is replanning which is required in real time situations.

Next it was shown that the problem of path planning can be formulated as a mixed integer/linear constraints optimization problem (MILP). Different constraints were formulated and applied to two case study examples. Using

MILP, however, to design a whole trajectory with a planning horizon fixed at the goal, is very difficult to perform in real time because the computational effort required grows so rapidly with problem size. On the other hand big time steps can lead to inaccurate or non-implementable solutions. It was shown that this limitation can be avoided by using a receding planning horizon strategy in which MILP is used to form a shorter plan that extends towards the goal but does not necessarily reach it. When using this receding horizon approach with hard constraints for obstacle and collision avoidance, an infeasible problem can often arise, though in theory there are solutions to the whole problem. This is because the look ahead horizon is limited, and the vehicle can be led to a critical state for which MILP has no solution at the next iteration. In other words, a feasible solution for future time steps at a current time step does not guarantee a feasible MILP at each time step. The performance of a receding horizon strongly depends on the proper evaluation of the terminal penalty on the shorter plan. This evaluation is difficult when the feasibility of the path beyond the plan must be ensured. This can be further explained by the situation in which at the last time step of the planning horizon the vehicle is moving at maximum speed while its position is close to an obstacle which has not yet been spotted. Since the position of the vehicle satisfies the anti-collision constraints, this situation corresponds to a feasible solution of the MILP. At the next time step, however, the obstacle is identified and the vehicle needs to brake or turn which then exceeds the constraints on acceleration or on the vehicle manoeuvre space, a solution will not therefore be found. The proposed approach modifies the hard constraints for obstacle and collision avoidance into soft constraints in such a way that the MILP formulation remains stable with no or minimum violation of the constraints. Robustness of receding horizon control is guaranteed by modelling the constraints as soft. The efficiency of the techniques depends upon proper modelling of the mixed linear constraints and also on the time horizon. The optimality can be increased by increasing the time window but by doing this the computational load will increase. There should be a realistic compromise between optimality and computational load. The MILP is solved using commercially available software AMPL/CPLEX that

uses the well known branch and bound algorithm.

The probabilistic nature of the problem is evident for three main reasons: first of all because of the inevitable uncertainty of the measurements from the sensors, secondly for the intrinsic uncertainty of an unknown environment and finally the structure of reasoning of any intelligent system is naturally probabilistic. Due to the inherent complexity and probabilistic nature of the problem, a three dimensional probabilistic approach was suggested that depends on a nonlinear performance index. The strategy was designed by taking into consideration three main objectives: restricted areas should be avoided, threat exposure levels should be minimized and proximity of the target must be achieved. The proposed algorithm is based upon a search of a point for local minima on a disc whose centre passes through the line of sight of the target from the current point and is also perpendicular to that line. The radius of the disc is decided upon by a maximum search angle and which in turn can be decided by the maximum turn angle. The disc is divided into a suitable number of lines all passing through its centre and a point is searched along these lines. The algorithm is not only capable of finding the safe path but also takes into account real world practical constraints. The algorithm is applied in a decentralized mode, that is, each vehicle has its own processor and applies the algorithm to find its own path with consideration of collisions with other vehicles by keeping itself at at some fixed distance from the others. The novelty of the algorithm lies in its ability for use in real time due a to very low computational load in spite of the fact that it finds a path in three dimensions. The paths are locally optimal and are feasible for the UAV to follow by keeping the turn angle within certain maximum limits. The UAVs are prevented from flying at very low altitudes because of the danger of crashing into ground objects. Since each UAV has limited fuel, a compromise has to be made between risk and fuel consumption by limiting the height and search angle. One important observation of this algorithm is that while evaluating the stealthy path for the UAV, the direct distance of the vehicle from the target decreases at each path step. This property can be utilized to explore the coordinated rendezvous aspects; a topic for future work.

All the real time techniques discussed so far are based on local optimizations and perform well for planning within some limit area at each time step. If we have some global knowledge of the world then this information can be combined with local optimization techniques for improved optimality and hence sometimes a global planner is needed. Deterministic approaches are used most often in global path planners. Two deterministic techniques are the Voronoi Diagram and Visibility Graph. A hybrid approach has been presented which is based on the Voronoi graph, local optimization and grid search methods. A software package has been developed at Leicester (see Appendix A) that adds extra subroutines to modify the Voronoi code. These subroutines expand the node list by adding in nodes on borders connecting the infinity nodes and by inclusion of corner points. Furthermore, they remove those nodes outside of the considered area and include instead the intersections of the corresponding edges and the border lines. Hence, the area is explicitly and completely partitioned into cells. The efficiency of the software has been demonstrated with some example scenarios that use a dynamic programming optimization approach to select the optimal path. Constraint optimization has also been discussed. The comparison reveals that most techniques have some advantages (and drawbacks) over the other and hence should not be used independently in all situations. MILP is very flexible in adopting the constraints on velocity and acceleration but for large problems in a centralized mode, it may become intractable for real time use. Therefore, it should be used in a decentralized fashion to reduce the computational effort and also to enhance the variety and flexibility of mission goals, tighter integration with other algorithms such as optimal control capturing important analytical features for an optimal path, extended Voronoi approach for global optimality and probabilistic local minimization technique to incorporate probabilistic nature of the problem, temporal planing for low observability while moving in the high risk region needs to be explored.

As a final study, a systematic decentralized cooperative control architecture for a team of cooperating autonomous UAVs was proposed which updates the target map and threat maps in a probabilistic way using a Baysian for-

mulation. The tasks are assigned and conflicts removed in a sequential way. Each UAV designs its own trajectory using a receding horizon formulation of the MILP that ends in a loiter pattern. A novel objective function has been developed that helps in deciding tasks for the UAVs. Theoretically, this architecture looks promising but extensive simulations need to be done to validate it experimentally. Further research work could be testing of this architecture, while incorporating learning and adaptation at the UAV and team levels, so that decision-making can improve with experience and individual UAVs can develop specialized expertise. Including dynamics in an expertise vector for each UAV could also be helpful in modelling losses in a UAV's capabilities in the event of munition use or damage. Also including prediction in the assignment process, so that UAVs can anticipate tasks likely to become available in the near future and include them in their plans. Further future research directions are:

- to model and incorporate the non-isotropic UAV radar cross (RCS) showing coupling with vehicle dynamics into the integrated probabilistic model suggested in chapter 5 to reflect the risk minimization with orientation. Also the actual dynamics of a UAV are much more complicated and so ways will have to be determined that take these additional features into account but do not significantly increase computational resources required to compute a trajectory.

- Two radar case for different strength ratios was compared in section 3.7 with the developed Voronoi paths using numerical technique such as gradient method when the UAV has to sought a path between the radars. A similar numerical study could be undertaken to determine some guidelines for the UAV to decide when to go around the radars.

- A model containing additional details such as landscape features, wind conditions would be more realistic with keeping computational complexity of the method at a practical level.

Appendix A

Software Package for Waypoint Selection

A.1 Main file of the software

```
%%
%% Main program of the way-point generator -- 3-dimension case.
%%      Waseem Kamal, Dawei Gu and Ian Postlethwaite
%% Dept of Engineering, University of Leicester

clear all
close all

MyTol = 1.0e-6;
%
% Input data
%
disp('Please define the boundaries of the operation area.');

xMin = input('Enter the min of x-coordinates: ');
xMax = input('Enter the max of x-coordinates: ');
yMin = input('Enter the min of y-coordinates: ');
yMax = input('Enter the max of y-coordinates: ');

Range = [xMin xMax yMin yMax];
xRange = xMax - xMin;
yRange = yMax - yMin;

nthreat = input('Enter the number of threats: ');

for i=1:nthreat,
   fprintf('Enter the x and y coordinates of Threat Number %3.0f\n', i);
  % threat(i,1) = xMin + rand*xRange;
   threat(i,1) = input('The x-coordinate: ');
  % threat(i,2) = yMin + rand*yRange;
   threat(i,2) = input('The y-coordinate: ');
 fprintf('Enter the strength of Threat Number %3.0f\n', i);
```

```
 strength(i) = input('The strength (1,2,3): ');
end % end of the "for" loop

disp('Positions of the threats are :' );
threat
disp('Please press any key to continue.');
pause;
% Add extra threats at the (four) corners to prevent the path along box edges.
%nthreat = nthreat + 1;
%threat(nthreat,1) = xMin;
%threat(nthreat,2) = yMin;
%strength(nthreat) = 2;

%nthreat = nthreat + 1;
%threat(nthreat,1) = xMax;
%threat(nthreat,2) = yMin;
%strength(nthreat) = 2;

%nthreat = nthreat + 1;
%threat(nthreat,1) = xMax;
%threat(nthreat,2) = yMax;
%strength(nthreat) = 2;

%nthreat = nthreat + 1;
%threat(nthreat,1) = xMin;
%threat(nthreat,2) = yMax;
%strength(nthreat) = 2;
%
% May include input data correction option here to amend the coordinates of threats.
%
% Plot threat locations.
%
figure(1),
plot(threat (:,1), threat (:,2), 'r+','linewidth',3);
axis([xMin xMax yMin yMax]);
title('Threat Locations');
disp('Please press any key to continue.');
pause;
%
% Create Voronoi graph.
%
figure(2),
[vtemp,ctemp]=voronoin(threat);
voronoi(threat (:,1), threat (:,2));
title('Voronoi Graph from Matlab');
axis([xMin max(xMax,max(vtemp(:,1))) yMin max(yMax,max(vtemp(:,2)))]);
disp('Please press any key to continue.');
pause;

[NodeList1,CellArray1,NodeList0,CellArray0,RangeExtend] = InftyPoints
(threat,Range,MyTol);
```

```
[NodeList2,CellArray2] = CornerPoints(threat,RangeExtend,NodeList1,CellArray1,MyTol);

[NodeList3,CellArray3] = OutsideNodes(threat,Range,NodeList2,CellArray2,MyTol);

[NodeList4,CellArray4] = CornerPoints(threat,Range,NodeList3,CellArray3,MyTol);
%
% Plot the Voronoi graph.
%
figure(3),
disp('Plotting the Voronoi Diagram');
PlotCells(Range,threat,NodeList4,CellArray4);
title('Improved Voronoi Diagram')
disp('Please press any key to continue.');
pause;
%hold off
%
% Enter the starting (Ps) and end (Pe) points of the mission.
%
%Ps = [xMin+rand*xRange yMin+rand*yRange];
Ps = input('Enter the (x,y)-coordinates of the starting point in the form [x y]: ');
zPs = input('Enter the altitude of the starting point (between 1 and 10): ');
%Pe = [xMin+rand*xRange yMin+rand*yRange];
Pe = input('Enter the (x,y)-coordinates of the end point in the form [x y]: ');
zPe = input('Enter the altitude of the end point (between 1 and 10): ');
disp('Position of the starting point is :' );
Ps
disp('Position of the end point is :' );
Pe
disp('Please press any key to continue.');
pause;

[TotalNodes,FirstNode,LastNode,FinalNodeList,FinalConMatrix] = augment(Range,threat,
NodeList4,CellArray4,Ps,Pe,MyTol);
%
% Calculation of cost for all edges (sides).
%
w = 0.5;   % weighting factor between the flight length and risk

height = 7;  % set the height in the 2-dim path case, roughly the average in risk in terms of
             % altitude
[nnz,D,row,col] = CostCalculation(threat,strength,FinalNodeList,FinalConMatrix,w,height);
%
% Calculation of optimal path (a series of way points).
%

[splen,path] = DijkstraAlgorithm(TotalNodes,FirstNode,LastNode,nnz,D,row,col);
%
% Output restuls.
%
WayPoints = [FinalNodeList(path,1) FinalNodeList(path,2)];
```

```
figure(4),
disp('Plotting Optimal Path');
PlotCells(Range,threat,NodeList4,CellArray4);
hold on
axis(Range);
plot(WayPoints(:,1),WayPoints(:,2),'r','linewidth',2);
%title('Optimal Path 1 (based on Voronoi graph)')
disp('Please press any key to continue.');
pause;

ContourThreat(Range,threat,strength,height);
disp('Please press any key to continue.');
pause;
riskTol = 0.1;    % at 90% confidence level
MinStep = 0.5; % minimum search step (should consider the actual flight constraints)
%
% Local optimization to best tune the path
%
path2D = LocMinPer2(Range,threat,strength,WayPoints,riskTol,MinStep,height,MyTol);
hold on
plot(path2D(:,1),path2D(:,2),'m','linewidth',2);
hold off
fprintf('Coordinates of the optimal path (way points) after local searches are:\n');
path2D
%
% 3D path below
%
zMin = 1; % define the altitude range, (1:10, or 5:10)
zMax = 10;
zStep = 0.5;
Ps3D = [Ps zPs];   % starting point
Pe3D = [Pe zPe]; % end point
[OptCost,path3D,Node3D] = PathGen3D(path2D,zMin,zMax,zStep,Ps3D,Pe3D,threat,
strength,w,MyTol);
WayPoint3D = [Node3D(path3D,1) Node3D(path3D,2) Node3D(path3D,3)]
[TotalRisk,TotalLength,WorstNode,PeakRisk,PRiskThreat] = PathEvaluate3D(WayPoint3D,
threat,strength,w,MyTol);
n3D = size(WayPoint3D,1);
hn3D = floor(n3D/2);
figure(5),
plot3(WayPoint3D(1:hn3D,1),WayPoint3D(1:hn3D,2),WayPoint3D(1:hn3D,3),'r','linewidth',2),
grid,xlabel('x'),ylabel('y'),zlabel('z');
hold on
plot3(WayPoint3D(hn3D:n3D,1),WayPoint3D(hn3D:n3D,2),WayPoint3D(hn3D:n3D,3),'b',
'linewidth',2);
title('3-D Flight Path')
hold off
TotalRisk
TotalLength
PeakRisk
PRiskThreat
```

A.2 Function to remove infinity nodes

```
function [NewNodeList,NewCellArray,NodeList,CellArray,RangeExtend] = InftyPoints
(threat,Range,MyTol)
% Codes for removing infty nodes, with new nodes on (an extended) boundaries.
%
% Inputs for this part: threat, Range
%   where, threat -- coordinate list of threats
%          Range -- fixed boundaries
%          MyTol -- user-set tolerance
%
% Outpus: NewNodeList -- NB. It still contains infty node as No.1 (but
%                        not appear in cells ).
%         NewCellArray -- cell array, without infty nodes
%         NodeList -- coordinate list of nodes (vertice) including infty nodes
%         CellArray -- cell array,
%         RangeExtend -- extended boundaries
%
% NB. Using "end" in cell call may increase computation; could be improved later on.
%
%       dwg 17-01-04
%
[v,c] = voronoin(threat);
n = size(v,1); % number of (original) nodes
nthreat = size(threat,1); % number of threats (and cells)
NodeList = v;
CellArray = c;
%
% Get the boundaries
%
xminR = Range(1);
xmaxR = Range(2);
yminR = Range(3);
ymaxR = Range(4);
%
% Calculate the extended boundaries
%
xvec = [v(2:end,1)' threat (:,1)'];
yvec = [v(2:end,2)' threat (:,2)'];
xMin = min(min(xvec),xminR);
xMax = max(max(xvec),xmaxR);
yMin = min(min(yvec),yminR);
yMax = max(max(yvec),ymaxR);

BB = max([abs(xMin) abs(xMax) abs(yMin) abs(yMax)]);
%
% Expand further 10%
%
if(xMin<0)
   xmin = 1.1*xMin;
elseif(xMin>0)
```

```
    xmin = 0.9*xMin;
else
    xmin = -0.1*BB;
end

if(xMax<0)
    xmax = 0.9*xMax;
elseif(xMax>0)
    xmax = 1.1*xMax;
else
    xmax = 0.1*BB;
end

if(yMin<0)
    ymin = 1.1*yMin;
elseif(yMin>0)
    ymin = 0.9*yMin;
else
    ymin = -0.1*BB;
end

if(yMax<0)
    ymax = 0.9*yMax;
elseif(yMax>0)
    ymax = 1.1*yMax;
else
    ymax = 0.1*BB;
end

RangeExtend = [xmin xmax ymin ymax];
%
% Move the infty node to the end of node list, if there is one in the cell.
%
for i=1:nthreat,
    ff = find(c{i} == 1);
    if(~isempty(ff))  % cell has infty node
        tt = length(c{i});
        while(c{i}(tt) ~= 1) % shift node forward by one, and move 1st to last
            temp1 = c{i}(1);
            for j = 1:tt-1,
                c{i}(j) = c{i}(j+1);
            end
            c{i}(tt) = temp1;
        end % end of while loop
    end % end of cells containing infty
end % end of i loop
%
% Create working cell arrays (cc will be the final one)
%
cc = c;
vv = v;
```

```
nNewNodes = 0;

for i=1:nthreat-1,
    tti = length(cc{i});
if(cc{i}(end) == 1) % cell has infty node
     NodeFirst = cc{i}(1);
     NodeLast = cc{i}(end-1);
for j=i+1:nthreat,              % Consider NodeLast case
    ttj = length(cc{j});
if(cc{j}(end) == 1) % cell j has infty node
if(cc{j}(end-1) == NodeLast) % the last but one node is a common node
if(abs(threat(i,2)-threat(j,2))<MyTol) % vertical line, 2 intersection points
    xx1 = (threat(i,1)+threat(j,1))/2;
    yy1 = ymin;
    xx2 = xx1;
    yy2 = ymax;
if(IsNearest(xx1,yy1,j,threat,MyTol)) % threat j(i) is the ( strictly ) nearest
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx1;
    vv(n+nNewNodes, 2) = yy1;
    cc{i}(end) = n+nNewNodes;
    cc{i}(end+1) = 1;
    cc{j}(end) = n+nNewNodes;
    cc{j}(end+1) = 1;
elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx2;
                    vv(n+nNewNodes, 2) = yy2;
                    cc{i}(end) = n+nNewNodes;
                    cc{i}(end+1) = 1;
                    cc{j}(end) = n+nNewNodes;
                    cc{j}(end+1) = 1;
end % end of vertical line case
elseif(abs(threat(i,1)-threat(j,1))<MyTol)
                                % horizontal line , 2 intersection points
                xx1 = xmin;
                yy1 = (threat(i,2)+threat(j,2))/2;
                xx2 = xmax;
                yy2 = yy1;
if(IsNearest(xx1,yy1,j,threat,MyTol)) % threat j(i) is the ( strictly ) nearest
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx1;
                    vv(n+nNewNodes, 2) = yy1;
                    cc{i}(end) = n+nNewNodes;
                    cc{i}(end+1) = 1;
                    cc{j}(end) = n+nNewNodes;
                    cc{j}(end+1) = 1;
 elseif(IsNearest(xx2,yy2,j,threat,MyTol))   % (xx2,yy2)
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx2;
```

```
                    vv(n+nNewNodes, 2) = yy2;
                    cc{i}(end) = n+nNewNodes;
                    cc{i}(end+1) = 1;
                    cc{j}(end) = n+nNewNodes;
                    cc{j}(end+1) = 1;
 end % end of horizontal line case
 else        % general case, will have 4 intersection points
   slope = (threat(j,1)-threat(i,1))/(threat(i,2)-threat(j,2));
   xx1 = xmin; % intersection point on x=xmin
   yy1 = vv(NodeLast,2) + slope*(xmin - vv(NodeLast,1));
   xx2 = xmax; % intersection point on x=xmax
   yy2 = vv(NodeLast,2) + slope*(xmax - vv(NodeLast,1));
   xx3 = vv(NodeLast,1) + (ymin - vv(NodeLast,2))/slope;
   % intersection point on y=ymin
   yy3 = ymin;
   xx4 = vv(NodeLast,1) + (ymax - vv(NodeLast,2))/slope;
   % intersection point on y=ymax
   yy4 = ymax;
if((yy1 <= ymax+MyTol) & (ymin-MyTol <= yy1) & IsNearest
  (xx1,yy1,j,threat,MyTol)) % threat j(i) is the ( strictly ) nearest
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx1;
                    vv(n+nNewNodes, 2) = yy1;
                    cc{i}(end) = n+nNewNodes;
                    cc{i}(end+1) = 1;
                    cc{j}(end) = n+nNewNodes;
                    cc{j}(end+1) = 1;
elseif((yy2 <= ymax+MyTol) & (ymin-MyTol <= yy2) & IsNearest
                      (xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx2;
                    vv(n+nNewNodes, 2) = yy2;
                    cc{i}(end) = n+nNewNodes;
                    cc{i}(end+1) = 1;
                    cc{j}(end) = n+nNewNodes;
                    cc{j}(end+1) = 1;
elseif((xx3 <= xmax+MyTol) & (xmin-MyTol <= xx3) & IsNearest
                      (xx3,yy3,j,threat,MyTol)) % (xx3,yy3)
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx3;
                    vv(n+nNewNodes, 2) = yy3;
                    cc{i}(end) = n+nNewNodes;
                    cc{i}(end+1) = 1;
                    cc{j}(end) = n+nNewNodes;
                    cc{j}(end+1) = 1;
  elseif((xx4 <= xmax+MyTol) & (xmin-MyTol <= xx4) & IsNearest
                      (xx4,yy4,j,threat,MyTol)) % (xx4,yy4)
                    nNewNodes = nNewNodes + 1;
                    vv(n+nNewNodes, 1) = xx4;
                    vv(n+nNewNodes, 2) = yy4;
                    cc{i}(end) = n+nNewNodes;
```

```
                cc{i}(end+1) = 1;
                cc{j}(end) = n+nNewNodes;
                cc{j}(end+1) = 1;
              end %
            end
elseif((ttj > 2) & (cc{j}(1) == NodeLast))
        % common node is the first node of cell j,
        % and cell j has more than two nodes (including infty)
if(abs(threat(i,2)-threat(j,2))<MyTol)
              % vertical line, 2 intersection points
               xx1 = (threat(i,1)+threat(j,1))/2;
               yy1 = ymin;
               xx2 = xx1;
               yy2 = ymax;
 if(IsNearest(xx1,yy1,j,threat,MyTol))
                % threat j(i) is the ( strictly ) nearest
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx1;
                vv(n+nNewNodes, 2) = yy1;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
      % add new node as the first node, and shift the rest
 elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx2;
                vv(n+nNewNodes, 2) = yy2;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
      % add new node as the first node, and shift the rest
              end % end of vertical line case
   elseif(abs(threat(i,1)-threat(j,1))<MyTol)
      % horizontal line, 2 intersection points
               xx1 = xmin;
               yy1 = (threat(i,2)+threat(j,2))/2;
               xx2 = xmax;
               yy2 = yy1;
     if(IsNearest(xx1,yy1,j,threat,MyTol))
        % threat j(i) is the ( strictly ) nearest
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx1;
                vv(n+nNewNodes, 2) = yy1;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
              elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx2;
                vv(n+nNewNodes, 2) = yy2;
```

```
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
              end % end of horizontal line case
            else        % general case, will have 4 intersection points
              slope = (threat(j,1)-threat(i,1))/(threat(i,2)-threat(j,2));
              xx1 = xmin; % intersection point on x=xmin
              yy1 = vv(NodeLast,2) + slope*(xmin - vv(NodeLast,1));
              xx2 = xmax; % intersection point on x=xmax
              yy2 = vv(NodeLast,2) + slope*(xmax - vv(NodeLast,1));
              xx3 = vv(NodeLast,1) + (ymin - vv(NodeLast,2))/slope;
              % intersection point on y=ymin
              yy3 = ymin;
              xx4 = vv(NodeLast,1) + (ymax - vv(NodeLast,2))/slope;
              % intersection point on y=ymax
              yy4 = ymax;
 if((yy1 <= ymax+MyTol) & (ymin-MyTol <= yy1) & IsNearest
   (xx1,yy1,j,threat,MyTol)) % threat j(i) is the ( strictly ) nearest
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx1;
                vv(n+nNewNodes, 2) = yy1;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
 elseif((yy2 <= ymax+MyTol) & (ymin-MyTol <= yy2) & IsNearest
                        (xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx2;
                vv(n+nNewNodes, 2) = yy2;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
              % add new node as the first node, and shift the rest
 elseif((xx3 <= xmax+MyTol) & (xmin-MyTol <= xx3) & IsNearest
                     (xx3,yy3,j,threat,MyTol)) % (xx3,yy3)
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx3;
                vv(n+nNewNodes, 2) = yy3;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
                cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
elseif((xx4 <= xmax+MyTol) & (xmin-MyTol <= xx4) & IsNearest
                           (xx4,yy4,j,threat,MyTol)) % (xx4,yy4)
                nNewNodes = nNewNodes + 1;
                vv(n+nNewNodes, 1) = xx4;
                vv(n+nNewNodes, 2) = yy4;
                cc{i}(end) = n+nNewNodes;
                cc{i}(end+1) = 1;
```

```
                cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
  end
  end
  elseif((tti>2) & (cc{j}(end-1) == NodeFirst))
      % the last but one node is a common node
  if(abs(threat(i,2)-threat(j,2))<MyTol)
                                % vertical line , 2 intersection points
                xx1 = (threat(i,1)+threat(j,1))/2;
                yy1 = ymin;
                xx2 = xx1;
                yy2 = ymax;
if(IsNearest(xx1,yy1,j,threat,MyTol))
                % threat j(i) is the ( strictly ) nearest
      nNewNodes = nNewNodes + 1;
      vv(n+nNewNodes, 1) = xx1;
      vv(n+nNewNodes, 2) = yy1;
      cc{i} = [n+nNewNodes cc{i}];
            % add new node as the 1st one, and shift the rest backwards
      cc{j}(end) = n+nNewNodes;
      cc{j}(end+1) = 1;
 elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
      nNewNodes = nNewNodes + 1;
      vv(n+nNewNodes, 1) = xx2;
      vv(n+nNewNodes, 2) = yy2;
      cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
      cc{j}(end) = n+nNewNodes;
      cc{j}(end+1) = 1;
 end % end of vertical line case
 elseif(abs(threat(i,1)-threat(j,1))<MyTol)
                        % horizontal line , 2 intersection points
      xx1 = xmin;
      yy1 = (threat(i,2)+threat(j,2))/2;
      xx2 = xmax;
      yy2 = yy1;
 if(IsNearest(xx1,yy1,j,threat,MyTol))
                          % threat j(i) is the ( strictly ) nearest
      nNewNodes = nNewNodes + 1;
      vv(n+nNewNodes, 1) = xx1;
      vv(n+nNewNodes, 2) = yy1;
cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
      cc{j}(end) = n+nNewNodes;
      cc{j}(end+1) = 1;
  elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
      nNewNodes = nNewNodes + 1;
      vv(n+nNewNodes, 1) = xx2;
      vv(n+nNewNodes, 2) = yy2;
      cc{i} = [n+nNewNodes cc{i}];
                % add new node as the 1st one, and shift the rest backwards
```

```
      cc{j}(end) = n+nNewNodes;
      cc{j}(end+1) = 1;
 end % end of horizontal line case
 else        % general case, will have 4 intersection points
     slope = (threat(j,1)-threat(i,1))/(threat(i,2)-threat(j,2));
     xx1 = xmin; % intersection point on x=xmin
     yy1 = v(NodeFirst,2) + slope*(xmin - vv(NodeFirst,1));
     xx2 = xmax; % intersection point on x=xmax
     yy2 = v(NodeFirst,2) + slope*(xmax - vv(NodeFirst,1));
     xx3 = v(NodeFirst,1) + (ymin - vv(NodeFirst,2))/slope;
                            % intersection point on y=ymin
     yy3 = ymin;
     xx4 = v(NodeFirst,1) + (ymax - vv(NodeFirst,2))/slope;
                            % intersection point on y=ymax
     yy4 = ymax;
 if((yy1 <= ymax+MyTol) & (ymin-MyTol <= yy1) & IsNearest
     (xx1,yy1,j,threat,MyTol)) % threat j(i) is the ( strictly ) nearest
     nNewNodes = nNewNodes + 1;
     vv(n+nNewNodes, 1) = xx1;
     vv(n+nNewNodes, 2) = yy1;
     cc{i} = [n+nNewNodes cc{i}];
     % add new node as the 1st one, and shift the rest backwards
     cc{j}(end) = n+nNewNodes;
     cc{j}(end+1) = 1;
 elseif((yy2 <= ymax+MyTol) & (ymin-MyTol <= yy2) & IsNearest
     (xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
     nNewNodes = nNewNodes + 1;
     vv(n+nNewNodes, 1) = xx2;
     vv(n+nNewNodes, 2) = yy2;
     cc{i} = [n+nNewNodes cc{i}];
        % add new node as the 1st one, and shift the rest backwards
     cc{j}(end) = n+nNewNodes;
     cc{j}(end+1) = 1;
 elseif((xx3 <= xmax+MyTol) & (xmin-MyTol <= xx3) & IsNearest
                            (xx3,yy3,j,threat,MyTol)) % (xx3,yy3)
     nNewNodes = nNewNodes + 1;
     vv(n+nNewNodes, 1) = xx3;
     vv(n+nNewNodes, 2) = yy3;
     cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
     cc{j}(end) = n+nNewNodes;
     cc{j}(end+1) = 1;
elseif((xx4 <= xmax+MyTol) & (xmin-MyTol <= xx4) & IsNearest
                             (xx4,yy4,j,threat,MyTol)) % (xx4,yy4)
     nNewNodes = nNewNodes + 1;
     vv(n+nNewNodes, 1) = xx4;
     vv(n+nNewNodes, 2) = yy4;
     cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
     cc{j}(end) = n+nNewNodes;
     cc{j}(end+1) = 1;
```

```
end %
end
elseif(( tti>2) & (ttj > 2) & (cc{j}(1) == NodeFirst))
            % common node is the first node of cell j,
            % and cell j has more than 2 nodes (including infty node)
if(abs(threat(i,2)-threat(j,2))<MyTol)
                                % vertical line, 2 intersection point
    xx1 = (threat(i,1)+threat(j,1))/2;
    yy1 = ymin;
    xx2 = xx1;
    yy2 = ymax;
 if(IsNearest(xx1,yy1,j,threat,MyTol))
                                    % threat j(i) is the ( strictly ) nearest
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx1;
    vv(n+nNewNodes, 2) = yy1;
    cc{i} = [n+nNewNodes cc{i}];
              % add new node as the 1st one, and shift the rest backwards
    cc{j} = [n+nNewNodes cc{j}];
                    % add new node as the first node, and shift the rest
 elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx2;
    vv(n+nNewNodes, 2) = yy2;
    cc{i} = [n+nNewNodes cc{i}];
              % add new node as the 1st one, and shift the rest backwards
    cc{j} = [n+nNewNodes cc{j}];
                    % add new node as the first node, and shift the rest
%                     disp('xx2 yy2')
 end % end of vertical line case
 elseif(abs(threat(i,1)-threat(j,1))<MyTol)
                                % horizontal line, 2 intersection points
    xx1 = xmin;
    yy1 = (threat(i,2)+threat(j,2))/2;
    xx2 = xmax;
    yy2 = yy1;
 if(IsNearest(xx1,yy1,j,threat,MyTol))
                                    % thrcat j(i) is the ( strictly ) nearest
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx1;
    vv(n+nNewNodes, 2) = yy1;
    cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
    cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
 elseif(IsNearest(xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx2;
    vv(n+nNewNodes, 2) = yy2;
    cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
```

```
    cc{j} = [n+nNewNodes cc{j}];
                    % add new node as the first node, and shift the rest
  end % end of horizontal line case
  else        % general case, will have 4 intersection points
    slope = (threat(j,1)-threat(i,1))/(threat(i,2)-threat(j,2));
    xx1 = xmin; % intersection point on x=xmin
    yy1 = v(NodeFirst,2) + slope*(xmin - vv(NodeFirst,1));
    xx2 = xmax; % intersection point on x=xmax
    yy2 = v(NodeFirst,2) + slope*(xmax - vv(NodeFirst,1));
    xx3 = v(NodeFirst,1) + (ymin - vv(NodeFirst,2))/slope;
                                    % intersection point on y=ymin
    yy3 = ymin;
    xx4 = v(NodeFirst,1) + (ymax - vv(NodeFirst,2))/slope;
                                    % intersection point on y=ymax
    yy4 = ymax;
 if((yy1 <= ymax+MyTol) & (ymin-MyTol <= yy1) & IsNearest
    (xx1,yy1,j,threat,MyTol)) % threat j(i) is the ( strictly ) nearest
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx1;
    vv(n+nNewNodes, 2) = yy1;
    cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
    cc{j} = [n+nNewNodes cc{j}];
                  % add new node as the first node, and shift the rest
%                   disp('xx1 yy1')
elseif((yy2 <= ymax+MyTol) & (ymin-MyTol <= yy2) & IsNearest
                                (xx2,yy2,j,threat,MyTol)) % (xx2,yy2)
    nNewNodes = nNewNodes + 1;
    vv(n+nNewNodes, 1) = xx2;
    vv(n+nNewNodes, 2) = yy2;
    cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
    cc{j} = [n+nNewNodes cc{j}];
                  % add new node as the first node, and shift the rest
%                   disp('xx2 yy2')
elseif((xx3 <= xmax+MyTol) & (xmin-MyTol <= xx3) & IsNearest
                                (xx3,yy3,j,threat,MyTol)) % (xx3,yy3)
   nNewNodes = nNewNodes + 1;
   vv(n+nNewNodes, 1) = xx3;
   vv(n+nNewNodes, 2) = yy3;
   cc{i} = [n+nNewNodes cc{i}];
          % add new node as the 1st one, and shift the rest backwards
   cc{j} = [n+nNewNodes cc{j}];
                  % add new node as the first node, and shift the rest
%                   disp('xx3 yy3')
elseif((xx4 <= xmax+MyTol) & (xmin-MyTol <= xx4) & IsNearest
                                (xx4,yy4,j,threat,MyTol)) % (xx4,yy4)
   nNewNodes = nNewNodes + 1;
   vv(n+nNewNodes, 1) = xx4;
   vv(n+nNewNodes, 2) = yy4;
   cc{i} = [n+nNewNodes cc{i}];
```

```
                % add new node as the 1st one, and shift the rest backwards
    cc{j} = [n+nNewNodes cc{j}];
                % add new node as the first node, and shift the rest
%       disp('xx4 yy4')
end %
end % end of cell{i}(1)=cell{j}(1)
end  % end of cases which will produce new nodes
%       ( cells  i and j share common finite node)
end % end of cell  j having infty  node
end % end of j loop
end  % end of if loop ( cell  i has infty  node)
end % end of i loop

for i=1:nthreat,  % delete the infty  node in  cells
   tti  = length(cc{i});
   if(cc{i}(tti) == 1)
      cc{i} = cc{i}(1:tti-1);
   end
end

%
% Prepare output data
%

NewNodeList = vv;
NewCellArray = cc;

%
```

A.3 Function to include corner points in the node list

```
function [NewNodeList,NewCellArray] = CornerPoints(threat,Range,v,c,MyTol)

% Codes for including corner points in the node list (if they are not already there).
%
% Inputs: threat, Range, v, c
%   where, threat -- coordinate list of threats
%          Range -- fixed boundaries
%          v -- coordinate list of nodes (vertice)
%          c -- cell array
%          MyTol -- user-defined tolerance
%
% Outpus: NewNodeList -- NB. It still contains infty node as No.1 (but
%                        not appear in cells ).
%         NewCellArray
%
%       dwg 24-01-04
%  Could be simplified.

n = size(v,1); % number of (original) nodes
nthreat = size(threat,1); % number of threats (and cells)

%
% Create working array and vector
%
cc = c;
vv = v;
CornerAlreadyIn = zeros(4,1);
          % record the node number, if the corner is already a node
%
% Corners 1 -- 4, from the bottom-left corner, in anti-clockwise rotation
%
CornersV(1,1) = Range(1);
CornersV(1,2) = Range(3);
CornersV(2,1) = Range(2);
CornersV(2,2) = Range(3);
CornersV(3,1) = Range(2);
CornersV(3,2) = Range(4);
CornersV(4,1) = Range(1);
CornersV(4,2) = Range(4);
%
% use a new range vector RangeN; perhaps can make corners 2-4 a loop
%
RangeN = zeros(4,1);
RangeN(1) = Range(1);
RangeN(2) = Range(3);
RangeN(3) = Range(2);
RangeN(4) = Range(4);
```

```
CornerFlag = zeros(4,1);
%
% Check if a corner is already in a cell
%
cFlag = 0;
for i=1:4,
   xx = CornersV(i,1);
   yy = CornersV(i,2);
   for j = 2:n,    % search vv to see if Corner i is already in
      if(SameNodes(xx,yy,vv(j,1),vv(j,2),MyTol))
         CornerFlag(i) = j;
         cFlag = cFlag + 1;
      end
   end
end
%
% Now, work with 4 corners
%
% Case 1, the bottom-left corner
%
if(CornerFlag(1) == 0) % Corner 1 is not already in the node list
   xx = CornersV(1,1);
   yy = CornersV(1,2);
   CellNumber = WhichCell(threat,xx,yy,MyTol);
                        % Find out the cell the point belongs to.
   tt = length(cc{CellNumber});
   nIndex = zeros(tt,1);
   nTotal = 0;
   nFront = 0;
 % will record the node in front of Corner 1, in anti-clockwise direction
   for j = 1:tt,    % find the node(s) which is (are) on the left edge
    if(abs(RangeN(1)-vv(cc{CellNumber}(j),1)) < MyTol) % node on left edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nFront = nIndex(j);
    end
   end
%
   if(nTotal > 1)  % find the lowest node on the left edge
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),2) < vv(nFront,2)))
            nFront = cc{CellNumber}(j);
         end
      end
   end
%
   if(nTotal == 0)  % no node on the left edge, has to search on top edge
      for j = 1:tt,    % find the node(s) which is (are) on the top edge
         if(abs(RangeN(4)-vv(cc{CellNumber}(j),2)) < MyTol) % node on top edge
            nIndex(j) = cc{CellNumber}(j);
```

```
            nTotal = nTotal + 1;
            nFront = nIndex(j);
         end
      end
      if(nTotal > 1)  % find the most left node on the top edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),1) < vv(nFront,1)))
               nFront = cc{CellNumber}(j);
            end
         end
      end
   end
%
   if(nTotal == 0)  % no node on the top edge either
      for j = 1:tt,    % find the node(s) which is (are) on the right edge
         if(abs(RangeN(3)-vv(cc{CellNumber}(j),1)) < MyTol) % node on right edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nFront = nIndex(j);
         end
      end
      if(nTotal > 1)  % find the highest node on the right edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),2) > vv(nFront,2)))
               nFront = cc{CellNumber}(j);
            end
         end
      end
   end  %end of searching on the right edge
%
% Now, find the node next nNext
%
   nIndex = zeros(tt,1);
   nTotal = 0;
   nNext = 0;    % will record the node after Corner 1, in anti-clockwise direction
   for j = 1:tt,    % find the node(s) which is (are) on the bottom edge
      if(abs(RangeN(2)-vv(cc{CellNumber}(j),2)) < MyTol) % node on bottom edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nNext = nIndex(j);
      end
   end
%
if(nTotal > 1)
               % find the most left node on the bottom edge, if there are more than 1
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),1) < vv(nNext,1)))
            nNext = cc{CellNumber}(j);
         end
      end
   end
```

```
%
   if(nTotal == 0)  % no node on the bottom edge, has to search on right edge
      for j = 1:tt,    % find the node(s) which is (are) on the right edge
         if(abs(RangeN(3)-vv(cc{CellNumber}(j),1)) < MyTol) % node on right edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
         end
      end
      if(nTotal > 1)  % find the lowest node on the right edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),2) < vv(nNext,2)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end
%
   if(nTotal == 0)  % no node on the right edge either
      for j = 1:tt,    % find the node(s) which is (are) on the top edge
         if(abs(RangeN(4)-vv(cc{CellNumber}(j),2)) < MyTol) % node on top edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
         end
      end
      if(nTotal > 1)  % find the most right node on the top edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),1) > vv(nNext,1)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end  %end of searching on the top edge
%
% Move nFront to the end of node list.
%
while(cc{CellNumber}(tt) ~= nFront)% shift node forward by one, and move 1st to last
      temp1 = cc{CellNumber}(1);
      for j = 1:tt-1,
         cc{CellNumber}(j) = cc{CellNumber}(j+1);
      end
      cc{CellNumber}(tt) = temp1;
end % end of while loop

%
% Now, insert (xx,yy) between nFront and nNext
%
   n = n + 1;
   vv(n,1) = xx;
   vv(n,2) = yy;
```

```
   if(cc{CellNumber}(1) == nNext)
      cc{CellNumber}(tt+1) = n;
   elseif(cc{CellNumber}(tt-1) == nNext)
      cc{CellNumber}(tt+1) = cc{CellNumber}(tt);
      cc{CellNumber}(tt) = n;
   else
      disp('␣Error:␣in␣inclusion␣of␣Corner␣1')
   end
end  % end of Corner 1 case
%
% Case 2, the bottom-right corner
%
if(CornerFlag(2) == 0) % Corner 2 is not already in the node list
   xx = CornersV(2,1);
   yy = CornersV(2,2);
   CellNumber = WhichCell(threat,xx,yy,MyTol);
                                    % Find out the cell the point belongs to.
   tt = length(cc{CellNumber});
   nIndex = zeros(tt,1);
   nTotal = 0;
   nFront = length(vv);
 % record the most right node on the bottom edge (there must be one node on that edge)
   for j = 1:tt,    % find the node(s) which is (are) on the bottom edge
      if(abs(RangeN(2)-vv(cc{CellNumber}(j),2)) < MyTol) % node on bottom edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nFront = nIndex(j);
      end
   end
%
   if(nTotal > 1)  % find the most right node on the bottom edge
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),1) > vv(nFront,1)))
            nFront = cc{CellNumber}(j);
         end
      end
   end
%
% Now, find the node next nNext
%
   nIndex = zeros(tt,1);
   nTotal = 0;
   nNext = 0;    % will record the node after Corner 2, in anti-clockwise direction
   for j = 1:tt,    % find the node(s) which is (are) on the right edge
      if(abs(RangeN(3)-vv(cc{CellNumber}(j),1)) < MyTol) % node on right edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nNext = nIndex(j);
      end
   end
%
```

```
   if(nTotal > 1)
              % find the lowest node on the right edge, if there are more than 1
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),2) < vv(nNext,2)))
            nNext = cc{CellNumber}(j);
         end
      end
   end
%
   if(nTotal == 0)  % no node on the right edge, has to search on top edge
      for j = 1:tt,    % find the node(s) which is (are) on the top edge
       if(abs(RangeN(4)-vv(cc{CellNumber}(j),2)) < MyTol) % node on top edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
       end
      end
      if(nTotal > 1)  % find the most right node on the top edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),1) > vv(nNext,1)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end
%
   if(nTotal == 0)  % no node on the top edge either
    for j = 1:tt,    % find the node(s) which is (are) on the left edge
     if(abs(RangeN(1)-vv(cc{CellNumber}(j),1)) < MyTol) % node on left edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
     end
    end
      if(nTotal > 1)  % find the highest node on the left edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),2) > vv(nNext,2)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end  %end of searching on the left edge
%
% Move nFront to the end of node list.
%
while(cc{CellNumber}(tt) ~= nFront)
              % shift node forward by one, and move 1st to last
      temp1 = cc{CellNumber}(1);
      for j = 1:tt-1,
         cc{CellNumber}(j) = cc{CellNumber}(j+1);
      end
```

```
      cc{CellNumber}(tt) = temp1;
   end % end of while loop
%
% Now, insert (xx,yy) between nFront and nNext
%
   n = n + 1;
   vv(n,1) = xx;
   vv(n,2) = yy;
   if(cc{CellNumber}(1) == nNext)
      cc{CellNumber}(tt+1) = n;
   elseif(cc{CellNumber}(tt-1) == nNext)
      cc{CellNumber}(tt+1) = cc{CellNumber}(tt);
      cc{CellNumber}(tt) = n;
   else
      disp('_Error:_in_inclusion_of_Corner_2')
   end
end  % end of Corner 2 case
%
% Case 3, the top-right corner
%
if(CornerFlag(3) == 0) % Corner 3 is not already in the node list
   xx = CornersV(3,1);
   yy = CornersV(3,2);
   CellNumber = WhichCell(threat,xx,yy,MyTol);
                          % Find out the cell the point belongs to.
   tt = length(cc{CellNumber});
   nIndex = zeros(tt,1);
   nTotal = 0;
   nFront = length(vv);
% record the highest node on the right edge (there must be one node on that edge)
   for j = 1:tt,  % find the node(s) which is (are) on the right edge
      if(abs(RangeN(3)-vv(cc{CellNumber}(j),1)) < MyTol) % node on right edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nFront = nIndex(j);
      end
   end
%
   if(nTotal > 1)  % find the highest node on the right edge
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),2) > vv(nFront,2)))
            nFront = cc{CellNumber}(j);
         end
      end
   end
%
% Now, find the node next nNext
%
nIndex = zeros(tt,1);
nTotal = 0;
nNext = 0; % will record the node after Corner 3, in anti-clockwise direction
```

```
   for j = 1:tt,    % find the node(s) which is (are) on the top edge
      if(abs(RangeN(4)-vv(cc{CellNumber}(j),2)) < MyTol) % node on right edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nNext = nIndex(j);
      end
   end
%
   if(nTotal > 1)
      % find the most right node on the top edge, if there are more than 1
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),1) > vv(nNext,1)))
            nNext = cc{CellNumber}(j);
         end
      end
   end
%
   if(nTotal == 0)  % no node on the top edge, has to search on left edge
      for j = 1:tt,    % find the node(s) which is (are) on the left edge
         if(abs(RangeN(1)-vv(cc{CellNumber}(j),1)) < MyTol) % node on top edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
         end
      end
      if(nTotal > 1)  % find the highest node on the left edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),2) > vv(nNext,2)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end
%
   if(nTotal == 0)  % no node on the left edge either
    for j = 1:tt,    % find the node(s) which is (are) on the bottom edge
if(abs(RangeN(2)-vv(cc{CellNumber}(j),2)) < MyTol) % node on bottom edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
end
    end
      if(nTotal > 1)  % find the most left node on the bottom edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),1) < vv(nNext,1)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end  %end of searching on the bottom edge
%
```

```
% Move nFront to the end of node list.
%
   while(cc{CellNumber}(tt) ~= nFront)
                          % shift node forward by one, and move 1st to last
      temp1 = cc{CellNumber}(1);
      for j = 1:tt-1,
         cc{CellNumber}(j) = cc{CellNumber}(j+1);
      end
      cc{CellNumber}(tt) = temp1;
   end % end of while loop
%
% Now, insert (xx,yy) between nFront and nNext
%
   n = n + 1;
   vv(n,1) = xx;
   vv(n,2) = yy;
   if(cc{CellNumber}(1) == nNext)
      cc{CellNumber}(tt+1) = n;
   elseif(cc{CellNumber}(tt-1) == nNext)
      cc{CellNumber}(tt+1) = cc{CellNumber}(tt);
      cc{CellNumber}(tt) = n;
   else
      disp('Error: in inclusion of Corner 3')
   end
end  % end of Corner 3 case

%
% Case 4, the top-left corner
%
if(CornerFlag(4) == 0) % Corner 4 is not already in the node list
   xx = CornersV(4,1);
   yy = CornersV(4,2);
   CellNumber = WhichCell(threat,xx,yy,MyTol);
                                % Find out the cell the point belongs to.
   tt = length(cc{CellNumber});
   nIndex = zeros(tt,1);
   nTotal = 0;
   nFront = length(vv);
% record the most left node on the top edge (there must be one node on that edge)
   for j = 1:tt,   % find the node(s) which is (are) on the top edge
      if(abs(RangeN(4)-vv(cc{CellNumber}(j),2)) < MyTol) % node on top edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nFront = nIndex(j);
      end
   end
%
   if(nTotal > 1)  % find the most left node on the top edge
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),1) < vv(nFront,1)))
            nFront = cc{CellNumber}(j);
```

```
        end
      end
   end
%
% Now, find the node next nNext
%
   nIndex = zeros(tt,1);
   nTotal = 0;
   nNext = 0;
   % will record the node after Corner 4, in anti-clockwise direction
   for j = 1:tt,    % find the node(s) which is (are) on the left edge
if(abs(RangeN(1)-vv(cc{CellNumber}(j),1)) < MyTol) % node on right edge
         nIndex(j) = cc{CellNumber}(j);
         nTotal = nTotal + 1;
         nNext = nIndex(j);
 end
   end
%
   if(nTotal > 1)
         % find the highest node on the left edge, if there are more than 1
      for j = 1:tt,
         if((nIndex(j) > 0) & (vv(nIndex(j),2) > vv(nNext,2)))
            nNext = cc{CellNumber}(j);
         end
      end
   end
%
   if(nTotal == 0)  % no node on the left edge, has to search on bottom edge
      for j = 1:tt,    % find the node(s) which is (are) on the bottom edge
    if(abs(RangeN(2)-vv(cc{CellNumber}(j),2)) < MyTol) % node on bottom edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
    end
      end
      if(nTotal > 1)  % find the most left node on the bottom edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),1) < vv(nNext,1)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end
%
   if(nTotal == 0)  % no node on the bottom edge either
      for j = 1:tt,    % find the node(s) which is (are) on the right edge
 if(abs(RangeN(3)-vv(cc{CellNumber}(j),1)) < MyTol) % node on right edge
            nIndex(j) = cc{CellNumber}(j);
            nTotal = nTotal + 1;
            nNext = nIndex(j);
 end
```

```
      end
      if(nTotal > 1)  % find the lowest node on the right edge
         for j = 1:tt,
            if((nIndex(j) > 0) & (vv(nIndex(j),2) < vv(nNext,2)))
               nNext = cc{CellNumber}(j);
            end
         end
      end
   end   %end of searching on the right edge
%
% Move nFront to the end of node list.
%
   while(cc{CellNumber}(tt) ~= nFront)
                  % shift node forward by one, and move 1st to last
      temp1 = cc{CellNumber}(1);
      for j = 1:tt-1,
         cc{CellNumber}(j) = cc{CellNumber}(j+1);
      end
      cc{CellNumber}(tt) = temp1;
   end % end of while loop
%
% Now, insert (xx,yy) between nFront and nNext
%
   n = n + 1;
   vv(n,1) = xx;
   vv(n,2) = yy;
   if(cc{CellNumber}(1) == nNext)
      cc{CellNumber}(tt+1) = n;
   elseif(cc{CellNumber}(tt-1) == nNext)
      cc{CellNumber}(tt+1) = cc{CellNumber}(tt);
      cc{CellNumber}(tt) = n;
   else
      disp('␣Error:␣in␣inclusion␣of␣Corner␣4')
   end
end   % end of Corner 4 case

%
% Assign output data.
%

NewNodeList = vv;

NewCellArray = cc;

%
```

A.4 Function to move outside points to the boundary

```
function [vNew,CellArrayNew] = OutsideNodes(threat,Range,NodeList,CellArray,MyTol)
%
% Codes for moving outside points to the boundary
% Inputs for this part: threat, Range, NodeList, CellArray,
% all nodes are within an extended boundary already
% no infty nodes in CellArray
% (the input threat is not necessary, nthreat=size(CellArray,1) )
%       dwg 17-01-04

n = size(NodeList,1); % number of (original) nodes
nthreat = size(threat ,1); % number of threats (and cells)
c = CellArray;
v = NodeList;
%
% Get the boundaries
%
xmin = Range(1);
xmax = Range(2);
ymin = Range(3);
ymax = Range(4);

BB = max([abs(xmin) abs(xmax) abs(ymin) abs(ymax)]);
BB = 10*BB; %%%temp
%
% Now, create a connection matrix which may contain zero rows/columns,
% infty nodes have not been deleted from v (but cells should contain
% no infty node).
%
dTemp = zeros(n);
for i= 1:nthreat,
   tt = length(c{i});
   for j = 1:tt-1,
     dTemp(c{i}(j),c{i}(j+1)) = 1;
     dTemp(c{i}(j+1),c{i}(j)) = 1;
   end
   dTemp(c{i}(tt),c{i}(1)) = 1;
   dTemp(c{i}(1),c{i}(tt)) = 1;
end

for i = 1:n,
% zero out 1st row and 1st column and diagonal elements (should be zero already)
    dTemp(1,i) = 0;
    dTemp(i,1) = 0;
    dTemp(i,i) = 0;
end

%
```

```
% Create a dIndex, and dTemp2 to store boundary nodes (their numbers)
%

dTemp2 = zeros(n);
dIndex = zeros(n);  % use of dIndex to delete repeated computation

%
% Create a vNew to store nodes, including original and new nodes (new nodes
% are on boundaries).
% Create NodesOutside (a single cell) to store nodes (in the original list ) which are
% beyond the given area Range.
%

vNew = v;
NodesOutside = cell(1);
NodeOutside{1}(1) = 1; % infty node

nNew = n;      % will be the number of total nodes later on

%
% Now, check all (original) nodes.
% Outside regions are ordered anti-clockwise, from bottom-left.
%

for i = 2:n,    % node 1 is infty
   nRegioni = WhichSubRegion(Range,v(i,1),v(i,2),MyTol);
   if(nRegioni ~= 9)       % node i is outside
      NodesOutside{1}(end+1) = i;
for j = 2:n,
% any cell will have 2 or more finite nodes (InftyPoints.m has been implemented)
         vFlag = 0;   % vFlag = 1, 2 new nodes
         TempV = zeros(5,2);
         TempV(1,1) = -BB*1.0e6; % a point not in the intersection list
         TempV(1,2) = -BB*1.0e6;
if((dTemp(i,j)> MyTol) & (dIndex(i,j) == 0))
    % there is connection (edge) and hasn't been done yet
     TempV(2,1) = xmin;
     % cooridnates of all 4 possible intersection points (with the boundaries)
     TempV(3,1) = xmax;
 if(v(j,1) == v(i,1))  % nodes i and j form a vertical line
                TempV(2,2) = ymin - BB;
                TempV(3,2) = ymax + BB;
            else
                TempV(2,2) = v(j,2) + (v(j,2)-v(i,2))*(xmin-v(j,1))/(v(j,1)-v(i,1));
                TempV(3,2) = v(j,2) + (v(j,2)-v(i,2))*(xmax-v(j,1))/(v(j,1)-v(i,1));
            end
            TempV(4,2) = ymin;
            TempV(5,2) = ymax;
            if(v(j,2) == v(i,2))   % nodes i and j form a horizontal line
                TempV(4,1) = xmin - BB;
                TempV(5,1) = xmax + BB;
```

```
            else
               TempV(4,1) = v(j,1) + (v(j,1)−v(i,1))*(ymin−v(j,2))/(v(j,2)−v(i,2));
               TempV(5,1) = v(j,1) + (v(j,1)−v(i,1))*(ymax−v(j,2))/(v(j,2)−v(i,2));
            end
%            disp('TempV =')
%            TempV
            for k=2:5,
               kFlag = 0;
               for l = 1:k−1,
                  if(SameNodes(TempV(k,1),TempV(k,2),TempV(l,1),TempV(l,2),MyTol))
                     kFlag = 1;
                  end
               end
%
               if(OnBoundary(Range,TempV(k,1),TempV(k,2),MyTol) & (kFlag == 0))
                  alpha = −BB; % negative number
                  if(v(j,1) == v(i,1))
                     alpha = (TempV(k,2) − v(i,2))/(v(j,2)−v(i,2));
                  else
                     alpha = (TempV(k,1) − v(i,1))/(v(j,1)−v(i,1));
                  end
if((alpha > 0) & (alpha < 1)) % point to be added (TempV(k) between nodes i and j)
                     if(vFlag > MyTol)  % positive
                        nNew = nNew + 1;
                        vNew(nNew,1) = TempV(k,1);
                        vNew(nNew,2) = TempV(k,2);
                        dTemp2(i,j) = −dTemp2(i,j);
                        dTemp2(j,i) = −dTemp2(j,i);
                        vFlag = vFlag + 1;
                     else
                        nNew = nNew + 1;
                        vNew(nNew,1) = TempV(k,1);
                        vNew(nNew,2) = TempV(k,2);
                        dTemp2(i,j) = nNew;
                        dTemp2(j,i) = nNew;
                        vFlag = vFlag + 1;
                     end
                  end  % end of if (test if the intersection point is required)
                  if(vFlag > 2)
                     disp('Error␣−−−␣more␣than␣2␣intersection␣points␣in␣OutsideNodes')
                     disp('␣i␣=␣')
                     i
                     disp('␣j=␣')
                     j
                     disp('␣k=␣')
                     k
                     vFlag
                     dTemp2(i,j)
                     pause;
                  end
               end % end of if
```

```
            end  % end of k loop
          end   % end of  if  (for connected nodes)
          dIndex(i,j) = 1;
          dIndex(j,i) = 1;
       end % end of j loop
    end  % end of if  (for outside nodes, do nothing for node i inside)
end  % end of i loop

%
% Now, a new cell array, with new boundary nodes.
%

cc = cell(nthreat,1);

for i = 1:nthreat,
    nNewCell = 0;  % index for the nodes of the new cell
    c{i} = [c{i}(end) c{i}];
    %% repeat the last node at the beginning in cell i node list
    nnn = length(c{i});
    for j = 1:nnn-1,
      if(MyFind(NodesOutside{1},c{i}(j),MyTol)) % outside node
        if(~MyFind(NodesOutside{1},c{i}(j+1),MyTol))
            % check the next node, if inside
          nNewCell = nNewCell + 1;
          cc{i}(nNewCell) = dTemp2(c{i}(j),c{i}(j+1));
        elseif(dTemp2(c{i}(j),c{i}(j+1)) < -MyTol)
            % next node outside, but has intersection points
          nNode = -dTemp2(c{i}(j),c{i}(j+1));
          nNodeNext = nNode + 1;
          xx1 = vNew(c{i}(j),1);  % coordinates of c{i}(j) and 2 intersection points
          yy1 = vNew(c{i}(j),2);
          xx2 = vNew(nNode,1);
          yy2 = vNew(nNode,2);
          xx3 = vNew(nNodeNext,1);
          yy3 = vNew(nNodeNext,2);
          dist1 = sqrt((xx1-xx2)^2+(yy1-yy2)^2);
          dist2 = sqrt((xx1-xx3)^2+(yy1-yy3)^2);
          if(dist1 < dist2)
              % always choose nearer point as the next node (all on a straight line)
            nNewCell = nNewCell + 1;
            cc{i}(nNewCell) = nNode;
            nNewCell = nNewCell + 1;
            cc{i}(nNewCell) = nNodeNext;
          else
            nNewCell = nNewCell + 1;
            cc{i}(nNewCell) = nNodeNext;
            nNewCell = nNewCell + 1;
            cc{i}(nNewCell) = nNode;
          end
        end
      else  % an inside node
```

```
        nNewCell = nNewCell + 1;
        cc{i}(nNewCell) = c{i}(j);
        if(MyFind(NodesOutside{1},c{i}(j+1),MyTol))
            % check the next node: if outside, record a new node
          nNewCell = nNewCell + 1;
          cc{i}(nNewCell) = dTemp2(c{i}(j),c{i}(j+1));
        end
      end
    end % end of j loop
end % i looop

CellArrayNew = cc;

%
```

A.5 Function to augment the initial and final point

```
function [NumofNodes,nPs,nPe,VV,DD] = augment(Range,threat,vv,cc,Ps,Pe,MyTol)

% Input:
% Output:
%    NumofNodes is the total number of nodes of voronoi diagram
%    nPs is starting node
%    nPe is the end node
%    VV contains the vertices of the nodes
%    CC is the cell array
%    DD is the connection matrix of nodes

n = size(vv,1);
nthreat = size(threat,1);
xs = Ps(1);
ys = Ps(2);
xe = Pe(1);
ye = Pe(2);

nPs = n+1;        % Next 6 lines added by DWG, 21-05-04
vv(nPs,1) = xs;
vv(nPs,2) = ys;

nPe = n+2;
vv(nPe,1) = xe;
vv(nPe,2) = ye;

%
% Get the boundaries
%

xmin = Range(1);
xmax = Range(2);
ymin = Range(3);
ymax = Range(4);


%
% Build a connection matrix dd
%

n = size(vv,1);  % DWG 21-05-04

dd = zeros(n);
for i= 1:nthreat,
   tt = length(cc{i});
   for j = 1:tt-1,
     dd(cc{i}(j),cc{i}(j+1)) = 1;
```

```
    dd(cc{i}(j+1),cc{i}(j)) = 1;
  end
  dd(cc{i}(tt),cc{i}(1)) = 1;
  dd(cc{i}(1),cc{i}(tt)) = 1;
end

% Identify the cells for Ps and Pe
j = 0;
for i=1:nthreat,   % May use WhichCell. dwg

    DisThreatS(i) = sqrt((xs-threat(i,1))^2+(ys-threat(i,2))^2);
    DisThreatE(i) = sqrt((xe-threat(i,1))^2+(ye-threat(i,2))^2);

end

[DisS,indS] = sort(DisThreatS);
[DisE,indE] = sort(DisThreatE);
indS = indS(1);
DisS = DisS(1);
cellS = cc{indS};
indE = indE(1);
DisE = DisE(1);
cellE = cc{indE};

cellS = [cellS  cellS (1)];

for i = 1:length(cellS)-1,
  x1 = vv(cellS(i),1);
  y1 = vv(cellS(i),2);
  x2 = vv(cellS(i+1),1);
  y2 = vv(cellS(i+1),2);
  [xQ,yQ] = intersect(x1,y1,x2,y2,xs,ys,MyTol);

 if((xmin<=xQ) & (xQ<=xmax) & (ymin<=yQ) & (yQ<=ymax)),
    % intersection point could be beyond the Range
    if(IsNearest(xQ,yQ,indS,threat,MyTol)),
      % include intersection points on the cell edges only
      n = n + 1;
      vv(n,1) = xQ;  % add into vv
      vv(n,2) = yQ;

      ddd = zeros(n-1,1);  % now expand and update dd
      dddd = zeros(1,n);
      dd = [dd ddd;dddd];
      dd(n,cellS(i)) = 1;
      dd(cellS(i),n) = 1;
      dd(n,cellS(i+1)) = 1;
      dd(cellS(i+1),n) = 1;
      dd(n,nPs) = 1;
      dd(nPs,n) = 1;
    end
```

```
   end
end

cellE = [cellE  cellE (1)];

for i = 1:length(cellE)-1,
   x1 = vv(cellE(i ),1);
   y1 = vv(cellE(i ),2);
   x2 = vv(cellE(i+1),1);
   y2 = vv(cellE(i+1),2);
   [xQ,yQ] = intersect(x1,y1,x2,y2,xe,ye,MyTol);

   if((xmin<=xQ) & (xQ<=xmax) & (ymin<=yQ) & (yQ<=ymax)),
     if(IsNearest(xQ,yQ,indE,threat,MyTol)),
       n = n + 1;
       vv(n,1) = xQ;  % add into vv
       vv(n,2) = yQ;

       ddd = zeros(n-1,1);  % now expand and update dd
       dddd = zeros(1,n);
       dd = [dd ddd;dddd];
       dd(n,cellE(i)) = 1;
       dd(cellE(i),n) = 1;
       dd(n,cellE(i+1)) = 1;
       dd(cellE(i+1),n) = 1;
       dd(n,nPe) = 1;
       dd(nPe,n) = 1;
     end
   end
end


dd=dd(2:end,2:end);
% Deleting 1st row and 1st col because they corresponds to
% the 1st node of the voronoi diagram (in vv) which is at  infinity
vv = vv(2:end,:);
% Deleting 1st node of the voronoi diagram which is at  infinity

nPs = nPs-1;
nPe = nPe-1;

k=1;
for i = 1:size(vv,1),%1
    if(vv(i,1)>=xmin & vv(i,1)<=xmax & vv(i,2)>=ymin & vv(i,2)<=ymax),%2
        VV(k,:) = vv(i,:);
        DD(k,:) = dd(i,:);
        I(k) = i;
        k = k+1;
    else%2
        if(i < nPs),%3
            nPs = nPs-1;
```

```
            nPe = nPe-1;
        end%3
    end%2
end%1


    DD = DD(:,I);

    NumofNodes = size(VV,1);
```

References

[1] M. Adams, W. Hall, M. Hanson, and G. Zacharias. Mixed initiative planning and control under uncertainty. In *Proceedings of the 1st AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conference and Workshop*, Portsmouth, VA, May 2002.

[2] M. Aicardi. Coordination and control of a team of mobile robots. In *Proceedings of the 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, France, October 1995.

[3] M. R. Anderson and A. C. Robbins. Formation flight as a cooperative game. In *Proc. of the AIAA GN&C Conf.*, pages 244–251, 1998.

[4] T. Arai and L.E. Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on Robotics and Automation*, 18:655–661, 2002.

[5] T. Balch and R.C. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14:926–939, 1998.

[6] J. L. Bander and C. C. White. A heuristic search algorithm for path determination with learning. *IEEE Transactions of Systems,Man, and Cybernetics - Part A: Systems and Humans*, January 1998.

[7] R. Beard, T. McLain, M. Goodrich, and E. Anderson. Coordinated target assignment and intercept for unmanned air vehicles. *IEEE Transactions on Robotics and Automation*, 2002.

[8] J. Bellingham, M. Tillerson, M. Alighanbari, and J. How. cooperative path planning for multiple UAVs in dynamic and uncertain environments. In *Proceedings of the 41st IEEE Conference on Decision and Control*, page 28162822, December 2002.

[9] J. Borenstein and Y. Koren. The vector field histogram, fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991.

[10] S. A. Bortoff. Path planning for unmanned air vehicles. Technical report, Air Force Research Laboratory, Air Vehicle Directorate, September 1999.

[11] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, April 1986.

[12] B. Brumitt and A. Stentz. Dynamic mission planning for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, MN, April 1996.

[13] A. E. Bryson and Y. C. Ho. *Applied Optimal Control.* Waltham, MA: Ginn and company, 1969.

[14] P. Burlina, D. DeMenthon, and L. S. Davis. Navigation with uncertainty: Reaching a goal in a high collision risk region. In *Proc. of IEEE International Conf. on Robotics and Automation*, pages 2440–2445, Nice, France, 1992.

[15] B. J. Capozzi. *Evolution-Based Path Planning and Management for Autonomous Vehicles.* Phd thesis, University of Washington, 2001.

[16] B. J. Capozzi and J. Vagners. An evolution-based alternative for autonomous motion planning. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Seattle, WA, August 2001.

[17] B. J. Capozzi and J. Vagners. Evolving (semi)-autonomous vehicles. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Montreal, Canada, August 2001.

[18] P. Chandler, M. Pachter, K. Nygard, and D. Swaroop. *Cooperative Control and Optimization*, chapter Distributed Cooperation and Control for Autonomous Air Vehicles. Kluwer, 2001.

[19] P. Chandler, M. Pachter, D. Swaroop, J. Fowler, J. Howlett, S. Rasmussen, C. Schumacher, and K. Nygard. Complexity in UAV cooperative control. In *Proc. of the ACC*, June 2002.

[20] M. Cherif. Motion planning for all terrain vehicles: A physical modelling approach for coping with dynamic and contact interaction constraints. *IEEE Transaction on Robotics and Automation*, 15(2):202–218, April 1999.

[21] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling for autonomous spacecraft. In *IJCAI99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World*, Stockholm, Sweden, August 1999.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[23] J.P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2864–2869, Leuven, Belgium, May 1998.

[24] M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *Information Systems and Operations Research*, (26):191–212, 1988.

[25] M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *Proceedings of the 6th International conference on intelligent autonomous systems*, page 115122, July 2000.

[26] W.B. Dunbar and R.M. Murray. Model predictive control of coordinated multi-vehicle formations. In *Proceedings of the 41st IEEE Internatinal Conference on Decision and Control*, 2002.

[27] A. Elnagar and A. Base. Heuristics for local path planning. *IEEE Transactions on Systems Man and Cybernetics*, 23(2):624–634, 1993.

[28] T. Estlin, G. Rabideau, D.Mutz, and S. Chien. *Using continuous planning techniques to coordinate multiple rovers*, chapter IJCAI Workshop on Scheduling and Planning. Stockholm, Sweden, August 1999.

[29] T. Estlin et al. An integrated architecture for cooperating rovers. In *Proceedings of the International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS)*, Noordwijk, The Netherlands, June 1999.

[30] J. T. Feddema, C. Lewis, and D. A. Schoenwald. Decentralized control of cooperative robotic vehicles: Theory and application. *IEEE Transactions on Robotics and Automation*, 18:852864, October 2002.

[31] K. Fischer, J.P. Mller, and M. Pischel. A model for cooperative transportation scheduling. In *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS95)*, pages 109–116, 1995.

[32] D. B. Fogel and L. J. Fogel. Optimal routing of multiple autonomous underwater vehicles through evolutionary programming. In *Proc. of Symposium on Autonomous Underwater Vehicle Technology*, pages 44–47, Washington D.C., 1990.

[33] R. Fourer, D. M. Gay, and B. W. Kernighar. *AMPL, A modeling language for mathematical programming.* The Scientific Press, 1993.

[34] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. In *Autonomous Robots*, volume 8(3), 2000.

[35] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Roboticsand Autonomous Systems*, 25:195–207, 1998.

[36] GARTEUR Action Group AG14. Autonomy in UAVs: A design challenge. *http://www.nlr.nl/projects/garteur_wan/inde x2.html*, 2003.

[37] D. P. Gillen and D. R. Jacques. Cooperative behavior schemes for improving the effectiveness of autonomous wide area search munitions. In *Proceeding of the Cooperative Control and Optimization Workshop*, Gainesville FL, December 2000.

[38] F. Giulietti, L. Pollini, and M. Innocenti. Autonomous formation flight. *IEEE Control System Magazine*, 20:34–44, December 2000.

[39] R. Glasius, A. Komoda, and S. Gielen. Neural network dynamics for path planning and obstacle avoidance. In *Neural Networks*, March 1994.

[40] M. Golfarelli, D. Maio, and S. Rizzi. Multi-agent path planning based on task-swap negotiation. In *Proceedings of the 16th UK Planning and Scheduling SIG Workshop*, Durham, England, 1997.

[41] M. Golfarelli, D. Maio, and S. Rizzi. A task-swap negotiation protocol based on the contract net paradigm. Technical report, Technical Report 005-97, Research Center for Computer Science and Telecommunication Systems, University of Bologna, Bologna, Italy, 1997.

[42] M. Golfarelli and S. Rizzi. Spatio-temporal clustering of tasks for swap-based negotiation protocols in multi-agent systems. In *Proceedings 6th International Conference on Intelligent Autonomous Systems*, page 172179, Venice, Italy, 2000.

[43] D.W. Gu, W. Kamal, and I. Postlethwaite. A UAV waypoint generator. In *AIAA 1st Intelligent Systems Technical Conference*, Chicago, Illinois, September 2004.

[44] C. Hocaoglu and A. C. Sanderson. Plannning multiple paths with evolutionary speciation. *IEEE Transctions on Evolutionary Computation*, 5:169–191, 2001.

[45] D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21:233–255, 2002.

[46] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, Febuary 1992.

[47] J. Ilari and C. Torras. 2D path planning: a configuration space heuristic approach. *International Journal of Robotics Research*, 9(1):75 – 91, February 1990.

[48] *ILOG CPLEX User's guide.* ILOG, 1999.

[49] G. Inalhan, D. Stipanovic, and C. Tomlin. Decentralized optimization with application to multiple aircraft coordination. In *Proceedings of the* 41^{st} *IEEE Conference on Decision and Control*, Las Vegas, NV, December 2002.

[50] P. A. Ioannou and J. Sun. *Robust Adaptive Control.* Upper Saddle River, NJ: PrenticeHall, 1996.

[51] M. Jun, A.I Chaudhry, and R. D'Andrea. The navigation of autonomous vehicles in uncertain dynamic environments: A case study. In *Proceedings of IEEE Conference on Decision and Control*, Las Vegas, NV, December 2002.

[52] W. A. Kamal, D. W. Gu, and I. Postlethwaite. A decentralised probabilistic framework for the path planning of autonomous vehicles. In *Proceedings of the 16th IFAC World Congress*, Prague, July 2005.

[53] W. A. Kamal, D. W. Gu, and I. Postlethwaite. Real time trajectory planning for UAVs using MILP. In *to be presented at the CDC-ECC Conference*, Seville, Spain, December 2005.

[54] W. A. Kamal, D. W. Gu, and I. Postlethwaite. MILP and its application in flight path planning. In *Proceedings of the 16th IFAC World Congress*, Prague, July 2005.

[55] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566580, 1996.

[56] Y. Kim, D. W. Gu, and I. Postlethwaite. Real-time path planning with limited information for UAVs. Technical report, Department of Engineering, University of Leicester, December 2005.

[57] D. E. Kirk. *Optimal Control Theory: An Introduction.* Englewood Cliffs, NJ: Prentice Hall, 1970.

[58] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proc. of the IEEE Conf. on Robotics and Automation*, pages 1398–1404, 1991.

[59] K. J. Kyriakopoulos and G. N. Saridis. An integrated collision prediction and avoidance scheme for mobile robots in non-stationary environments. In *Proc. of IEEE International Conf. on Robotics and Automation*, pages 194–199, Nice, France, May 1992.

[60] T. Simon. L. Jaillet. A PRM-based motion planner for dynamically changing environments. In *IEEE Int. Conf. on Int. Robots and Systems*, 2004.

[61] J.P. Laumond. Motion planning for PLM: State of the art and perspectives. Technical report, Rapport LAAS No 04328, ISICAD-2004, Novosibirsk, Russia, June 2004.

[62] S. Lavalle and J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20:378–400, 2001.

[63] N.E. Leonard and E. Fiorelli. Virtual leaders, artificial potentials and coordinated control of groups. In *Proceedings of IEEE Conference on Decision and Control*, pages 2986–2973, Orlando, FL, December 2001.

[64] F. L. Lewis. *Optimal Control.* New York: John Wiley & Sons, 1986.

[65] T. Maddula, A. A. Minai, and M. M. Polycarpou. Multi-target assignment and path planning for groups of uavs. In *Proceedings of the Conference on Cooperative Control and Optimization*, Gainesville, FL, 2002.

[66] C. S. Mata and J. S. B. Mitchell. a new algorithm for computing shortest paths in weighted planar subdivisions. In *Symposium on Computational Geometry*, page 264273, 1997.

[67] M. B. McFarland, R. A. Zachery, and B. K. Taylor. Motion planning for reduced observability of autonomous aerial vehicles. In *Proc. of the 1999 IEEE conf. on Control Applications*, pages 231–235. IEEE press, 1999.

[68] T. McLain, P. Chandler, S. Rasmussen, and M. Pachter. Cooperative control of UAV rendezvous. In *Proc. of the ACC*, pages 2309–2314, June 2001.

[69] T. W. McLain and R. W. Beard. Trajectory planning for coordinated rendezvous of unmanned air vehicles. In *Proc. of the AIAA GN&C Conf.*, 2000.

[70] T. W. McLain and R. W. Beard. Cooperative path planning for timing-critical missions. In *Proceedings of the American Control Conference*, Denver, Colorado, June 2003.

[71] R. R. Murphy. *Introduction to AI Robotics.* MIT Press,, 2000.

[72] M. Neus and S. Maouche. Motion planning using the modified visibility graph. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 12–15, October 1999.

[73] N. J. Nilsson. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Publisher Company, 1980.

[74] K. Nygard, P. Chandler, and M. Pachter. Dynamic network optimization models for air vehicle resource allocation. In *Proc. of the ACC*, pages 1853–1856, June 2001.

[75] M. H. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proceedings of the workshop on Algorithmic Foundations of Robotics, The 1994 Workshop on the Algorithmic Foundations of Robotics*, number ISBN:1-56881-045-8, pages 19–37, San Francisco, California, United States, 1995.

[76] L. E. Parker. On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 10(6), 1996.

[77] L.E. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14:220–240, 1998.

[78] K. Passino. Cooperative control for autonomous air vehicles. In *Proceedings of the Cooperative Control and Optimization Workshop*, Gainesville FL, 2000.

[79] M. B. Pellazar. Vehicle route planning with constraints using genetic algorithms. In *IEEE National Aerospace and Electronics Conference*, pages 111–119, 1998.

[66] C. S. Mata and J. S. B. Mitchell. a new algorithm for computing shortest paths in weighted planar subdivisions. In *Symposium on Computational Geometry*, page 264273, 1997.

[67] M. B. McFarland, R. A. Zachery, and B. K. Taylor. Motion planning for reduced observability of autonomous aerial vehicles. In *Proc. of the 1999 IEEE conf. on Control Applications*, pages 231–235. IEEE press, 1999.

[68] T. McLain, P. Chandler, S. Rasmussen, and M. Pachter. Cooperative control of UAV rendezvous. In *Proc. of the ACC*, pages 2309–2314, June 2001.

[69] T. W. McLain and R. W. Beard. Trajectory planning for coordinated rendezvous of unmanned air vehicles. In *Proc. of the AIAA GN&C Conf.*, 2000.

[70] T. W. McLain and R. W. Beard. Cooperative path planning for timing-critical missions. In *Proceedings of the American Control Conference*, Denver, Colorado, June 2003.

[71] R. R. Murphy. *Introduction to AI Robotics.* MIT Press,, 2000.

[72] M. Neus and S. Maouche. Motion planning using the modified visibility graph. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 12–15, October 1999.

[73] N. J. Nilsson. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Publisher Company, 1980.

[74] K. Nygard, P. Chandler, and M. Pachter. Dynamic network optimization models for air vehicle resource allocation. In *Proc. of the ACC*, pages 1853–1856, June 2001.

[75] M. H. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proceedings of the workshop on Algorithmic Foundations of Robotics, The 1994 Workshop on the Algorithmic Foundations of Robotics*, number ISBN:1-56881-045-8, pages 19–37, San Francisco, California, United States, 1995.

[76] L. E. Parker. On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 10(6), 1996.

[77] L.E. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14:220–240, 1998.

[78] K. Passino. Cooperative control for autonomous air vehicles. In *Proceedings of the Cooperative Control and Optimization Workshop*, Gainesville FL, 2000.

[79] M. B. Pellazar. Vehicle route planning with constraints using genetic algorithms. In *IEEE National Aerospace and Electronics Conference*, pages 111–119, 1998.

[80] M. M. Polycarpou, Y. Yang, and K. M. Passino. A cooperative search framework for distributed agents. In *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*, pages 1–6, Mexico City, Mexico, September 2001.

[81] M. A. Potter and K. A. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8:1–29, 2000.

[82] A. Proud, M. Pachter, and J. D'Azzo. Close formation control. In *Proc. of the AIAA GN&C Conf.*, August 1999.

[83] S. J. Qin and T. A. Badgewell. An overview of industrial model predictive control technology. In *Chemical Process Control-V*, volume 93, pages 232–256, AlChe Symp. Ser. New York, American Institute of Chemical Engineers, 1997.

[84] A. Richards, J. Bellingham, M. Tillerson, and J. P. How. Coordination and control of multiple UAVs. In *Proc. of the AIAA GN&C Conf.*, Aug 2002.

[85] A. Richards, J. P. How, T. Schouwenaars, and E. Feron. Plume avoidance maneuver planning using mixed integer linear programming. In *Proc. of AIAA GN&C Conf.*, August 2001.

[86] A. Richards and J.P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the American Control Conference*, May 2002.

[87] R. T. Rysdyk and A. J. Calise. Nonlinear adaptive flight control using neural networks. *IEEE Control Systems Magazine*, 18, December 1998.

[88] T. W. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, page 295308, Hidden Valley, Pennsylvania, 1993.

[89] T. Schouwenaars, B. DeMoor, E. Feron, and J. P. How. Mixed integer programming for multi-vehicle path planning. In *Proc. of the ECC Conf.*, pages 2603–2608, Porto, Portugal, September 2001.

[90] T. Schouwenaars, J. How, and E. Feron. Decentralized cooperative trajectory planning of multiple aircraft with hard safety guarantees. In *AIAA Guidance, Navigation and Control Conference*, Rhode Island, Agust 2004.

[91] C. Schumacher, P. Chandler, and S. Rasmussen. Task allocation for wide area search munitions via network flow optimization. In *Proc. of the AIAA GN&C Conf.*, August 2001.

[92] D. H. Shim, H. J. Kim, and S. Sastry. Decentralized nonlinear model predictive control of multiple flying robots in dynamic environments. In *Proceedings of the* 44^{th} *IEEE Conference on Decision and Control*, Maui, HI, December 2003.

[93] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 10:11041113, December 1980.

[94] G. Song, S. Miller, , and N. Amato. Customizing PRM roadmaps at query time. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2001.

[95] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 94)*, page 33103317, May 1994.

[96] D. J. Stilwell and B. E. Bishop. Platoons of underwater vehicles. *IEEE Control System Magazine*, 20:45–52, December 2000.

[97] K. Sugihara and J. Smith. Genetic algorithms for adaptive planning of path and trajectory of a mobile robot in 2D terrains. In *IEICE Transactions on Information and Systems*, pages 309–317, 1999.

[98] Hamdy A. Taha. *Operation Research, An Introduction.* Macmillan Publishing Company, New York, 4 edition, 1987.

[99] O. Takahashi and R. J. Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143–150, April 1989.

[100] S. Thrun, A. Bucken, W. Burgard, D. Fox, T. Frohlinghaus, D. Hennig, M. Krell T. Hofmann, and T. Schimdt. *Map learning and high-speed navigation in rhino.* Cambridge, MA: MIT Press, 1998.

[101] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2), 2001.

[102] F. E. Udwadia and R. E. Kalaba. *Analytic Dynamics: A new Approach.* Cambridge University Press, 1996.

[103] P. Wahi, R. Raina, and F. N. Chowdhury. A survey of recent work in adaptive flight control. In *Proceedings of the 33rd Southeastern Symposium on System Theory*, page 711, 2001.

[104] M. P. Wellman and P. R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24:115125, September 1998.

[105] J. D. Wolfe, D. F. Chichka, and J. L. Speyer. Decentralized controllers for unmanned aerial vehicle formation flight. In *Proc. of the AIAA GN&C Conf.*, 1996.

[106] J. Xiao and L. Zhang. Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, 1(1):18–28, April 1997.

[107] A. B. Kahng Y. U. Cao, A. S. Fukunaga and F. Meng. Cooperative mobile robotics: Antecedents and directions. Technical report, Tech. Rep. 950049, 1995.

[108] Q. Zhu. Hidden markov model for dynamic obstacles avoidance of mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 7(3): 390–397, June 1991.