Algorithms for Motion Planning and Target Capturing

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

by

Mangal Kothari

Control Research Group
Department of Engineering
University of Leicester

April 2010
Abstract

Algorithms for Motion Planning and Target Capturing

Mangal Kothari

This thesis addresses the development and implementation of algorithms for unmanned air vehicles (UAVs). With advances in technology, it is relatively easy to manufacture and operate UAVs that are particularly useful for dull, dirty and dangerous operations. The success of such autonomous missions depends heavily on the planning algorithms used. A key consideration in the thesis is the path planning problem for single and multiple UAV systems in obstacle rich environments in the presence of uncertainty. Recently, rapidly-exploring random trees (RRTs) have been applied to find feasible trajectories quickly in complex motion planning problems. We use RRTs to construct trees of kinematically feasible trajectories made of waypaths, and feasibility is evaluated by checking for collisions with the predicted trajectories. When there are uncertainties acting on the system, we can identify probabilistic feasible paths by growing trees of state distributions and ensuring that the probability of constraint violation is below a pre-defined value. In addition to this, a guidance law is designed combining a pursuit law with a line-of-sight law, to track the path generated by the path planner with minimum deviation.

In the penultimate chapter, an application is presented where a multi-UAV system captures a more capable target by forming a target centred formation around it. The approach combines a consensus algorithm with a controller to develop a robust distributed control law for formation control. Under certain conditions, it is shown that a set of UAVs can form a target centred formation even when target information is not known. The effectiveness of this algorithm is demonstrated using numerical results. In the appendix, a decentralised scheme is described for target tracking using consensus theory in conjunction with a data fusion algorithm which guarantees perfect fault detection and isolation. This scheme was developed by the Leicester team. As part of this thesis the theoretical algorithm was tested experimentally on a set of real robots.
Acknowledgements

I would like to express my gratitude to advisor, Professor Ian Postlethwaite, for his inspiring guidance, discussions and support. The freedom which he gave me both in collaboration and selecting research topics proved to be immensely beneficial and will be a great resource in my future endeavours. Professor Da-Wei Gu not only discussed with me the research topics but also helped me broaden my horizon beyond a particular planning problem. His technical virtuosity and emphasis on the details have always amazed and inspired me. The atmosphere and facilities of the Control Research Group in our Department have provided an encouraging ambience for the completion of this thesis. I feel greatly indebted to my labmates Bharani, Paresh, Indira, Rosli, Rihan, Naeem, Prathyush, Kannan, Alex, Sajjad, Haleem. I am indebted to the UKIERI programme for providing the funding to carry out my research at Leicester.

The greatest pleasure of my academic life at the University of Leicester was being able to collaborate with brilliant people. Professor Jonathan P. How, at Massachusetts Institute of Technology, ignited my interest in probabilistic robust path planning during my summer internship at MIT. He and his PhD Student, Brandon Luders, were an invaluable resource for me. The short visit to Brigham Young University (MAGICC LAB) was very fruitful to me. Rajnikant Sharma a PhD Student at Brigham Young University and I spent many skype sessions on the target capturing problem and he was a willing resource for any problem that I wanted to talk about including consensus and SLAM. I am also grateful to Dr Graham Watson and other members of the Systems Engineering Innovation Centre at Loughborough for their help in setting up the experiments, which allowed us to test one of the Leicester algorithms. The feedback which I received from the UKIERI team was very valuable and helped me shape my research, especially from Prof. Debasish Ghose.

One reason I could survive the grind of PhD life year after year was the support network that I got from my friends, especially the friends that I made at IISc known collectively as the “coffee-board-freaks”. During the last few years I worked with them, especially with Sujit, Rajnikant, Joel, Ashwini and Dhananjay, their intellectual breadth and research attitude really motivated me throughout my PhD. I am lucky to have a few publications with them.

And finally, I owe a tremendous amount of gratitude to my family, which has always been...
a pillar of strength, inspiration and support. In particular, I would like to thank my parents, Mahendra Kothari and Hement Devi: I will never be able to repay the unconditional love and patience that they have always shown towards me. I am lucky to have the support of my brothers, Kushal and Jaideep. Their constant inspirations helped me to reach the highest degree in education. I would like to give my special thanks to my wife Unnati, whose love is a great strength to me.
Dedicated to
Coffee-Board-Freaks
## Contents

1 Introduction  
   1.1 Contributions ......................................................... 4  
   1.2 Organization of the thesis ........................................... 5  

2 Path following in Cluttered Environments under Windy Conditions  
   2.1 Introduction ............................................................ 7  
   2.2 Problem Statement ..................................................... 9  
   2.3 Combined pursuit plus line-of-sight guidance law ................. 10  
      2.3.1 Error dynamics: Stability conditions .......................... 11  
      2.3.2 Characteristic of the guidance law ............................ 12  
      2.3.3 Straight-line following ......................................... 12  
      2.3.4 Circular path following ........................................ 13  
      2.3.5 Effect of wind .................................................... 13  
      2.3.6 Waypath following in an urban environment .................. 15  
   2.4 Numerical results ..................................................... 16  
      2.4.1 Straight line following .......................................... 16  
      2.4.2 Circular path following ........................................ 18
2.4.3 A combination of straight line and circular arc following 20
2.4.4 Waypath following 21
2.5 Conclusions 22

3 Robust Real-time Path Planning Algorithms using Rapidly-exploring Random Trees 24

3.1 Introduction 25
3.2 Problem Description 29
  3.2.1 Scenario 29
  3.2.2 Assumptions and constraints 31
  3.2.3 Solution approach 31
3.3 Algorithms 32
  3.3.1 RRT algorithm 32
  3.3.2 Modified RRT algorithm 34
  3.3.3 Robust RRT algorithm 36
  3.3.4 Robust RRT algorithm: Execution 44
3.4 Planning for multiple UAVs 47
  3.4.1 Assumptions 50
  3.4.2 Conflict resolution 52
  3.4.3 Finding alternative paths 55
  3.4.4 Multi-UAV RRT: Algorithm 56
3.5 Numerical Results 58
  3.5.1 Off-line Robust RRT 59
  3.5.2 On-line Robust RRT for a single UAV system 67
5.2 Problem Formulation .......................................................... 111
  5.2.1 UAV and target kinematics ............................................. 112
  5.2.2 Control objectives ....................................................... 112
  5.2.3 Communication and sensing topologies .............................. 113
5.3 Theoretical Framework ....................................................... 114
  5.3.1 Target tracking with complete target information ................. 114
  5.3.2 Target tracking with incomplete target information .............. 116
  5.3.3 Cooperative control for target-capturing with accurate target information 117
  5.3.4 Cooperative target capturing with incomplete target information ... 121
5.4 Numerical results ............................................................ 123
  5.4.1 Single vehicle target tracking ......................................... 123
  5.4.2 Target-capturing with complete target information .............. 125
  5.4.3 Target-capturing with incomplete target information ............ 127
5.5 Conclusions ................................................................. 128

6 Conclusions and Future Scope ...................................................... 130

A A Robust Target Tracking Algorithm .............................................. 133
  A.1 Introduction .................................................................. 133
  A.2 Problem Formulation ...................................................... 135
  A.3 Solution Approach ........................................................ 136
    A.3.1 Algorithm I – Semi-decentralized dynamic data fusion .......... 136
    A.3.2 Algorithm II – Fault detection and isolation ...................... 136
  A.4 Implementation of the Algorithm ........................................ 137
A.4.1 Robot information ........................................ 138
A.4.2 Experimental setup ...................................... 139
A.4.3 Experimental results .................................... 142
A.5 Conclusions and Future Scope ........................... 143
List of Figures

2.1 Formulation Geometry ...................................................... 10
2.2 Straight-line path geometry ............................................. 13
2.3 Circular path geometry .................................................... 14
2.4 Trajectory of the UAV while following waypath W1 – W2 – W3 sequentially ........................................... 15
2.5 Geometry of smooth transition from one waypath to another ......................................................... 15
2.6 Trajectories of the UAV under no wind for five different cases .................................................. 17
2.7 Guidance command requirements under no wind for five different cases ........................................ 17
2.8 Distance errors under no wind for five different cases ................................................................. 18
2.9 Trajectories of the UAV under wind for five different cases .......................................................... 18
2.10 Guidance command requirements under wind for five different cases ........................................ 19
2.11 Distance errors under wind for four different cases ................................................................. 19
2.12 Trajectories of the UAV under no wind for four different cases .................................................. 20
2.13 Guidance command requirement under no wind for four different cases ........................................ 20
2.14 Distance errors under no wind for four different cases ................................................................. 20
2.15 Trajectories of the UAV under wind for four different cases .......................................................... 20
2.16 Guidance command requirement under wind for four different cases ........................................ 21
2.17 Distance errors under wind for four different cases ................................................................. 21
2.18 UAV trajectory under no wind ........................................ 22
2.19 UAV trajectory under wind condition ................................. 22
2.20 Waypath following under no wind ..................................... 23
2.21 Waypath following in urban terrain under wind influence \( v_w = 4 \text{ m/sec} \) at \( \phi_w = 23^\circ \) ............ 23

3.1 Obstacle representation ................................................... 41
3.2 Transition behaviours ..................................................... 42
3.3 Smooth switching .......................................................... 43
3.4 Three UAVs have their paths in conflict. The point of conflict in space is shown in the gray triangle. .................. 48
3.5 Flow diagram of the conflict resolution strategy ....................... 49
3.6 A multi-UAV path planning scenario, each UAV is within communication range of each other and they are trying to find de-conflicting paths. UAVs do not have next waypath to fly, and they are still searching for paths to their goals. .................................................. 50
3.7 A multi-UAV path planning scenario where all UAVs are within communication range of each other and all are trying to find de-conflicting paths. Each UAV has the next waypath to fly to, but not got the complete path. ........................................ 51
3.8 A multi-UAV path planning scenario, all UAVs are within communication range of each other and they have de-conflicting paths. ........................................ 52
3.9 UAV 3 does not change its plan whereas UAVs 1 and 2 find alternative paths to bypass conflict by introducing intermediates nodes. ........................................ 53
3.10 Sample tree generated by the robust RRT algorithm in the cluttered environment ............ 60
3.11 Sample tree with safe corridor, perpendicular red lines on the green lines (branch) show extension of corridors, which are collision free ........................................ 60
3.12 Distribution of nodes in the cluttered environment .......................... 61
3.13 Sample tree with 10 found paths that are shown in black .......................... 61
3.14 Found paths from the starting location to the goal locations ........................... 61
3.15 The shortest path among the found paths and tracked trajectory .......................... 61
3.16 Visibility graph and the shortest path for a scenario with 5 obstacles .................. 62
3.17 The shortest path in a scenario with 25 obstacles ......................................... 62
3.18 A dense obstacle rich environment, in which the starting position is represented by a black triangle and the goal location is represented by a magenta square .................. 66
3.19 Tree after 183 sec (in blue), and the found path is shown in black colour ................. 66
3.20 An obstacle rich environment of 100 obstacles in an area of 1 km \times 1 km ............ 67
3.21 Tree after 1 sec (in blue), and a complete path to the goal location is not traced yet .......... 67
3.22 A path is chosen using heuristic (line number 7 Algorithm 4) to move towards the goal location. The back tracked path is shown in blue. The path is processed to remove unnecessary nodes, the processed path is shown in black ................................................................. 68
3.23 Expansion of the tree during the tracking of the waypath, branches of the tree are shown in blue. When a path to the goal location is found, the algorithm uses optimization heuristic to find near optimal solutions. In this example, we consider that the UAV starts a little away from the starting location. ................................................................. 68
3.24 The best path is chosen from all the found solutions. The path is shown in black and the tracked path is shown in red ................................................................. 69
3.25 Expansion of the tree while tracking the waypath with emphasis on an optimization heuristic ................................................................. 69
3.26 The algorithm does not find a better path during expansion, hence it keeps the old solution, Again the tracked path is shown in red ................................................................. 69
3.27 As the goal location can be directly connected, the algorithm does not search further. The actual UAV trajectory is shown in red ................................................................. 69
3.28 Tree expansion during preparation time (the expansion algorithm is run for 1 sec) .......... 70
3.29  Tree expansion while the UAV follows the waypath. ................................. 70
3.30  Generated path and tracked path .............................................................. 71
3.31  Path obtained using the VL method ........................................................... 71
3.32  An obstacle rich environment in a terrain of $1 \text{km} \times 1 \text{km}$ with 50 polyhedral obstacles ................................. 72
3.33  Cost of paths found in each iteration .......................................................... 72
3.34  Path planning with handling of pop-up obstacles in a terrain of $1 \text{km} \times 1 \text{km}$ with 75 polyhedral obstacles (shown in blue). Pop-up obstacles are shown in red. ................................. 74
3.35  An example showing how a pop-up obstacle is avoided by generating a new path in real-time and then improving the existing path while following the current path. ................................. 74
3.36  Path planning for three UAVs in a terrain of $1 \text{km} \times 1 \text{km}$ with 50 polyhedral obstacles (shown in blue). Planned trajectories are shown as a bold line and tracked trajectories are shown as a dashed line. The planned trajectory and corresponding tracked trajectories are superimposed except at corners. ........................................... 76
3.37  Path planning for three UAVs in a terrain of $1 \text{km} \times 1 \text{km}$ with 50 polyhedral obstacles (shown in blue). Trajectories of three UAVs are in conflict. The conflict point is shown as a black dot. ........................................... 76
3.38  UAV 3 (the path of which is shown in blue) does not change its path whereas UAV 2 (path of which is shown in red) changes its path to avoid conflict. ........................................... 77
3.39  After UAV 2 changes its path, UAV 1 (path of which is shown in blue) changes its path to avoid conflict. ........................................... 77

4.1  Obstacle representation, the vector $a_1, \ldots, a_4$ are the unit outward normals to the four line segments that define the obstacle. ........................................... 86
4.2  Constraint avoidance diagram. The change of variable $V = a_1^T X_t - b_1$ converts two dimensional variable into one dimension, the satisfaction of which is determined by the sign of $V$ which is the distance between the constraint and the vehicle, defined as positive for values of $X_t$ for which constraint is satisfied, and negative for values of $X_t$ for which the constraint is violated [11]. ........................................... 86
4.3 Diagram of the CC-RRT algorithm. Given an initial state distribution at the tree root (blue), the algorithm grows a tree of state distributions in order to find a probabilistically feasible path to the goal (yellow star). The uncertainty in the state at each node is represented as an uncertainty ellipse. Each state distribution is checked probabilistically against the constraints (cyan). If the probability of constraint violation is too high, the node is discarded (red); otherwise the node is kept (green) and may be used to grow future trajectories.

4.4 Closed-loop prediction using CC-RRT, reference commands are sampled in the configuration space and guidance command is generated using (2.4) to follow the reference command for the closed-loop system.

4.5 Trajectories are propagated using the closed-loop vehicle model as suggested in Figure 4.4 and checked for feasibility.

4.6 Reference path and predicted future trajectory (rather distributions).

4.7 Closed-loop chance constrained RRT with 4 obstacles, all cases.

4.8 Sample tree with $p_{safe} = 0.5$ generated by CC-RRT algorithm for a simple environment. Each node corresponds to the state distribution mean; a $2 - \sigma$ uncertainty ellipse is centered at each node. The mean is shown by ‘x’ around each ellipse.

4.9 Sample tree with $p_{safe} = 0.9$ generated by CC-RRT algorithm for a simple environment. The tree shows forward simulated trajectory of a UAV using (4.45), the reference path is not shown.

4.10 Sample tree after 1 sec with $p_{safe} = 0.9$, reference path (black), predicted trajectory (blue), path chosen from the tree (magenta), tracked path (red), nodes on the tree (magenta). Note that path to the goal location is not found yet.

4.11 Sample tree after following the first path segment, the tree is grown during path following. Colour codes for trajectories remains the same. Multiple paths are found. The path with minimum cost, shown in magenta, is selected for execution.

4.12 Complete reference path (in black) and tracked trajectory (in red) from the starting location to the goal location.
4.13 Sketch of expanded obstacles during constraint checking. The chance constraint formulation expands each obstacle and checks the probabilistic constraint satisfaction. The original obstacle is shown in green and the expanded portions are shown in black and red.

4.14 Sample tree in the dynamic uncertain environment, static obstacles are shown in red. The green obstacle is dynamic and moving towards right at the speed of 7 m/s. The dynamic locations at two places shown in blue.

4.15 The completed reference path (black) and tracked path (red) in the uncertain environment.

5.1 An unknown aircraft enters a secure zone. UAV A1, A2, and A3 are deployed to capture the target.

5.2 Desired formation around the target. UAVs have to fly in formation with the target at relative distance $R_i = r_i - r_i = \xi [\cos \alpha_i \sin \alpha_i]^T$. Angle of formation is $\alpha_i = \frac{2\pi i}{n}$, $i = 0, \ldots, n - 1$.

5.3 The graph of $n$ UAVs and target. The target is the root of the graph, which has a directed path to every UAV in the graph.

5.4 Trajectories of the UAVs while maintaining a constant separation from the target.

5.5 Tracking errors.

5.6 Speed and heading convergence.

5.7 Target centric trajectory in polar coordinates.

5.8 Information exchange topology graph including the target.

5.9 Trajectories of the UAVs while maintaining a constant separation from the target.

5.10 Tracking errors.

5.11 Speed and heading convergence.

5.12 Target centric trajectory in polar coordinates.

5.13 Trajectory of the UAVs while maintaining a constant separation with the target.

5.14 Tracking errors.
5.15 Speed and heading convergence ........................................ 129
5.16 Target centric trajectory in polar coordinates ...................... 129
A.1 The formation geometry .................................................. 140
A.2 Relative X position of each robot .................................... 142
A.3 Relative Y position of each robot .................................... 143
A.4 Trajectories of the robots in the formation .......................... 143
List of Tables

2.1 Initial conditions for straight line path following .......................... 17
2.2 Initial conditions for circular path following ............................... 19
3.1 Set Description ........................................................................ 52
3.2 Simulation results, simple scenarios ......................................... 63
3.3 Simulation results, simple scenarios ......................................... 64
3.4 Simulation results, cluttered environment ................................. 66
4.1 Simulation results, cluttered environment ................................. 104
5.1 Initial conditions ....................................................................... 126
Chapter 1

Introduction

Unmanned Air Vehicles (UAVs) have received considerable attention from research because of their wide range of applications. They have been successfully applied to many challenging civilian and defence applications such as surveillance, search, rescue, border and coast patrol, fire monitoring, etc. They are particularly useful for dull and dirty operations without exposing humans to dangerous situations. With rapid advances in electronics, sensors and communications, it is relatively easy to manufacture inexpensive and easy to operate UAVs, and hence they are expected to be widely deployed. The success of a UAV mission heavily depends on the algorithms employed. To increase the reliability of UAVs, there is a need to develop state-of-the-art-algorithms which enhance their mission success rate.

A fundamental requirement of an autonomous vehicle is to travel from one location to another. And in order to travel between different locations, a vehicle has to track the path quite accurately. Path following techniques are important for any mission because the success of the mission heavily depends on how close a given path is being tracked, especially in cluttered environments. In this thesis, we first address the problem of path following in cluttered and/or windy environments. By combining a pursuit guidance law with a line-of-sight guidance law, we propose a novel guidance law to follow standard manoeuvres quite accurately.

Another important requirement of a mission is to plan its path whilst satisfying practical mission constraints. The path/motion planning problem is an important area of interest in the field of UAVs and robotics with requirements on optimality, completeness and computational complexity. A number of path planning algorithms have been developed in these fields such as
road map, cell decomposition, potential field, etc [56], [19], [58]. Generally, motion planning is a purely geometric problem: given the geometry of a UAV and static obstacles, compute a collision-free path of the UAV between two given locations. This statement however ignores several key aspects of the real physical world. In particular, a UAV’s motion is often subject to kinematic and dynamic constraints (kinodynamic constraints) that cannot be ignored and are important in obstacle rich environments. Unlike obstacles avoidance, such constraints cannot be represented as forbidden regions in the configuration space. RRTs have been recently shown to be effective as sampling-based approaches to identify feasible trajectories quickly in complex motion planning problems [14], [28], [38], [96], [55], [60], [70]. Chapter 3 considers real-time motion planning problems for single and multi-UAV systems. Augmenting some heuristics with rapidly-exploring random trees and using a time window approach, we propose a real-time planner for a single UAV system to effectively compute paths in complex environments [49]. Then, we develop a multi-UAV path planner by embedding a single UAV path planner in a framework that manages the interactions between UAVs and use a conflict resolution strategy to resolve conflicts [51]. The multi-UAV path planner allows each UAV to operate in a decentralized manner when not within the communication range and provides conflict free paths when within the communication range.

While operating in uncertain (stochastic) environments, it may not be possible to identify a feasible path because of the trade-off between planner conservatism and the risk of infeasibility [81]. An important approach to trade-off risk and performance is to limit the probability of failure (chance constraint) and to maximize the performance under this constraint [81], [87], [65]. Chapter 4 describes a chance constraint formulation using incremental sampling-based methods, and in particular rapidly-exploring random trees (RRT) for linear systems subject to process noise and/or uncertain dynamic obstacles [68].

Recently, there has been an increasing interest in the control of multi-agent networks due to their relevance in areas such as rendezvous control of multi nonholonomic agents [24], flocking attitude alignment [20], and formation control [79], [90], [88]. Formation control which coordinates the motion of relatively simple and inexpensive multiple UAVs is one of the essential technologies that enables UAVs to cover large operational areas and achieve complex tasks. Coordination control strategies that achieve a desired formation around a target vehicle (object) by multiple mobile vehicles using neighbour information have been useful in many practical appli-
cations such as investigations in hazardous environments, mobile sensor networks and security systems [84], [46]. Chapter 5 describes such a problem in which an unknown aircraft enters a secure zone that could be hostile. The task is to either escort the aircraft away until it lands or to restrict its movement in a desired manner in order to assure safety and security [99].

The tracking of objects using distributed multiple sensors is an important field of work in the application areas of UAVs, military applications and mobile systems. It is often important to be able to track a common variable without employing a centralized strategy, since such a strategy is vulnerable to node failures. A prominent example is distributed tracking of a moving target using a wireless sensor network (WSN). In this case, sensors have to cooperate in order to accomplish accurate information of, e.g., target position, velocity, clock, etc. WSN is used in many diverse fields and facilitates Network Enabled Capability (NEC) [77]. If platforms such as warships and airplanes are networked together and their data is shared, then they will be able to compile a more accurate picture of their environment than with just data from their own sensors. Appendix A considers a problem of target tracking in a specific formation in the presence of sensor faults and bad weather conditions. Real-time reconfiguration of networked control systems to track a target is demonstrated on a set of robots for a target tracking application. The experiment shows that if the robots are networked they can track the target robustly even in the presence of sensor faults and/or degraded sensor performance due to bad weather conditions [86].

The thesis introduces novel solutions to single and multiple UAV systems for various applications. Path following is considered in Chapter 2 and the approach is used for tracking of generated paths by path planning algorithms presented in later chapters (Chapters 3 and 4). Path planning is discussed in Chapters 3 and 4 while target-capturing is discussed in Chapter 5. Chapter 3 considers real-time path planning problems in obstacle rich environments and Chapter 4 considers real-time path planning problem in uncertain environments. The complexity of the problem arises because of the large number of obstacles and UAV motion constraints and/or uncertainty in the environment. Rapidly-exploring random trees (RRTs) have received increased attention recently as a sampling based approach to identify quickly feasible trajectories in complex motion planning problem [57], [59], [52], [61]. Even though several approaches have been proposed to solve general path planning problems [56], [19], [58], approaches that use randomization or incremental sampling have demonstrated several advantages for real-time
path planning problems. The RRT based algorithms are developed for two situations. The first situation considers a single UAV system in a partially known environment and the path planning task is to find close to optimal solutions in real-time while avoiding static and pop-up obstacles [49]. The second situation considers a multi-UAV system and the task is to generate de-conflicting trajectories while avoiding static and dynamic obstacles [51]. As is known, real-world systems can be exposed to significant levels of stochastic disturbance [81]. Stochastic systems typically have a risk of failure due to unexpected events, such as unpredictable wind. To stay away from failure a UAV needs to stay away from the failure states, such as flying near an obstacle (otherwise the unpredictable wind may take the UAV into the obstacle and/or building). Chapter 4 considers path planning in such uncertain environments [68].

In chapter 5 and Appendix A of the thesis, the emphasis is on coordination and cooperation. Chapter 5 describes a novel approach combining a robust controller with the consensus algorithm in order to achieve a target centred formation when information is only partially available to the capturing UAVs. Numerical results are presented to validate the proposed approach [99]. Appendix A considers a sensor network problem in which the objective is to track a target in a specified formation even in the presence of sensor faults and/or with degraded sensor performance due to bad weather conditions [86]. Using consensus theory an algorithm is developed to track the target robustly and is tested on a set of robots. Although the work has been focused on UAVs, we do not have access to such a system and the robotics set up is a good representation in two dimensions.

1.1 Contributions

- Chapter 2: This chapter develops a new guidance law by combining pursuit-type and line-of-sight guidance laws for tracking straight paths, circular paths and combination of both. Stability analysis of closed loop error dynamics is presented along with the conditions on the gains of the guidance law. Performance of the guidance law is verified by simulation under wind disturbances.

- Chapter 3: This chapter initially develops a real-time robust path planning algorithm using rapidly-exploring random trees augmented with some heuristics for a single UAV system. Heuristics are proposed to improve the performance of the path planner. The turn radius
constraint of the UAV is incorporated in the path planner by converting it into a geometric constraint and parameterizing waypaths in terms of straight lines and circular arcs of radii greater than the minimum turn radius of the UAV. The path planner effectively avoids pop-up obstacles and searches for less costly paths while tracking a path. The performance of the path planner is compared with the visibility line (VL) method to show that the proposed approach can provide close to optimal solutions in real-time. Then, we develop a multi-UAV path planner by embedding the single UAV path planner in a framework that manages interaction among UAVs and uses a coordinate strategy to resolve conflicts.

- Chapter 4: In this chapter, we consider uncertainties in the system model and environment and extend the path planning algorithm to provide robust probabilistic paths. We use a chance constraint to limit the probability of constraint violation below a pre-defined value, and to trade-off between planner conservatism and the risk of infeasibility. We predict future distributions to grow the tree using the closed-loop vehicle model and ensure that the predicted path satisfies the probabilistic feasibility condition. Numerical results are presented to illustrate the performance of the path planner.

- Chapter 5: In this chapter, we present a target capturing problem using formation control. The new algorithm embeds a consensus algorithm into the controller to form a target centred formation. Theoretical and numerical results are presented to show the effectiveness of the algorithm.

- Appendix A: We present here the experimental results of a seedcorn project “Novel experiments of real-time reconfiguration of networked systems on collaborative missions in uncertain environments”, supported by the EPSRC/BAE Systems funded project NECTISE - Network Enabled Capability Through Innovative Systems.

1.2 Organization of the thesis

A summary of the thesis is as follows. Chapter 2 introduces a novel guidance scheme by combining a pursuit guidance law with a line-of-sight guidance law to track piecewise paths accurately in clutter and/or windy environments. Chapter 3 briefly reviews some of the key existing approaches for path planning and then a path planning algorithm is developed using RRTs
for single and multi-UAV systems in obstacle rich environments. Chapter 4 outlines problems associated with path planning when exposed to real-world practicalities and then introduces a stochastic approach for path planning in uncertain environments. Chapter 5 advocates the necessity of cooperation and coordination in formation control. A practical application of target capturing is addressed in this chapter. The final chapter summarizes the contributions of the thesis and outlines avenues for future development. Appendix A describes experimental results of a robust target scheme and shows that by cooperation multi agent systems can track a target more robustly even with degraded performance of sensors.
Chapter 2

Path following in Cluttered Environments under Windy Conditions

This chapter considers UAV path following in cluttered environments under windy conditions. Owing to unstructured wind patterns in cluttered environments, path following can be difficult resulting in high errors and possibly collisions with buildings. Combining a pursuit guidance law philosophy with a line-of-sight guidance law, we develop a novel guidance law that has low computational complexity and can track straight line paths, circular paths, a combination of both and waypaths accurately in the presence of wind blowing as high as fifty percent of the UAV’s air speed. Performance of the guidance law is demonstrated through numerical simulations.

2.1 Introduction

Recently, unmanned aerial vehicles (UAVs) have been successfully deployed in several applications such as search and rescue, surveillance, urban highway traffic monitoring, etc. One of the main objectives of such missions is to generate paths and follow them accurately in order to accomplish the assigned tasks. A path is usually described in terms of a set of waypoints that a UAV has to visit sequentially. UAV path following in cluttered/urban environments is a challenging task because of unstructured and high magnitude wind patterns. In this chapter we address UAV path following under such conditions.

UAV path following can be achieved in two ways. The first way is by designing a separate guidance loop based on geometric and kinematic properties and assuming that an inner loop
controller is fast enough to generate the commanded acceleration. As simple and well established methods are available for inner loop control design, the separate inner and outer loop approach is preferred in real-world applications. An alternative is to use an integrated approach to design the inner and outer loop simultaneously. Receding horizon, Dynamic Inversion, and Optimization based approaches are, to name a few, the examples of these. In this work, we consider the separate design structure and develop the guidance law based on geometric and kinematic properties.

Classical guidance schemes like pursuit and PN guidance laws generate acceleration commands based on problem specific engagement geometry [115]. In those problems, the emphasis is on interception rather than following a desired path. To some extent line-of-sight guidance law captures the essence of path following as the commanded lateral acceleration is a function of LOS separation (the minimum distance between the vehicle and the desired trajectory) but it does not strictly achieve it [115]. Hence, there is need to develop a guidance law which can keep a UAV as closely as possible on the desired trajectory for most of the time, especially in cluttered environments.

There is a wide body of literature on developing guidance laws for UAVs to follow straight lines, curved trajectories and to execute loiter manoeuvres. Shehab and Rodrigues [100] developed a piecewise-affine control law with Lyapunov based controller for UAVs to follow a path. Breivik and Fossen [13] presented guidance laws for path following in 2D and 3D using control Lyapunov functions. The above approaches do not take wind into account. Kaminer et al. [41] developed an optimal control law for multiple UAV trajectory following with coordination. The authors parameterize the trajectory as a function of time to follow, which becomes an issue when the UAV is initially located away from the path. Cao et al. [16] developed an $L_1$ adaptive controller for UAV path following taking wind disturbances into account. A simple and effective path following controller is developed by Nelson et al. [75] using vector fields. Vector fields are developed for both straight line and circular paths in the presence of winds. In vector field based path following, a boundary along the path is designed to allow the UAV to steer along the path. Rysdyk [95] developed guidance laws to follow straight line and circular orbits based on helmsman behaviour. The guidance law changes the course of the vehicle based on the wind magnitude and direction which are estimated on the fly using a wind estimator. Lawrence et al. [63] developed a generalized vector field method with Lyapunov stability for
different manoeuvres. Park et al.[82] present a non-linear guidance law that generates a reference point on the path and uses this reference point as a target to manoeuvre the aircraft along the path. The authors present theoretical results and show the performance through experiments under wind disturbances. Recently, Tsourdos et al. [110] presented a linear “Carrot” guidance and nonlinear dynamic inversion based guidance laws for UAV path following. The nonlinear guidance law presented therein improves the path following performance in the case of curved Dubin’s segments.

In this work, combining a pursuit guidance law philosophy with line-of-sight (LOS) guidance laws we develop a guidance law for UAV path following in cluttered/urban environments with high magnitude wind patterns. The LOS guidance controls the UAV position error with respect to the desired path and the pure pursuit component controls its heading error with respect to the next waypoint. So a combination tries to achieve both the objectives simultaneously. The resultant guidance law is capable of following standard manoeuvres in the form of straight lines, circular arcs and a combination of both. The guidance law performs well under wind speeds upto fifty percent of the air speed of a UAV.

The rest of the chapter is organized as follows. Section 2.2 describes the path following problem and navigation model of a UAV. In Section 2.3, we propose a new guidance law for path following and derive stability conditions. In Section 2.4, the performance of the guidance algorithm is demonstrated in the presence of wind for various path following geometries including urban terrain. Concluding remarks are presented in Section 2.5.

2.2 Problem Statement

The UAV has to navigate along a path which is generated by the path planning algorithms in the two dimensional configuration space. In order to track a given path, a guidance law is required which allows a UAV to follow the path with minimum deviation. To design a guidance law, we need a navigation model of the UAV. For this purpose, we consider the following navigation
model

\[
\begin{align*}
\dot{x} &= v_a \cos \psi \\
\dot{y} &= v_a \sin \psi \\
\dot{\psi} &= \frac{a}{v_a}
\end{align*}
\]  

(2.1) (2.2) (2.3)

where, \( v_a \) represents the speed of the UAV, \( \psi \) is the flight path angle and \( a \) represents the lateral acceleration which acts as a guidance parameter.

Consider a UAV path following geometry as shown in Figure 2.1. The UAV has position error of \( d \) and has flight path angle error of \( \tilde{\psi} \triangleq (\psi_r - \psi) \) with respect to the desired path, where \( \psi_r \) represent the angle of the desired path with respect to the reference. To navigate along the given path, the UAV has to keep adjusting its flight path angle. The flight path angle adjustment is achieved by applying lateral acceleration which acts normal to the velocity vector. The path following problem is to compute lateral acceleration \( a \) for the UAV to keep the errors in position \( d \) and heading \( \tilde{\psi} \) small in the presence of changing desired path geometries and with wind.

\subsection{2.3 Combined pursuit plus line-of-sight guidance law}

Contrary to missile guidance applications, the UAV path following problem has to maintain position error and flight path angle error close to zero for all time and not just at the termination of flight. For this purpose, we employ a guidance law that generates lateral acceleration as a weighted sum of position and flight path angle errors as follows

\[ a = k_1 (\psi_r - \psi) + k_2 d \]  

(2.4)

Figure 2.1: Formulation Geometry
where, \( k_1 > 0 \) and \( k_2 > 0 \) are the design parameters. In the next section, we will define the conditions on them. The UAV path following essentially involves controlling the UAV position and heading error with respect to a desired path. The LOS guidance component controls the UAV position error with respect to the desired path and pure pursuit component controls its heading error with respect to the next waypoint. So a combination tries to achieve both these objectives simultaneously. Next, we present stability analysis of the error dynamics of the path following problem.

### 2.3.1 Error dynamics: Stability conditions

In this section, we carry out stability analysis of the closed-loop error dynamics of the path following formulation. Toward this direction, we first write the UAV velocity error and flight path angle error as follows

\[
d = v_a \sin(\psi_r - \psi) \tag{2.5}
\]
\[
\dot{\psi} = -\frac{k_1}{v_a} (\psi_r - \psi) - \frac{k_2}{v_a} d \tag{2.6}
\]

For small angles, we assume \( \sin(\psi_r - \psi) \approx \psi_r - \psi \) and (2.5)-(2.6) can be rewritten as

\[
\begin{bmatrix}
\dot{d} \\
\dot{\psi}
\end{bmatrix} = \begin{bmatrix}
0 & v_a \\
-\frac{k_2}{v_a} & -\frac{k_1}{v_a}
\end{bmatrix} \begin{bmatrix} d \\ \dot{\psi} \end{bmatrix} \tag{2.7}
\]

The stability of the system in (2.7) can be checked by finding the eigenvalues as,

\[
eig \left( \begin{bmatrix}
0 & v_a \\
-\frac{k_2}{v_a} & -\frac{k_1}{v_a}
\end{bmatrix} \right) = -\frac{k_1}{2v_a} \left( 1 \pm \sqrt{1 - 4 \frac{k_2^2 v_a^2}{k_1^2}} \right) \tag{2.8}
\]

The closed loop system will be stable if

\[
k_1 > 0, \tag{2.9}
\]
\[
k_2 > 0. \tag{2.10}
\]

Choosing positive gains we can ensure that the position error and flight path angle error stay small, and hence, the UAV stays on the desired trajectory.
2.3.2 Characteristic of the guidance law

Consider a straight line path following geometry as shown in Figure 2.2. When the reference path is on the left of the UAV, the distance error will be positive. However, flight angle error can be positive, negative or zero. In the absence of the flight path angle error, lateral acceleration would turn the velocity vector in the anticlockwise direction and it keeps rotating the velocity vector until the position error becomes negative. When the position error becomes negative, the velocity vector turns clockwise and it keeps rotating clockwise until the position error becomes positive. In this manner, the UAV will fluctuate left to right and right to left without staying on the desired path. In order to control this behaviour, we introduce flight path angle error term in the guidance law. It controls (introduces damping) rotation of velocity vector, hence fluctuation around reference line can be controlled. To gain further insight with small angle approximation and assuming no inner loop dynamics, we write \( \dot{d} = v_{a} \sin(\psi_{r} - \psi) \simeq v(\psi_{r} - \psi) \) and \( a \simeq -\ddot{d} \).

The equation (2.4) can then be rewritten as

\[
\ddot{d} + k_{1} \frac{\dot{d}}{v_{a}} + k_{2} \dot{d} = 0.
\]  

Equation (2.11) turns out to be a second order system with \( w_{n} = \sqrt{k_{2}} \) and \( \zeta = \frac{k_{1}}{2\sqrt{k_{2}k_{1}}} \). This analysis provides a better way to choose gain \( k_{1} \) and \( k_{2} \). In general, small \( k_{2} \) and relatively high \( k_{1} \) are preferred.

2.3.3 Straight-line following

Consider a straight line following geometry as shown in Figure 2.2. The UAV has to travel along the line from waypoint \( W1 \) to waypoint \( W2 \) with minimum deviation to avoid any possible collisions with obstacles. Let \( R \) be the displacement of the UAV with respect to point \( W1 \) making an angle \( \psi_{c} \) with the reference \( OX \). The UAV distance and flight path angle errors can be calculated as

\[
d = R \sin(\psi_{r} - \psi_{c}) \quad \text{(2.12)}
\]

\[
\tilde{\psi} = \psi_{r} - \psi \quad \text{(2.13)}
\]
where, $\psi_r$ is the angle which the desired path makes with the reference. In this case, we assume the x-axis is the reference line.

![Diagram of straight-line path geometry](image)

**Figure 2.2:** Straight-line path geometry

### 2.3.4 Circular path following

Consider a circular path of radius $R_c$ with centre at $(x_c, y_c)$ as shown in Figure 2.3. Let $R'$ be the displacement of the UAV with respect to the centre of the circular path. The UAV position and flight path error can be calculated as

\[
d = R' - R_c, \tag{2.14}
\]

\[
\tilde{\psi} = \psi_r - \psi, \tag{2.15}
\]

where, $R' = \sqrt{(x - x_c)^2 + (y - y_c)^2}$ and $\psi_r = \arctan(y_c - y, xc - x) \pm \arctan(R_c, \sqrt{R'^2 - R_c^2})$.

Note that any circle has tangents from the points $(x_g, y_g)$ which satisfies

\[
R_g = \sqrt{(x_g - xc)^2 + (y_g - y_c)^2} > R_c.
\]

As per mission requirement, one can choose to move either clockwise or anticlockwise by adding or subtracting $\arctan(R_c, \sqrt{R'^2 - R_c^2})$ angle, which is the angle between the tangent and the line connecting the centre of circle and the UAV’s current position.

### 2.3.5 Effect of wind

The lateral acceleration generated by a UAV is a function of its airspeed. However, motion of a UAV with respect to the ground or a fixed desired path is affected by the wind speed.
In the presence of wind, the navigation model of the UAV can be written as

\[
\begin{align*}
\dot{x} &= v_a \cos \psi + v_w \cos \psi_w \\
\dot{y} &= v_a \sin \psi + v_w \sin \psi_w \\
\dot{\psi} &= \frac{a}{v_a}
\end{align*}
\]  

(2.16) (2.17) (2.18)

where, \( v_w \) is the wind speed and \( \psi_w \) the direction vector of the wind. The inertial velocity and flight path angle can be computed as

\[
\begin{align*}
\nu_{init} &= \sqrt{v_a^2 + v_w^2 + 2v_a v_w \cos \psi \cos \psi_w + 2v_a v_w \sin \psi \sin \psi_w} \\
\psi_{init} &= \arctan\left(\frac{v_a \sin \psi + v_w \sin \psi_w}{v_a \cos \psi + v_w \cos \psi_w}\right)
\end{align*}
\]

(2.19) (2.20)

Note that in the presence of wind, the rate of change in distance error is now a function of both airspeed and the given wind speed as follows

\[
\dot{d} = v_a \sin(\psi_r - \psi) + v_w \sin(\psi_r - \psi_w)
\]

(2.21)

As wind works as an external input or disturbance, the error cannot be driven to zero with the guidance law as in (2.4). In order to stay on the desired path, we modify the guidance law using ground flight angle as follows

\[
a = k_1 (\psi_r - \psi_{init}) + k_2 d
\]

(2.22)
Note that changing $\psi$ to $\psi_{init}$ does not effect the stability conditions. We can write rate of change in inertial position error as $\dot{d} = v_{init} \sin(\psi_r - \psi_{init})$, and using this quantity with the guidance law as in (2.22) and deriving the stability conditions of the closed loop error dynamics, we get the same conditions on gains $k_1$ and $k_2$. Note that here we assume that the UAV is capable of computing inertial velocity and heading from onboard processor/sensors.

2.3.6 Waypath following in an urban environment

Path planning is an important area of research in the field of UAVs and robotics. A path is usually described in terms of a set of waypoints that a UAV has to visit sequentially. The path planner then generates paths assuming that a guidance law is capable of tracking the generated paths. This assumption holds good when planning is performed in environments that contain few obstacles and have large feasible regions. But these assumptions do not hold in typical cluttered environments where passages are narrow. Moreover, it is difficult to find smooth paths because of space constraints. Hence, in such environments path following needs more attention; otherwise even small tracking errors can cause to crash a UAV into a building, especially in the presence of wind.

Consider a scenario as shown in Figure 2.4. A UAV has to travel through waypoints $W1 \rightarrow W2 \rightarrow W3$ sequentially as a part of a mission (because of space constraints). If the UAV takes a turn after reaching waypoint $W2$, it may overshoot and take some time to converge back to the desired path irrespective of guidance law being used. This behaviour may endanger the mission.
In order to avoid this problem, we parameterize the waypath $W_1 - W_2 - W_3$ in terms of straight lines and a circular arc of radius $R > R_{\text{min}}$ as shown in Figure 2.5. Now the UAV starts taking the turn at point $p_1$ and follows the arc until it reaches point $p_2$. The distance $dis$ of points $p_1$ and $p_2$ from waypoint $W_2$ is a function of the arc radius and the angle between the waypath as follows

$$dis = \frac{R}{\tan \beta}$$

(2.23)

where, $2 \beta$ is the angle between the waypaths formed by joining waypoint $W_1$ to $W_2$ and $W_2$ to $W_3$.

There could be other parameterizations for smooth transition from one waypoint to another, like k-trajectory [2]. We choose this particular one because it is simple and yet effective. Moreover, it requires only one turn between two waypaths. While following an arc we do not use the LOS term in the guidance law as the UAV has to turn and there is no LOS for arc following. The pursuit term will keep the UAV along the arc. Note that this idea avoids computation of finding the center of the arc and associated terms as presented in Section 2.3.4, and hence it will reduce the computation burden on the processor, which can be considered as an added advantage.

2.4 Numerical results

Now we demonstrate the performance of the guidance law based on the following example scenarios. The first three examples show the path following capability of the guidance law to follow standard manoeuvres like straight lines, circular paths and a combination of both. Next, we generate an obstacle rich environment to mimic cluttered/urban terrain and show that the proposed law keeps the UAV on the waypath most of the time in the presence of wind except during transition from one waypath segment to another.

2.4.1 Straight line following

In this section, the objective is to check the straight line following capabilities under different initial conditions and with constant wind. We assume the UAV speed to be $13 \text{ m}/\text{s}$ and the minimum turning radius $R_{\text{min}}$ of the UAV is limited to three times the velocity of the UAV,
that is, 39 m. Therefore, the maximum lateral acceleration is \( \frac{v^2}{R_{\text{min}}} = 4.33 \text{ m/s}^2 \). The design parameters used for simulation are \( k_1 = 15 \), and \( k_2 = 1 \). The UAV follows the waypath formed by joining a straight line between points \( W_1 \) and \( W_2 \). Initially we consider five different random initial conditions for simulations, which are tabulated in Table 2.1. Figure 2.6 shows trajectories of the UAV under no winds and the associated guidance command and position errors are shown in the Figures 2.7 and 2.8 respectively. From Figure 2.6 and the associated initial flight path angle from Table 2.1, it can be observed that the UAV takes some time to converge if there is a large flight path angle error. This is because the UAV has to turn along the waypath and there are bounds on lateral acceleration, which need to be taken into account as the UAV is subjected to a turn radius constraint. On the other hand, for low flight path angle error, the UAV converges to the reference path quickly in spite of a comparatively high position error. In this case, the UAV has only to correct position error and this is achieved by slightly turning and aligning the UAV towards the waypath (Case 3 in Figure 2.6 is an example of this).

Next, we introduce wind with a speed of 4 m/sec blowing at an angle of 23\(^\circ\), that is the...
wind is traveling from southwest to northeast. In each case, the UAV trajectory converges to the reference path as shown in Figure 2.9. The associated guidance command and position errors are shown in Figures 2.10 and 2.11, respectively. However, contrary to the no wind condition, the guidance command demand increases as the UAV acts to nullify the effect of the wind. It can be seen from Figure 2.11 that for each case the position error goes to zero but the settling time (to reach and stay on the reference line) depends both on the initial conditions and on the wind speed. Before we discuss the reason for this, note that in the inertial frame the turn radius would not be the same as under no wind condition. This is because in the inertial frame \( R_{\text{min}} = \frac{V_{\text{init}}^2}{a} \leq \frac{v_a^2}{a} \), and hence the turn radius which a UAV can follow in the wind will be higher than under no wind condition. Because of this fact, the UAV takes a turn of higher radius as compared to the no wind conditions, which results in a relatively larger error (see Figure 2.11) for large initial flight angle errors.

![Figure 2.8: Distance errors under no wind for five different cases](image)

![Figure 2.9: Trajectories of the UAV under wind for five different cases](image)

### 2.4.2 Circular path following

We now test the validation of the guidance law for circular path following. The radius of the circle is 50 m and the centre is located at (0, 0). The UAV velocity is kept unchanged while the design parameters are changed to \( k_1 = 15 \), \( k_2 = 0.1 \). All other assumptions remain the same. In this exercise, we consider four different initial conditions, which are tabulated in Table 2.2. Initially, we assume there is no wind and performed the simulations. Figure 2.12 depicts the UAV trajectories for these initial conditions. It can be noted that in each case the UAV follows
the circular path accurately. The corresponding lateral acceleration commands are shown in Figure 2.13. Note that in Cases 1 and 3, and also in Cases 2 and 4, the lateral acceleration commands are the same, and therefore sit on top of each other. This is because they have the same position errors, which can be observed from Figure 2.14.

Table 2.2: Initial conditions for circular path following

<table>
<thead>
<tr>
<th>Case no</th>
<th>x position</th>
<th>y position</th>
<th>initial flight path angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>-100</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-100</td>
<td>-100</td>
<td>π</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>-100</td>
<td>π</td>
</tr>
</tbody>
</table>

Next, we again consider a constant wind of speed 4 m/sec blowing at an angle of 23°. The UAV trajectory under influence of wind is shown in Figure 2.15 for four different initial conditions, which are the same as for the no wind situations. The corresponding lateral acceleration commands and position errors are shown in Figures 2.16 and 2.17, respectively. It can be seen from Figure 2.15 that the UAV tracks the circular path accurately even in the presence of wind. The reason for tracking the path accurately is that the guidance scheme takes into account the inertial flight path angle error instead of the aerial one. This allows the UAV to compensate for the gain in flight path angle due to the wind. Moreover, it can be observed from Figure 2.16, that the lateral acceleration commands remain within bounds most of the time because of the usage of inertial flight path angle error along with the position error. Had we not used the
inertial flight path angle, the position and flight angle errors would have been larger and there would have been large acceleration demands to compensate for these errors. From Figure 2.17, it can be seen that the position error remains near zero except in the initially when the UAV is not yet on the circular orbit.

2.4.3 A combination of straight line and circular arc following

As mentioned before the applicability of the proposed guidance law is not limited to urban terrain, and it also can be applied in general path following scenarios. In order to illustrate this, the performance of the guidance law is checked during switching manoeuvres from one type
to another. In this section, we consider a simple scenario with a straight line followed by a loiter and then another straight line manoeuvre. Figure 2.18 shows the UAV trajectory for such a scenario under no wind. Initially the UAV follows a straight line from point (0, 0) to (0, 300), then it switches to loiter manoeuvre in a clockwise direction of radius 100 m until it reaches point (100, 300). Again from (100, 300) it follows a straight line till the point (400, 400). It can be observed from Figure 2.18 that the UAV follows the path reasonably well except in the transition region. Next, we validate the applicability of the guidance law for such a scenario in the presence of wind. The wind magnitude and direction are assumed to be the same as used in the previous case. Figure 2.19 shows the UAV trajectory under the influence of the wind. From Figures 2.18 and 2.19, it can be noted that the performance is almost similar except in the transition period where in windy conditions the UAV takes some time to achieve convergence. As mentioned before, it is because the minimum turn radius capability does not stay the same in the inertial frame for windy conditions.

### 2.4.4 Waypath following

After illustrating the path following ability for different manoeuvres, in this section we verify the performance of the guidance law in an urban terrain. We assume that the reference path in the urban environment is generated using a suitable path planner. We also assume that the path planner yields piecewise linear paths. The objective is to follow the reference path with minimum deviation and to smoothly transit to another waypath so that tracking errors due to
transition do not force the UAV into buildings. As shown in Figure 2.4, if the UAV starts taking a turn after reaching the waypoint W2, the UAV may have to oscillate before it settles. For sharp turns, deviations due to transition will be more with increasing possibility of collision with nearby buildings. In order to avoid this from occurring, we adopt the strategy as proposed in Section 2.3.6 and test the guidance law in various scenarios. For illustration purposes, we create an artificial scenario as shown in Figure 2.21 (to mimic an urban terrain) with the reference path as shown in black. The UAV follows the path using the strategy proposed in Section 2.3.6. And the trajectory of the UAV is shown in red. The UAV follows the path quite accurately even in the presence of wind. It can be noted from the figure that the transitions are reasonably smooth. Figure 2.20 shows path following capability for a rectangular manoeuvre under no wind. From both example scenarios, it can be concluded that the guidance law keeps tight bounds on position error while following the waypath.

2.5 Conclusions

In this chapter, combining a pursuit guidance law philosophy with a Line-of-sight guidance law, we have developed a novel guidance law for path following in cluttered environments under wind conditions. The guidance law has the capability to follow different types of paths in the presence of wind. Stability conditions of the closed loop error dynamics are provided under small angle assumptions. The performance of the guidance algorithm is illustrated by performing several simulations.
Figure 2.20: Waypath following under no wind

Figure 2.21: Waypath following in urban terrain under wind influence $v_w = 4$ m/sec at $\phi_w = 23^\circ$
Chapter 3

Robust Real-time Path Planning Algorithms using Rapidly-exploring Random Trees

This chapter presents robust real-time path planning algorithms in obstacle rich environments using rapidly-exploring random trees (RRTs) for single and multiple UAV systems. We use several heuristics including optimization and exploration to improve the performance of the algorithm. Apart from heuristics, a time window approach, similar to model predictive control (MPC), combined with a greedy approach is used to explore the environment quickly and to generate paths in real-time is proposed. The tree is allowed to grow for a given time window and a path in the tree is chosen to move towards a goal position using a heuristic. While following a segment of the path, the tree is further expanded with emphasis on an optimization heuristic searching for low cost paths. This strategy facilitates in finding close to optimal solutions. Then, we develop a multi-UAV path planner by embedding a single UAV path planner in a framework that manages interaction among UAVs and uses a conflict resolution strategy to resolve conflicts. The multi-UAV path planner allows each vehicle to operate in a decentralized manner when it is not within the communication range and provides conflict free paths when UAVs are within the communication range. The performance of the path planner is demonstrated through various simulations.
3.1 Introduction

Unmanned aerial vehicles (UAVs) have recently been deployed in a variety of search and surveillance missions. One of the main objectives in such a mission is to generate de-conflicting (i.e., collision free) paths for the UAVs to follow. A path is usually described in terms of a set of waypoints that the UAV has to visit sequentially. The path planning problem is therefore concerned with generating de-conflicting paths in terms of waypoints taking various environmental and physical constraints of the UAV into account. Designing a path planner for multiple UAVs in obstacle rich environments that contain static and moving obstacles is challenging. The difficulty further increases when a solution is sought in real-time while accounting for motion constraints of the UAVs.

For these challenging UAV applications, the computational complexity of the path planning algorithm is an important issue that needs to be addressed. Since UAVs typically operate at high speeds, the path planner has to function quickly. When unknown (pop-up and dynamic) obstacles are detected, a new path (or paths) needs to be generated in real-time from the current location to avoid collisions. If the path planner fails to generate a feasible solution within a given time window, the UAVs may collide with the obstacles leading to mission failure.

The path planning problem is important in the fields of UAVs and robotics with requirements on optimality, completeness and computational complexity. Generally, a path planning problem is solved by dividing it into two subproblems. The first subproblem is to find a path or paths from a starting location to a goal location off-line based on a priori information, this phase is usually termed global path planning. Global path planning requires complete knowledge about the environment and an obstacle map. A number of path planning algorithms have been developed to find paths in such a setting such as road map, cell decomposition and potential field, [56], [58] to name a few. The second subproblem is known as local path planning and is executed during flight. The principle idea is detect obstacles which are on the flight path using onboard sensors and avoid them by finding collision-free paths around the detected obstacle. The solution of the local path planning problem heavily depends on the operational environment and available computational time. Several path planning algorithms have been proposed in the literature for both global path planning and local path planning. However, few of these try to solve the planning problem in its full generality [56] and, depending on the nature of
the problem, some techniques work better than others. An extensive introduction to the path planning problem and existing solutions may be found in [19],[56].

In road map approaches, curve or line segments that connect a vertex of one obstacle to a vertex of another without entering the interior of any polygonal obstacles are searched. A continuous path is then chosen, if one exists, from the initial point to the goal point through these vertices. There are various types of road maps, including the visibility graph, the Voronoi diagram and the freeway net. One of the earliest path planning methods was the visibility graph (VG) method which has been widely used to implement path planners for mobile robots. The principal idea of the VG algorithm is to construct a path as a set of polygonal lines connecting the initial position \((x_{\text{init}})\) to the goal position \((x_{\text{goal}})\) through vertices of the obstacles. This set of polygonal lines is called the VG. It has been proved that VGs are guaranteed to compute the “shortest” path. Any shortest path (there may be more than one) between \((x_{\text{init}})\) and \((x_{\text{goal}})\) that avoids the set of disjoint polygonal obstacles is a polygonal path whose inner vertices are vertices of the set VG [56].

In cell decomposition [56], the state space is broken up into discrete cells, and planning is performed between these cells. The task of the planning algorithm is to find a finite sequence of actions that transforms the initial state \(x_I\) to some state \(x_G\) via cells. Obstacles are avoided by not visiting cells which are marked as obstacles. Basic graph search techniques, such as Dijkstra’s algorithm [23] and A* [36], are often used to find a minimum cost path. With a finite, well-defined search space, it is often possible to find an optimal path. However, as the size of the planning problem increases, it becomes increasingly difficult to find the optimal path in a “short” time. Moreover, it often becomes difficult to find a feasible path that accounts for motion constraints in these problems without significantly increasing the size of the search space and search time. When planning in wide open areas, motion constraints might not be important and a feasible path can be found relatively easily. However, when trying to plan a path for a UAV in complex and compact environments, it is difficult to discretize the environment in such a way that a path can be planned without considering constraints on the motion.

The potential field algorithm models autonomous vehicles (AV) as particles moving under the influence of a potential field that is determined by the set of obstacles and the target destination [56]. This method is usually very efficient because at any instant the motion of the
AV is determined by the potential force at the AV’s location. Thus, only information of direct relevance to the AV’s motion is computed and no computational power is wasted. Moreover, since potential fields are additive, introducing a new obstacle is easy because its field can be simply summed to the existing one. However, this artificial potential field approach has a major problem: The AV can be easily trapped at a local minimum before reaching its goal and it may not be possible to avoid obstacles completely.

The performance of motion planning algorithms is usually evaluated in terms of completeness and computational complexity. An algorithm is considered complete if it returns a valid solution to the motion planning problem when one exists and returns a “no solution” outcome if no solution exists. The computational time of deterministic and complete algorithms grows exponentially with the dimension of the configuration space and increases polynomially with the number of obstacles (geometric complexity). These deterministic algorithms do not scale well and therefore cannot be employed for real-time UAV path planning problems, especially those problems that contain many static and moving obstacles.

One solution to the scalability problem is to use a sampling-based algorithm [58] whose computation time does not depend directly on the dimension of the complex environment. In sampling-based path planning, a continuous map is converted into a search graph by random sampling of the environment. Rather than performing an explicit analysis of the whole space, sampling-based planners build their representations of space by sampling random configurations and using a fast collision checker to determine whether collisions will occur or not. These algorithms find a path between arbitrary samples of the search space and have a fast collision checker to determine whether a collision occurs on the path between two sampled states. A graph is built from the feasible sampled states to create a representation of the environment from which to select a path. As the number of samples increases, these methods can often be shown to be probabilistically complete, meaning that the probability of finding a valid path approaches 1, if a valid path exists. For many real-world path-planning problems, this method is very fast and reliable in practice.

One example of a sampling-based motion planning algorithm is the probabilistic road map [1], [42]. In this algorithm, random states in the terrain are chosen. The algorithm then attempts to join pairs of states using a local planning method. It then creates a road map out of all the
states that can be joined together. This is done in a preprocessing phase. When a path between initial and goal states needs to be determined, the algorithm only has to connect these states on the road map. This allows for the quickest determination of a path or multiple paths. Building a road map is a time-consuming process. The advantage of doing so is that once the road map is built, and assuming that the obstacles are not allowed to move, it can be used to plan quickly an arbitrary number of paths. If the goal is only to find a single path, however, much of the effort of building the road map may not be worthwhile. It would then be preferable to use a method that is concerned with connecting the start and goal locations rather than covering the complete configuration space. Moreover, it is difficult to include motion constraints without increasing the dimensionality of the problem.

The RRT method is another example of a sampling-based planning method [57],[58]. In this, the search space can be quickly explored without a lengthy preprocessing phase to explore the whole space. Furthermore, differential constraints can be easily incorporated, since these constraints are considered when adding states to the tree. Also, the standard RRT algorithm does not require the solution of local planning problems as in the probabilistic road map method. Instead of being given start and goal states and then finding the exact inputs to connect them, the algorithm chooses an input, computes the resulting goal state and then adds that state to the tree. The tree structure makes intuitive sense when planning a path for a system with motion constraints, since the algorithm can test to ensure that new motions are possible before adding new states.

In [37], it is shown that the probability of finding a path generated by an incremental planner converges to 1 exponentially fast with the number of random samples used to build the tree. Note that the convergence rate does not depend on the number of obstacles; rather it depends on the geometric properties of the environment. The convergence guarantee is true when samples are drawn form the search space without considering whether a feasible control exists. In [54], RRTs are used for real-time motion planning applications. In [21], an RRT path planner is shown to be effective in urban environments.

Motivated by these recent developments, we have developed real-time motion planning algorithms for a single UAV system and a multi-UAV system for use in obstacle rich environments (which is partially known). The motion planning algorithm controls two processes: (i) expan-
sion and (ii) execution. The expansion process builds a tree by adding feasible nodes to it while accounting for environmental and physical constraints. Whereas, the role of the execution process depends on the type of mission. In a single UAV mission, the execution loop runs the expansion process for the given time window and then selects a path for execution. While executing the path, the process allows the expansion process to expand the tree for searching lower cost paths with emphasis on an optimization heuristic. In addition to this, it checks if there is any pop-up obstacle appearing on the flight path. If a pop-up obstacle is encountered, the algorithm predicts the time to manoeuvre and searches for a new solution to avoid the pop-up obstacle and then continues with the mission. On the other hand, in a multi-UAV system, the execution process manages the way interactions take place and employs a conflict resolution strategy to resolve dynamic conflicts.

The rest of the chapter is organized as follows. Section 3.2 describes the path planning problems, scenario, constraints and solution approaches. In Section 3.3, we initially review the original RRT algorithm and then develops a real-time robust RRT algorithm for a single UAV system. In Section 3.4, path planning algorithm for a multi-UAV system is developed. A coordinate strategy is proposed which manage the interaction among UAVs and provide conflict free paths. In Section 3.5, the performance of the proposed path planner for single and multiple UAV systems is demonstrated by simulation results. Finally, concluding remarks are presented in Section 3.6.

3.2 Problem Description

In this section, we formally define path planning problems for a single UAV and a multi-UAV system in the presence of obstacle rich (cluttered) environments whose maps are not available in advance.

3.2.1 Scenario

Initially, we consider a mission in which a UAV needs to travel from its starting location, $x_{\text{init}}$, to a goal location $x_{\text{goal}}$, through an environment consisting of static and pop-up obstacles. We assume the environment is partially known and use the map to plan the path. The pop-up (i.e.,
previously unknown) obstacle may appear anywhere in the environment but it can be detected only when it is within the sensor range of a vehicle. Let \( X = \{(x,y) : x \in (a,b), y \in (c,d)\} \) be a search space in \( \mathbb{R}^2 \). Let \( X_{obs} \) be a set of polyhedral obstacles present in the search space \( X \). The free space is denoted as \( X_{free} \triangleq X \setminus X_{obs} \). Let \( x_{init} \) be an element of \( X_{free} \). The objective is to find a path from \( x_{init} \) to \( x_{goal} \) while satisfying geometric and kinematic constraints. By geometric constraints, we mean that a UAV is not allowed to enter into any obstacle. Note that a path is described by a set of waypoints that a UAV has to visit sequentially. The problem can be formalized as follows.

**Problem 1** (Feasible planning). Given \( x_{init} \) to \( x_{goal} \), find a sequence of waypoints \( W_1, \ldots, W_n \) in real-time such that the last waypoint \( W_n \) corresponds to the goal position \( x_{goal} \) while the path connecting the sequence of waypoints satisfies the geometric and kinematic constraints.

The problem becomes difficult because the motion constraint of the UAV (i.e., its turn radius) is comparable with the structure and dynamics of the environment. This means that the UAV constraints must be taken into account when finding paths so that unexpected collisions can be avoided. In this work, we account for actual trajectory characteristics while generating the paths, and hence, possible collisions, because of deviation from the actual path, can be avoided. The details of this are presented later in this chapter. After finding a feasible path in real-time using the proposed algorithm, the emphasis is to improve the quality of the path while following the waypath. Let \( J(x, x_{goal}) \) be the cost of a path from a waypoint at \( x \) to another waypoint at \( x_{goal} \). In this work, we use the following definition to evaluate the cost of a path.

**Definition 1** (Cost of the path \( J(x, x_{goal}) \)). Given a sequence of waypoints \( W_1, \ldots, W_n \), the cost of the path \( P \) from \( x \) to \( x_{goal} \) is defined as \( J(x, x_{goal}) = \sum_{i=1}^{n-1} ||W_{i+1} - W_i|| \), where \( ||.|| \) is the Euclidean norm and \( W_1 \) corresponds to the current position \( x \).

**Problem 2** (Close to optimal planning). Given a set of feasible paths, \( P_1, \ldots, P_m \), select the least cost path given as \( P^* = \min_i P \), where \( P = P_1, \ldots, P_m \).

**Problem 3** (Multi-UAV planning): Given the initial vehicle positions and desired goal positions, find a sequence of waypoints for each UAV such that the last waypoint corresponds to the goal position while the path connecting the sequence of waypoints satisfy the geometric and kinematic constraints and avoids collision with the other UAVs in the environment.
3.2.2 Assumptions and constraints

It is assumed that the UAVs are subjected to limited sensor and communication ranges, turn radius constraints and have constant velocity. Since the UAVs have limited sensor range, they can only detect pop-up obstacles when they are in sensor range. If the UAV detects a pop-up obstacle during its course, it has to re-plan its path from its current position. Depending on the velocity of the UAV, it can predict the time to collide $t_c$ and it needs to find a new path before $t_c$ is reached. Determining a path which satisfies physical constraints while avoiding pop-up obstacles, in obstacle rich environments, is difficult and computationally intensive. A UAV may take a detour to avoid a collision, however, to find such a new route which satisfies the turn radius constraint and leads to the goal location is not trivial.

Because of turn radius constraint, a UAV cannot make an instant turn, and hence, it has to start its collision avoidance manoeuvre by time $t_m$, the time to manoeuvre, which has to be less than the predicted time of collision $t_c$. Therefore, the time window available to produce a feasible solution is $t_m < t_c$.

We propose several variants of the standard RRT algorithm to generate a path in real-time. The UAVs are also subjected to a limited communication range, and hence a given UAV cannot always inform other UAVs about its planned path which may be in conflict with others. In this case, when UAVs arrive within communication range of each other they exchange their routes and determine the time of conflict $t_{con}$ and time of manoeuvre $t_m$ if they exist and find a new path if required. Designing a path planner with these constraints is a difficult problem that we address in this chapter. Note the distinction between $t_c$, the time for a UAV to collide with an obstacle, and $t_{con}$ the time for the paths of two UAVs to meet.

3.2.3 Solution approach

The path planner for a single UAV system is developed in a time window framework. Initially, the proposed robust RRT algorithm is run for 1 sec (preparation time $t_p < t_m$) to expand the tree while avoiding static obstacles and accounting for the turn radius constraint of the UAV. A path is chosen using a heuristic to move towards the goal position. While following the path, the tree is further expanded to find more solutions. Once a path is found, an optimization heuristic is
used to search for lower cost paths. After the UAV reaches a desired waypoint, the cost of the path which was going to executed next is compared with newly founded paths, so the path with minimum cost can be executed next. This process is continued until a UAV reaches $x_{goal}$.

The path planning algorithm for a multi-UAV system is developed by embedding the path planner developed for a single UAV within a framework that manages interactions among UAVs and uses a conflict resolution strategy to resolve conflicts.

### 3.3 Algorithms

In this section, we develop a path planner for UAVs based on the Rapidly-exploring Random Tree (RRT) algorithm. First, we give descriptions of the original RRT algorithm and its modification as in [96] to find an admissible path from a starting location to a goal location, given a priori. Then, we modify this algorithm to improve its performance and the quality of solutions obtained by incorporating certain heuristics and using a technique to convert the differential constraints into geometric ones thus incorporating the admissible trajectory characteristics into the planner. Next, we develop a real-time RRT for path planning of a single UAV using a time window approach, similar to the model predictive control approach, where the exploring tree is grown for a given fixed time window. We combine this with the anytime approach to give an algorithm that finds near optimal admissible paths to the desired goal location in the presence of static and pop-up obstacles. Finally, a multi-UAV path planner is developed in which each UAV uses the above path planning algorithm.

#### 3.3.1 RRT algorithm

The Raipdly-exploring Random Tree (RRT) algorithm was developed by LaValle [57] to explore a large configuration space efficiently in a short time. It has grown in popularity and many challenging problems have been solved using it. For example, see [14], [28], [38], [96], [55], [60] and [70]. The original algorithm incrementally builds a search tree that explores the whole configuration space quickly. The RRT algorithm is given in Algorithm 1, using the usual notation.

Given a starting point $x_{init}$, Algorithm 1 grows a tree for $k$ steps with $x_{init}$ as the first vertex
Algorithm 1  Original RRT Algorithm

1: \( T\.ADD\_VERTEX(x_{\text{init}}); \)  //Initialize tree \( T \) and add initial vertex to the tree
2: \( \textbf{for} \ i = 1 \text{ to } k \ \textbf{do} \)
3: \( x_{\text{rand}} \leftarrow RANDOM\_VERTEX(); \)  //Generate a random point in configuration space
4: \( x_{\text{near}} \leftarrow NEAREST\_VERTEX(x_{\text{rand}}, T); \)  //Find the vertex in tree \( T \) nearest to \( x_{\text{rand}} \)
5: \( u \leftarrow SELECT\_INPUT(x_{\text{near}}, x_{\text{rand}}); \)  //Find the input that generates an admissible path from \( x_{\text{near}} \) to \( x_{\text{rand}} \)
6: \( x_{\text{extend}} \leftarrow EXTEND\_EDGE(x_{\text{near}}, u, \Delta t); \)  //Propagate the state \( x_{\text{near}} \) along the path for \( \Delta t \) time
7: \( T\.ADD\_VERTEX(x_{\text{extend}}); \)  //Add the vertex \( x_{\text{extend}} \) to tree
8: \( T\.ADD\_EDGE(x_{\text{near}}, x_{\text{extend}}, u); \)  //Add the path from \( x_{\text{near}} \) to \( x_{\text{extend}} \) to tree
9: \( \textbf{end for} \)
10: return \( T \)

of the tree \( T \). To find an admissible path from \( x_{\text{init}} \) to a given goal point, \( x_{\text{goal}} \), the algorithm is run until a feasible path is found from any of the vertices of the tree to \( x_{\text{goal}} \). Note that in the original algorithm \( x \) represents the state of a system in its configuration space \( C \). However, it is suggested that one can choose suitable \( x \) according to the application in hand, which can be, in our case, waypoints. At each step, a random point, \( x_{\text{rand}} \), is selected from the configuration space using the \( RANDOM\_VERTEX \) function which draws a sample using a probability distribution function (step 3). Next, the function \( NEAREST\_VERTEX \) determines the vertex, \( x_{\text{near}} \), in the current tree \( T \) that is closest to \( x_{\text{rand}} \) with respect to a given metric \( \rho \) (step 4). Step 5 selects an input \( u \in U \), where \( U \) is the set of feasible inputs, which minimizes the \( \rho \)-norm between \( x_{\text{near}} \) and \( x_{\text{rand}} \) subject to the state equation \( \dot{x} = f(x, u) \) and the constraint that the resultant path should stay within the configuration space. The function \( EXTEND\_EDGE \) applies the chosen control input \( u \) to propagate the state, for \( \Delta t \) time, from \( x_{\text{near}} \) to \( x_{\text{extend}} \) (step 6). Then, the state \( x_{\text{extend}} \) and the edge from \( x_{\text{near}} \) to \( x_{\text{extend}} \) are added to the tree (steps 7 and 8). One can find more details of the algorithm in [57] and the references therein.

The RRT algorithm has the following interesting properties that make it efficient and thus popular. The exploration policy used in the RRT algorithm has a Voronoi bias [67]. That is, the probability distribution function used by \( RANDOM\_VERTEX \) in step 3 of Algorithm 1 is chosen such that for each sampling, the probability that a vertex becomes the one nearest to the random point generated is proportional to the volume of its Voronoi region. Thus, the search is biased toward those vertices with larger Voronoi regions which in turn represent unexplored regions of the configuration space. This makes RRTs explore rapidly. It can also be shown that the RRT algorithm is probabilistically complete.
Algorithm 1 gives the basic RRT algorithm. One can find several variations of this in the literature (for example, see [57] and [96]) designed to improve its performance for the particular application under study. The standard RRT algorithm provides a time-parametrized set of controls, which are open loop controls to move from $x_{\text{init}}$ to $x_{\text{goal}}$. The accuracy of the resultant path depends on the validity of the state space model being used. In reality, apart from the modelling uncertainties, there exist sensor inaccuracies, wind disturbances and other unmodelled factors. To tackle these, in [96], while a modified RRT algorithm is used to generate waypoints, the waypath is tracked using an inner loop controller. This is described below.

### 3.3.2 Modified RRT algorithm

To tackle the issues of uncertainties and disturbances that make the open loop solution provided by the RRT algorithm practically unusable, [96] suggests a modified RRT algorithm to be used in a closed-loop manner. Towards this, reference paths in terms of waypoints are generated using a modified RRT algorithm and these paths are tracked using an autopilot. The steps of the modified RRT algorithm proposed in [96] are given in Algorithm 2.

**Algorithm 2** Modified RRT Algorithm

1: $T$.ADD_VERTEX($x_{\text{init}}$); //Initialize tree $T$ and add initial vertex to the tree  
2: while $x_{\text{goal}} \notin T$ do  
3:  $x_{\text{rand}} \leftarrow$ RANDOM_NODE(); //Generate a random point in configuration space  
4:  $x_{\text{near}} \leftarrow$ NEAREST_VERTEX($x_{\text{rand}}$, $T$); //Find the vertex in tree $T$ nearest to $x_{\text{rand}}$  
5:  $x_{\text{extend}} \leftarrow$ EXTEND_EDGE($x_{\text{near}}$); //Propagate the state $x_{\text{near}}$ along the path for subject to constraints  
6:  COLLISION_CHECK($x_{\text{extend}}$, $x_{\text{near}}$); //Check for collision in the path from $x_{\text{near}}$ to $x_{\text{extend}}$  
7:  if no collision then  
8:  $T$.ADD_VERTEX($x_{\text{extend}}$); //Add the vertex $x_{\text{extend}}$ to tree  
9:  $T$.ADD_EDGE($x_{\text{near}}$, $x_{\text{extend}}$); //Add the path from $x_{\text{near}}$ to $x_{\text{extend}}$ to tree  
10:  if EXIST_PATH($x_{\text{extend}}$, $x_{\text{goal}}$) then  
11:  $T$.ADD_VERTEX($x_{\text{goal}}$); //Add the goal point to tree  
12:  $T$.ADD_EDGE($x_{\text{extend}}$, $x_{\text{goal}}$); //Add the path from $x_{\text{extend}}$ to $x_{\text{goal}}$ to tree  
13:  end if  
14:  end if  
15:  end while  
16: return $T$

The modified RRT algorithm is similar to the original RRT algorithm (Algorithm 1) except for the following differences. In step 3, the function $\text{RANDOM\_NODE}$ samples a new point in the configuration space as per a distribution function which sets $x_{\text{rand}} = x_{\text{goal}}$ with a small
probability. This aids in pulling the tree generated towards the goal point. In step 4, the vertex in the tree nearest to $x_{rand}$ is found with respect to a metric that takes into account the path constraints. Thus the constraints are built into the metric itself rather than selecting an admissible $u$ such that constraints are satisfied as in Algorithm 1. Moreover, given an autopilot, it is possible to estimate the expected error at each point along a path in tracking it and therefore, it is possible to plan valid paths without specifying the exact control inputs. In step 5, a point $x_{extend}$ is selected along the path segment of $x_{near}$ to $x_{rand}$. The length of the path segment $x_{near}$ to $x_{extend}$ is pre-defined based on the kinematic constraint at the beginning of the algorithm. The function COLLISION_CHECK checks whether there are any collisions with obstacles in the extended path (step 7). If no collision is detected, then $x_{extend}$ and the corresponding edge are added to the tree (steps 8 and 9). In step 11 it is checked whether the newly added point in the tree can be directly connected to the goal point. The algorithm repeats until a feasible path to goal point is found.

The modified RRT algorithm is similar to the Algorithm 1 except for the modifications as described above. Also, now the solution is given in terms of a reference path rather than time parametrized control inputs leading to an open loop solution. The reference path is tracked by a given autopilot. As the autopilot capabilities are taken into account in the planning phase itself, the path tracked during actual implementation is guaranteed to be feasible in the absence of moving obstacles and pop-up threats.

Due to the inherent random nature of the RRT algorithm, the paths generated by the algorithm can be far from being optimal. Also, a naive implementation of the RRT algorithm to find feasible paths may, in some cases, take a long to compute which makes it inadequate for use in real-time. We propose some heuristics and techniques to aid the RRT algorithm so that it can be used in real-time to give near-optimal solutions. The framework that we use to generate the paths is similar to that in [96]. However, we incorporate several heuristics to enhance the performance of the path planner both in terms of the cost and the computational time required to find paths so that the resultant path planner can be used to develop a real-time path planner for multi-UAV systems.
3.3.3 Robust RRT algorithm

We propose a modification of the RRT algorithm to grow the tree. The modified algorithm is given in Algorithm 3. Similar to [96], we generate reference paths in terms of waypoints. However, as discussed below, we incorporate a number of heuristics to accelerate the performance of the algorithm. Algorithm 3 expands the tree for the given time window ($t_p$ seconds) and the best solution obtained within this time window is chosen for execution. The heuristics used at different steps of Algorithm 3 are explained below.

As emphasized earlier, the motion planner has to find a near optimal solution in real-time for use in an obstacle rich environment. Moreover, it has to incorporate turn radii constraints of UAVs to avoid collisions. For this purpose, we design a robust RRT algorithm which assumes the existence of both a kinematic planner that is able to generate waypaths while accounting for the turn radius constraint of a UAV and a controller that can track paths which satisfy motion constraints of the UAV. The task of the algorithm is to provide a path to the controller that guides the vehicle from its current location to the goal location while avoiding obstacles (static, pop-up and dynamic) and maintains sufficient separation margins from obstacles to provide robustness against mild disturbances. In order to develop such a robust algorithm, we first develop a planner which generates a path from waypoint $w_{init}$ to waypoint $w_{goal}$ in a static environment. Next, using this as a basic planner, we develop a path planner which can find a near optimal solution in relatively quick time while avoiding static, pop-up and dynamic obstacles. The steps of the basic robust path planner are given in Algorithm 3.

Starting node: Step 1

The root of the search tree is created by assigning the initial position of the vehicle as the first node of the tree. Exploration to find a path then begins with the following steps.

Sampling strategy: Step 3

For the path planning problem, the configuration space is the environment which is a two-dimensional Euclidean space. A random sample, in that case, is a point $x_{rand} = (x, y)$ in the Cartesian coordinates that is chosen according to some sampling strategy. The sampling strategy
**Algorithm 3** Robust RRT Algorithm: Tree Expansion

**Input:** tree $T$, a goal location $x_{\text{goal}}$, time window for tree expansion $t_p$

1. **while** $t < t_p$ **do**
   2. $x_{\text{rand}} \leftarrow \text{RANDOM_VERTEX}();$ //Generate a random point in configuration space
   3. $(x_{\text{near}1}, x_{\text{near}2}, \ldots, x_{\text{near}N}) \leftarrow \text{NEAREST_VERTEX}(x_{\text{rand}}, T);$ //Find N nearest vertices of $x_{\text{rand}}$ in tree $T$
   4. **for** $K=1$ to $N$ **do**
      5. $x_{\text{extend}} \leftarrow \text{EDGE_EXTEND}(x_{\text{near}k}, x_{\text{rand}});$ //Extend the state $x_{\text{near}k}$ along the direction of $x_{\text{rand}}$
      6. $\text{CHECK_COLLISION}(x_{\text{near}k}, x_{\text{extend}});$ //Check for collision in the path from $x_{\text{near}k}$ to $x_{\text{extend}}$
      7. **if** no collision **then**
          8. $x_{\text{parent}} \leftarrow \text{FIND_PARENT}(x_{\text{near}k});$ //Find the parent node of $x_{\text{near}k}$
          9. $\text{CHECK_TRAJECTORY_COLLISION}(x_{\text{parent}}, x_{\text{near}k}, x_{\text{extend}});$ //Check for collision during transition and in presence of disturbance
          10. **if** no collision **then**
              11. $T.\text{ADD_VERTEX}(x_{\text{extend}});$ //Add vertex to the tree
              12. $x_{\text{greedy}} \leftarrow \text{GREEDY_EXTEND}(\text{FIND_PARENT}(x_{\text{extend}}), x_{\text{extend}});$ //Extend the new edge by distance equal to three times the turn radius of the UAV in the same direction
              13. $\text{CHECK_COLLISION}(x_{\text{extend}}, x_{\text{greedy}});$ //Check for collision of this extended edge
              14. **if** no collision **then**
                  15. $\text{CHECK_TRAJECTORY_COLLISION}(x_{\text{near}k}, x_{\text{extend}}, x_{\text{greedy}});$ //Check for collision during transition and in presence of disturbance
                  16. **if** no collision **then**
                      17. $T.\text{ADD_VERTEX}(x_{\text{greedy}});$ //Add vertex to the tree
                      18. **end if**
                  19. **end if**
              20. **end if**
          21. **end if**
      22. **end if**
   23. **end for**
24. **end while**
can be either to sample the entire environment equally (global exploration) or to sample the goal position more (biased exploration). A small bias towards the goal location aids in pulling the tree towards the goal. Therefore, we propose the following sampling strategy that has a bias towards the goal position. We choose the random sample point as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{cases} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} r_1L \\ r_2W \end{bmatrix}, & \text{with probability } (1 - p) \\ \begin{bmatrix} x_g \\ y_g \end{bmatrix}, & \text{with probability } p. \end{cases}$$

(3.1)

Here, \((x_0, y_0)\) is the South-West Cartesian coordinates of the search space. That is, \(x_0\) and \(y_0\) are the lowest \(x\) and \(y\) coordinate values of the search space. The random numbers, \(r_1\) and \(r_2\), are selected from a uniform distribution in the interval \([0, 1]\). \(L\) and \(W\) are the length and width of the search space and the goal point, \(x_{goal} = (x_g, y_g)\). Note that choosing the sample point as goal point with a non-zero probability \(p\) will bias the tree growth towards the goal point.

**Selecting nearest neighbour: Step 4**

In this step, we select the nodes to expand the tree based on some heuristics. In the basic RRT algorithm, at every stage of growing the tree, only one node nearest (with respect to a chosen metric) to the randomly generated sample point is selected and a new edge is created between this point and the random sample. We propose a scheme where \(N\) nearest neighbours are selected at every stage of the tree expansion and explore the possibility of extending the tree from these vertices. Also, the criterion (metric) used to select the nearest nodes alternates between two of heuristics that are described below. This is similar to that presented in [111].

**Exploration heuristic**

The primary emphasis of the search algorithm based on RRT is to find a feasible path through exploration as quickly as possible. In order to achieve this, we use the exploration metric which chooses neighbours depending upon the Euclidean distance to the random sample. Note that unlike the modified RRT algorithm, at every stage, we choose \(N\) nodes and extend paths from these vertices to the random sample. Thus, we have more nodes to extend the tree in subsequent sampling stages.

**Optimization heuristic**
The basic RRT algorithm does not take into account the path cost while expanding the tree. Due to this, the path cost could be far from the optimal cost. In order to generate paths which can mimic close to optimal solutions, we use the following metric for choosing nearest nodes. The optimization metric evaluates the nodes based on an estimate of the total path length through each node from the root to the goal location; that is, the sum of the cost from the root to the node in question and the approximate estimated cost (Euclidean distance) from this node to the goal node (ignoring obstacle avoidance constraints). Nodes of the tree are then ordered in an ascending order of estimated path lengths and the first $N$ vertices are selected as the nearest neighbours.

**Extended node: Step 5**

Once the nearest neighbour(s) is (are) found, the standard approach is to check the feasibility of the path(s) connecting the neighbour node(s) and the sample point. This is an important step in the process of growing the tree and needs to be considered carefully. Consider a case where the sampling point is too far from the nearest node in the tree and the straight line path connecting the two passes through an obstacle. In this case, the path is not feasible and hence the sample point will not be added to the tree. This is an example of a case where the sample point which is important in growing the tree is not beneficial. Another case, which is also more interesting, is the one in which a sample point is too close to the neighbours and has feasible (obstacle free) links with the neighbours. Even if this sample point is added to the tree, it may not serve the purpose of tree expansion since it results in nodes being too near to each other. Moreover, the presence of such waypoints (nodes) close to each other may require a UAV to make a turn before it completes the previous turn which may cause difficulties from a tracking point of view. Thus, we note that, all the generated sample points cannot be added to the tree as it is. To overcome this problem, we fix the minimum length of a newly added edge as three times the turn radius of the UAV. Note that the function $EDGE\_EXTEND$ takes a sample point as input and gives as output a node in the same direction (along the line connecting the nearest neighbour to the sample point) but at a distance equal to three times the turn radius of the UAV from the nearest neighbour node. It is this edge so created that is checked for collision with obstacles in the next step (step 6) and added to the tree along with the corresponding node (step 10) if found collision free. Also, note that the extension of edges as described above provides a
way to generate straight paths along long straight corridors.

Collision checker: Step 6

In this step, the edge extended as in step 5 is checked for collision with any of the obstacles in the configuration space. That is, we need to check whether the line segment from $x_{\text{near}}$ to $x_{\text{rand}}$ passes through any of the given obstacles. We assume that the obstacles are polytopes or can be approximated as polytopes. Thus each obstacle is characterized by hypersurfaces that form the faces of these polytopes. For example, an obstacle in a two dimensional configuration space is a polygon defined by lines through each of its edges (see Figure 3.1). Let the configuration space consist of $B$ obstacles. Let the $j$th obstacle be defined by $n_j$ hypersurfaces. Let the $i$th hypersurface of the $j$th obstacle be defined by the unit vector $a_{ij}$ perpendicular to that hypersurface towards the obstacle (refer Figure 3.1). Then, a point $x$ on the line segment from $x_{\text{near}}$ to $x_{\text{rand}}$ passes through the $j$th obstacle if there exists an $\alpha \in [0, 1]$ such that the conjunction of $n_j$ linear equalities $\bigwedge_{i=1}^{n_j} (a_{ij}^T x > b_{ij})$ are satisfied (all should be satisfied) where $x = \alpha x_{\text{near}} + (1 - \alpha) x_{\text{rand}}$. Here, $b_{ij}$ (refer Figure 3.1) is positive if the unit vector in the direction from the origin to the closest point on the hypersurface is aligned along $a_{ij}$ and is negative if they are aligned in opposite directions.

Thus, the criterion for a valid path is that the disjunction of the constraints $\bigvee_{i=1}^{n_j} (a_{ij}^T x \leq b_{ij})$ should hold (at least one should be satisfied) for all $\alpha \in [0, 1]$ and for all $j = 1, \ldots, B$.

Given $x_{\text{near}}$ and $x_{\text{rand}}$, the function CHECK_COLLISION checks whether the path from $x_{\text{near}}$ to $x_{\text{rand}}$ passes through any of the known obstacles using the procedure described above and if true, sets up a collision flag.

Ensuring collision free trajectory

The path planning algorithm that we propose takes into account the deviation of an actual UAV from the waypaths (line segments joining the waypoints) that occurs, due to kinematic constraints like minimum radius of turn, while tracking the waypath. This is a significant contribution as the other path planning algorithms do not account for this. The actual UAV trajectory can deviate substantially from the waypoint path [28], especially near sharp corners where the
waypath changes abruptly (see Figure 3.2). If the UAV travels to the waypoint $k$ as shown in the Figure 3.2, while switching to waypoint $k + 1$ it will overshoot. As seen in the figure, these oscillations can be significant for a sharp turn and this may lead to collisions with obstacles which are undesirable.

To address the issue of a UAV overshooting – leading to possible collisions with obstacles – while taking sharp turns, we have devised the following strategy that enables the UAV to change direction smoothly. The actual trajectory of a UAV can be parameterized in terms of straight lines and circular arcs such that all the circular arcs have radii of turn greater than the minimum radius of turn of the UAV. For the case depicted in Figure 3.2, since the $(k + 1)$th node (which was selected randomly) is known, it is possible to parameterize the actual trajectory of the UAV from $k - 1$ to $k + 1$ in terms of straight lines and an arc that satisfies the minimum turn radius constraint as shown in the figure. The trajectory of the UAV now stays within the isosceles
triangle $\triangle p_1 kp_2$ with similar sides each having length $d$ (see Figure 3.2). The parameter $d$ depends on how sharp the change in waypath is. The path planner has to ensure that there are no obstacles within this triangle. If there is an obstacle, the randomly selected node is not incorporated in the tree. The path planning algorithm along with the smooth trajectory transition proposed, thus ensures that the actual trajectory is free from collisions.

By the above procedure, we have successfully converted a differential constraint (minimum turn rate) into a geometric constraint with an additional obstacle (the isosceles triangle) for each new node. Moreover, the parametrization of the trajectory in terms of straight lines and arcs gives a way to switch to the next waypoint smoothly. For example, a UAV tracking the waypoints $k - 1$, $k$, and $k + 1$ starts taking the turn towards $k + 1$ at the point $p1$ and continues the turn until it arrives at point $p2$ as shown in Figure 3.2. With a good path following technique, the UAV should be able to track this trajectory as long as the radius of arc is greater than the minimum turn radius, $R_{\text{min}}$. The guidance law developed in Chapter 2 tracks such a path smoothly. Note that we have parameterized the trajectory of the UAV in terms of straight lines and circular arcs as it is simple. However, one can choose any other parametrization that ensures that the parameterized trajectory does not pass through the obstacles.

We define safe corridors about the waypaths to ensure that the UAV trajectory is free from collision even when the UAV is operating in the presence of mild disturbances arising due to wind, gusts and navigation errors. Figure 3.3 depicts a schematic of such a safe corridor of width $w$. The required width, $w$, of the corridor is determined by computing the maximum

![Figure 3.2: Transition behaviours](image-url)
deviation of the UAV, flying in constant wind using the guidance law described in Chapter 2, from a straight line reference path. We perform several simulations with different wind and initial conditions to determine the required corridor width. For a given corridor width, the path planner ensures that these corridors about the waypaths are free from obstacles. This is achieved by dividing the corridor into smaller regions, and checking for the presence of obstacles in each of them. The incorporation of this heuristic guarantees robustness to mild disturbances, that is, a collision free trajectory is maintained even in the presence of small disturbances. Note that the function \textit{CHECK\_TRAJECTORY\_COLLISION} checks for both the turn radius constraint as described previously, and robustness through incorporation of safe corridors in step 9.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{smooth_switching.png}
\caption{Smooth switching}
\end{figure}

\textbf{Greedy approach for edge extension: Steps 14-18}

Due to the nature of random incremental growth of the RRT, the resultant path that is composed of the branches of the tree will not be straight for long distances. This will result in a UAV, following such a path, doing a large number of unnecessary turns. To obtain longer straight paths, we propose a greedy strategy that exploits the basic incremental growth characteristic
of RRT while preserving random features. In such a strategy, the incremental growth (the edge length equals three times the turn radius of the UAV) is continued until an obstacle is encountered, and intermediate nodes are inserted at each increment. Note that this is different from the standard RRT algorithm in which the increment is random. The intermediate nodes serve as vertices for further extension of the tree which is important in growth of the tree. This strategy will curb the tendency of growing too many path segments of shorter lengths which are replaced by fewer paths of longer lengths, and therefore speeds up the algorithm. At the same time, insertion of intermediate nodes preserves the randomness of the tree expansion. Given an expansion point, $x_{\text{extend}}$, the function $\text{GREEDY\_EXTEND}$ provides another point, in the same direction, such that the length of the edge is three times the turn radius of the UAV.

This completes the steps in tree expansion. We have described the important steps of the algorithm. The actual implementation depends on the particular application; one can modify each step to suit the problem at hand. The UAV path planner allocates a certain duration, $t_p$, for tree expansion. Based on the tree built in that interval, a path is chosen to be followed by the UAV. By the time the UAV follows a portion of this path, the tree can be further expanded and a complete path to the goal location can be found. Next, we present how this algorithm is executed.

### 3.3.4 Robust RRT algorithm: Execution

Although a path planner based on RRT will eventually find a feasible path to the goal location, it is not possible, \textit{a priori}, to determine an upper bound for the time within which a feasible solution will be obtained. Also, the path planner has to be used in real-time in an obstacle rich environment, the map of which may not be available well in advance. If there is sufficient time, like in the case of ground robots which can halt until the computations are over, then one can use off-line methods (e.g., the visibility line method) to find an optimal path to the goal location. However, this is not possible in the case of fixed wing UAVs which have to do computations on the fly. Therefore, we assume that a UAV has only $t_p$ seconds, which is the preparation time for the mission, to find a feasible path. This necessitates the modification of the RRT based path planner to an ‘anytime’ algorithm. We propose the following heuristics, given in Algorithm 4, to convert the RRT based path planner to an anytime algorithm to be used in real-time. The steps in this algorithm are detailed below.
Algorithm 4 Robust RRT Algorithm : Execution

Inputs: a starting location $x_{init}$, a goal location $x_{goal}$, the environment (obstacles map), mission preparation time $t_p$

1: $\mathcal{T}.ADD\_VERTEX(x_{init})$ //Initialize tree $\mathcal{T}$ with node at $x_{init}$
2: $\mathcal{T} \leftarrow ROBUST\_RRT\_EXPANSION(\mathcal{T}, x_{goal}, x_{init}, t_p)$ // Expand the tree by adding nodes (Algorithm 3) for $t_p$ second
3: while UAV has not reached $x_{goal}$ do
4: if $PATH\_TO\_GOAL(\mathcal{T})$ then
5: $P \leftarrow CHOOSE\_PATH\_TO\_GOAL(\mathcal{T})$ //Choose the best path to execute
6: else
7: $P \leftarrow CHOOSE\_PATH\_TOWARDS\_GOAL(\mathcal{T})$ //Use a heuristic to choose a path towards the goal location
8: end if
9: $x_{next} \leftarrow NEXT\_WAYPOINT(P)$ //Find next waypoint on this path
10: $t_r \leftarrow TIME\_TO\_GO(x_{next})$ // Estimate time to reach $x_{next}$
11: while UAV has not reached $x_{next}$ do
12: $\mathcal{T} \leftarrow ROBUST\_RRT\_EXPANSION(\mathcal{T}, x_{goal}, x_{next}, t_r)$ // Expand the tree by adding nodes (Algorithm 3) for $t_r$ second
13: end while
14: end while

Initialization: Step 1

In this step, the root of the search tree is created by assigning the initial position of the vehicle which becomes the first node of the tree. Each node of the tree holds the following information: its parent node, its own position, the cost to travel to the node from the root node, the underapproximate cost (that is, the Euclidean distance between the node and goal without considering the presence of obstacles and vehicle kinematics) to reach the goal, the actual cost to reach the goal if the node is connected via an edge to the goal node (otherwise this value is assigned infinity), and a flag which is set to 1 if the node can be connected to the goal and 0 otherwise.

Exploration: Step 2

Although the RRT algorithm is much quicker compared to many other path planning algorithms, there is no guarantee that it will provide a feasible path in a given fixed time. The way we handle this issue is by expanding the tree, using Algorithm 3, for the given time window only. The function $ROBUST\_RRT\_EXPANSION$, which is Algorithm 4, explores the search space to find paths to the goal within the given time interval; once the available time runs out, it returns
the current tree.

**Path selection: Step 4**

The function *PATH_TO_GOAL* checks whether any feasible path from the initial node to the goal node exists in the tree returned by *ROBUST_RRT_EXPANSION*. If true, then a complete continuous path is found by backtracking nodes from the $x_{goal}$ to $x_{initial}$. If more than one feasible path is found, then the best path for the UAV to follow is chosen, by *CHOOSE_PATH_TO_GOAL*, by comparing the costs of the feasible paths.

**Branch selection: Step 6**

In the case that no valid path from the initial node to the goal location is found within the given time window, we have to choose a branch that facilitates the motion of the UAV towards the goal location. A tempting choice for this is to select a path, in the current tree, from the initial node to a node that is closest to the goal location. However, such a path may not be the one with minimum total cost. Finding the minimum cost path by exhaustively evaluating all the paths would be computationally very expensive. Therefore, we use a heuristic in which we consider only those nodes that are in the neighbourhood of the goal node. To be precise, we compare the paths from initial node to all those nodes that fall within a ball\(^1\) of radius $R$ about the goal position, and choose the one with minimum cost. This is what the function *CHOOSE_PATH_TOWARDS_GOAL* does. We start with a nominal value of $R$ and if there is no node within the ball of radius $R$, then we increase the radius incrementally until a node falls in the ball.

**Path tracking (execution): Step 8**

Once a branch is selected using the heuristic suggested above, the UAV will start following the first segment of the selected path using the guidance law developed in Chapter 2. As discussed before, the path planner gives the path in terms of waypoints. When a UAV is flying from one waypoint to another, its computational power is used only for computing the control required to

\(^1\)The ball of radius $R$ about a point $(x_g, y_g)$ is defined as $B_R(x_g, y_g) = \{(x, y) \in \mathbb{R}^2 : \sqrt{(x-x_g)^2 + (y-y_g)^2} \leq R\}$
track the waypath, and its processor mostly stays idle. The computation time available during
this phase can be used to find a better path to the goal. Towards this, we do the following:
First, we predict the time to reach the next waypoint using the kinematic properties of the UAV;
next, we further expand the tree until the UAV reaches the next waypoint. Given a path to
the goal, the function NEXT WAYPOINT will give the next waypoint on this path. Given
the current position of the UAV, the function TIME TO GO estimates the time required, \( t_r \), to
reach the next waypoint. During this time window \( t_r \), the tree is further expanded in pursuit
of finding better paths to the goal. Once the UAV reaches the next waypoint, the algorithm
ROBUST RRT EXPANSION returns the updated tree with new nodes and edges. This tree is
checked for a path with lesser cost, and if found, the UAV again checks that a complete path
to the goal is found. If it finds a path, then it tracks it; otherwise it continues in the previously
selected path. Either way, the tree expansion and the search for a better path are continued until
the next waypoint is reached. This process is repeated until the goal point is reached.

This explains the basic steps of our proposed algorithm to find a path from a starting location
to a given goal location using the RRT algorithm augmented with some heuristics to obtain
better performance.

3.4 Planning for multiple UAVs

Cooperative path planning has been used for area coverage navigation in obstacle environments
and for task assignment. Tsourdos et al. [109] proposed cooperative path planning for a team
of UAVs to cover an area of interest while avoiding obstacles and reaching a particular location
at specified time and orientation. Later, the algorithm was extended to produce safe and flyable
paths using Pythagorean hodograph curves that have lengths close to the Dubin’s path. They
also consider coordinated path following for multiple UAVs with rendezvous [98]. Recently, the
authors have presented their work on feasible path planning for safe operation of multiple UAVs
in a book [110]. In this work, we are considering multiple UAVs path planning in obstacle rich
environments with collision avoidance. A number of multiple agent path planning techniques
have been developed in [85], [74], [8], [97] but multiple UAVs path planning has not received
much attention.

Generating simultaneously de-conflicting paths for several UAVs is a challenging task due
to the space constraints that one path imposes on another and the physical constraints which are especially important in an obstacle rich environment. In this section, we propose a multi-UAV path planning algorithm for a team of cooperative UAVs by combining the path planning algorithm developed in the last section with a conflict resolution strategy (CRS). Because of this, the resultant algorithm preserves the benefits of a single UAV path planning algorithm while avoiding conflicts.

As UAVs are operating in a decentralized manner, it is necessary to have a conflict resolution strategy that ensures the UAVs have de-conflicting paths while travelling towards their destination. The CRS should include conflict detection, search for solutions and communication with other UAVs to reach an agreement for conflict resolution. Two UAVs are said to be in conflict if they are within $\varepsilon$-distance from each other at some time $t$. The proposed CRS is invoked when UAVs are in conflict. While the $i$th UAV is communicating with neighbours, it can be in one of the following situations: (i) its current waypath is in conflict with neighbouring UAVs (refer Figure 3.4); (ii) its current waypath is not in conflict but it does not have next waypath (refer Figure 3.6); (iii) its current waypath is not in conflict but it has a waypath to execute next (refer

Figure 3.4: Three UAVs have their paths in conflict. The point of conflict in space is shown in the gray triangle.
Figure 3.7) and (iv) its current waypath is not in conflict and it has a complete path to the goal location (refer Figure 3.8).

![Flow diagram of the conflict resolution strategy](image)

Figure 3.5: Flow diagram of the conflict resolution strategy

Depending on the situation of the $i$th UAV, an appropriate action needs to be taken for planning. In the first case, UAVs are in a conflict, and they need to determine a conflict free path. The process of conflict resolution is shown in Figure 3.5 and the actual process of conflict resolution is presented in Algorithm 5. In the second and third cases, the UAVs are not in conflict but they do not have a complete path to goal, and therefore the path planning algorithm has to keep searching to find paths to the goal. In the last case, each UAV has a complete path to goal and they are not in conflict, so each UAV can carry out its mission without changing its path. However, in order to improve the quality of the path, the algorithm keeps exploring the search space with emphasis on optimization.

Consider a scenario shown in Figure 3.4 in which three UAVs are travelling from their given starting locations to desired goal locations. During their flight, each UAV finds that their paths are in conflict with neighbouring UAVs. The point of conflict in space is shown by a gray triangle in Figure 3.4. In order to avoid conflict, one may think of using a roundabout
Figure 3.6: A multi-UAV path planning scenario, each UAV is within communication range of each other and they are trying to find de-conflicting paths. UAVs do not have next waypath to fly, and they are still searching for paths to their goals.

collision avoidance approach [76]. However, such a scheme may not be applicable in obstacle rich environments as there may not be sufficient space for a roundabout manoeuvre. Hence, UAVs have to find alternative paths to avoid conflict. While finding alternative paths, each UAV has to coordinate with neighbouring UAVs; otherwise the alternative paths could be in conflict. For coordination, each UAV has to share its plan (where a plan corresponds to the determined path) with neighbours through broadcasts. If a conflict exists, the conflict resolution strategy presented in Algorithm 5 is employed to resolve it.

3.4.1 Assumptions

At the beginning of the mission, it is assumed that the number of obstacles, the location of the obstacles and their shapes are known. With this information each UAV executes the robust RRT algorithm (Algorithm 3) to find a path to its goal location for a given time window. If the starting locations of the UAVs are within communication range then the UAVs share their plans.
If paths are in conflict (which would result in a collision), then the corresponding UAVs need to resolve these conflicts with each other by changing their paths appropriately. Next, we outline the key assumptions made in developing the multi-UAV path planning algorithm presented in Algorithm 7.

- Each UAV operates in a decentralized manner and exchanges information with its neighbours when they are within the communication range.
- UAVs which cannot communicate are not going to be in conflict in the immediate near future.
- UAVs which are in conflict plan alternative paths sequentially.

The first assumption allows all UAVs to plan continuously either in search of a goal or to find lower cost paths. The information exchange is required to determine a conflict and to update new information perceived by each UAV. The second assumption limits the constraints imposed by other UAVs. This means that a UAV can detect a dynamic conflict at any time. The third assumption ensures that asynchronous planning does not lead to conflicting paths by modifying the plan of one UAV while the other UAV is planning. These assumptions are necessary to resolve conflicts.
Table 3.1: Set Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_i$</td>
<td>The set neighbouring UAVs including the $i$th UAV</td>
</tr>
<tr>
<td>$P_i$</td>
<td>The set of the plans of neighbouring UAVs including the $i$th UAV</td>
</tr>
<tr>
<td>$N_i^c$</td>
<td>The set neighbouring UAVs that are in a conflict with the $i$th UAV</td>
</tr>
<tr>
<td>$P_i^c$</td>
<td>The set of the plans of neighbouring UAVs that are in a conflict with the $i$th UAV</td>
</tr>
<tr>
<td>$N_{nc}$</td>
<td>The set neighbouring UAVs that are not in a conflict with the $i$th UAV</td>
</tr>
<tr>
<td>$P_{nc}$</td>
<td>The set of the plans of neighbouring UAVs that are not in a conflict with the $i$th UAV</td>
</tr>
</tbody>
</table>

3.4.2 Conflict resolution

The conflict resolution strategy presented in Algorithm 5 determines de-conflicted paths sequentially. The order in which the UAVs find de-conflicted paths is based on a token number. A token number is assigned to each UAV at the beginning of the mission. For this purpose, we propose a simple strategy in which intermediate nodes are introduced into the current path to bypass the conflict, the steps of introducing intermediate nodes are explained in Algorithm 6. Figure 3.9 illustrates the idea; two intermediates nodes are introduced for UAVs 1 and 2, respectively, so that their paths are not in have conflict and a solution can be found quickly. In this way, we can use the already available paths for generating conflict free paths which saves computational effort, and hence, can be helpful for real-time applications. However, the resul-
tant paths can be very sub-optimal. To address this issue, after avoiding conflicts, each UAV continues to search for newer paths with lower cost.

Figure 3.9: UAV 3 does not change its plan whereas UAVs 1 and 2 find alternative paths to bypass conflict by introducing intermediates nodes.

Once a conflict is detected, the set of UAVs that are in conflict (called \( N^c_i \)) is processed sequentially to find alternative paths for each UAV except the one holding the highest token number. We assume that each UAV is identified by its token number, assigned at the beginning of the mission. The inputs to the algorithm are \( N_i, \mathcal{P}_i, N^c_i, \mathcal{P}^c_i, p_i \) and \( x_{current} \). The description of each set is given in Table 3.1.

The algorithm initially creates two sets: (i) a set of non-conflicting UAVs \( \mathcal{N}^{nc}_i \) and (ii) the corresponding set of plans \( \mathcal{P}^{nc}_i \). These are the sets that are not in conflict and the proposed algorithm has to ensure that the newly obtained search paths should not be in conflict with any plans of \( \mathcal{P}^{nc}_i \). Next, the \( N^c_i \) is sorted out in a descending order so that UAVs with higher token numbers can be processed first; The sorted set is called \( S_{N^c_i} \). Then, the algorithm compares the first element of the set \( S_{N^c_i} \) with its own token number. If both are equal, then the \( i \)th UAV is holding the highest token and it does not have to find an alternative path. In this case, the UAV
Algorithm 5 Conflict resolution

Input: $N_i$, $P_i$, $N^c_i$, $P^c_i$, $p_i$, $x_{current}$

1: $P^{nc}_i = P_i - P^c_i$, $N^{nc}_i = N_i - N^c_i$ // Create the sets of non-conflicting UAVs
2: $S_{N^{nc}_i} \leftarrow$ SORT($N^{nc}_i$) // Sort the set of conflict UAVs set in a descending order
3: if $i == S_{N^{nc}_i}(1)$ then
4: Keep the existing path // Since the $i$th UAV has the highest token number, it does not have to find an alternative path
5: else
6: $p^{nc}_i = P^{nc}_i \cup p_{S_{N^{nc}_i}(1)}$, $N^{nc}_i = N^{nc}_i \cup S_{N^{nc}_i}(1)$ // Update the sets of non-conflicting UAVs
7: for $k = 2$ to $|S_{N^{nc}_i}|$ do
8: if $i == S_{N^{nc}_i}(k)$ then
9: $p_i \leftarrow$ FINDALTERNATIVE_PATH($P^{nc}_i$, $N^{nc}_i$, $p_i$, $x_{current}$) // Given a set of paths of neighbouring UAVs, find conflict free path, by introducing intermediates nodes
10: SENDALTERNATIVE_PATH($p_i$, $i$) // Broadcast the newly find alternative path to neighbours so they can use it for finding new paths
11: else
12: $(p_{S_{N^{nc}_i}(k)}, S_{N^{nc}_i}(k)) \leftarrow$ RECEIVEALTERNATIVE_PATH() // Receive the newly find alternative path from a neighbour that has recently found an alternative path
13: $P^{nc}_i = P^{nc}_i \cup p_{S_{N^{nc}_i}(k)}$, $N^{nc}_i = N^{nc}_i \cup S_{N^{nc}_i}(1)$ // Update the sets of non-conflicting UAVs
14: end if
15: end for
16: end if

keeps following its existing path and other conflicting UAVs find alternative paths.

In case the $i$th UAV does not hold the highest token number, it has to find an alternative path to avoid conflict. Next, $N^{nc}_i$ and $P^{nc}_i$ are updated by including the token number and plan of the UAV that holds the highest token number. The sorted set (list) $S_{N^{nc}_i}$ is processed sequentially using a for loop in line number 7. The algorithm compares each element of $S_{N^{nc}_i}$ with its own token number. If they are not equal, it waits for a broadcast from a neighbouring UAV of its plan. When it receives the plan, the $i$th UAV updates $N^{nc}_i$ and $P^{nc}_i$. When both are equal, the $i$th UAV finds an alternative path using the FINDALTERNATIVE_PATH function. The function takes as input the plans of UAVs that are not in conflict, $P^{nc}_i$, the current plan of the $i$th UAV, $p_i$, and the position where a UAV would initiate a turn, $x_{current}$, and returns a de-conflicting path. The steps of finding an alternative path are presented in Algorithm 6. Once the $i$th UAV finds the conflict free path, it broadcasts the path to its neighbours. In this way, the conflict resolution algorithm runs on each UAV that is in conflict and provides conflict free paths for each one.
3.4.3 Finding alternative paths

The function $\text{FIND\_ALTERNATIVE\_PATH}$ performs two tasks in order to generate de-conflicting paths: (i) it first generates local paths by introducing intermediate nodes and (ii) it checks for conflicts based on received paths. Both these steps are independent. However, paths need to be identified before they are checked for conflicts with the plans of neighbouring UAVs. In order to generate local paths quickly, as discussed earlier, we introduce intermediate nodes to bypass conflicts. The UAV predicts its position from where it would like to initiate a manoeuvre and makes this predicted position the root for the tree (lines 1-2). Then, instead of the actual goal position, the next waypoint which is after the conflict point, is assigned as the goal location to find local paths quickly (line 3). As the effective search space is reduced, the algorithm should find paths very quickly. When local paths are identified, each path is checked for conflicts with the plans of neighbouring UAVs (line 10). If there is no conflict, then the solution is accepted, otherwise, the algorithm keeps generating local paths until a de-conflicting path is identified.

In order to speed up this process, the algorithm starts generating local paths when a conflict is detected and when it receives paths from neighbouring UAVs through the conflict resolution mechanism, it checks the feasibility of each local path vs received paths until a feasible (de-conflicting) path is identified.

\begin{algorithm}
\caption{Local planning algorithm : finding alternative paths}
\input{local_planning_algorithm}
\end{algorithm}

\begin{tabular}{l}
\textbf{Algorithm 6} Local planning algorithm : finding alternatives paths \\
\textbf{Input : } $\mathcal{P}_i^{nc}, \mathcal{M}_i^{nc}, P_1, x_{\text{current}}$
\end{tabular}

1: $x_{\text{manoeuvering}} \leftarrow PREDICT\_POSITION(x_{\text{current}}, t)$ // Predict the position of the UAV after $t$ seconds
2: $\mathcal{T}.ADD\_VERTEX(x_{\text{manoeuvering}})$ //Initialize tree $\mathcal{T}$ with node at $x_{\text{manoeuvering}}$
3: $x_{\text{goal}} \leftarrow NEXT\_CONFLICT\_FREE\_WAYPOINT(p_i)$ //Find next waypoint on the path that is after the conflict point
4: $\mathcal{T} \leftarrow ROBUST\_RRT\_EXPANSION(\mathcal{T}, x_{\text{goal}}, x_{\text{manoeuvering}})$ // Expand the tree by adding nodes (Algorithm 3) until a path is found
5: $p_{li} \leftarrow CHOOSE\_PATH\_TO\_GOAL(\mathcal{T})$ //Find the local path $p_{li}$ by backtracking the path from $x_{\text{goal}}$ to $x_{\text{manoeuvering}}$
6: $CHECK\_DYNAMIC\_CONFLICT(p_{li}, \mathcal{P}_i^{nc})$
7: \textbf{while} the selected path is in conflict with any path \textbf{do}
8: $\mathcal{T} \leftarrow ROBUST\_RRT\_EXPANSION(\mathcal{T}, x_{\text{goal}}, x_{\text{manoeuvering}})$ // Expand the tree by adding nodes (Algorithm 3) until a new path is found
9: $p_{li} \leftarrow CHOOSE\_PATH\_TO\_GOAL(\mathcal{T})$ //Find the local path $p_{li}$ by backtracking the path from $x_{\text{goal}}$ to $x_{\text{manoeuvering}}$
10: $CHECK\_DYNAMIC\_CONFLICT(p_{li}, \mathcal{P}_i^{nc})$
11: \textbf{end while}
12: return $p_{li}$
3.4.4 Multi-UAV RRT: Algorithm

In this section, we present a real-time multi-UAV path planner using the robust RRT algorithm presented in Section 3.3.3 and the conflict resolution mechanism presented in Algorithm 5. The algorithm works similar to the robust RRT algorithm for a single UAV system except when UAVs are in conflict. The steps of the path planning algorithm are presented in Algorithm 7.

Algorithm 7 Multi-UAV Path Planning: Execution

This algorithm runs parallelly on each UAV

1: \( T . \text{ADD_VERTEX}(x_{\text{init}}) \) //Initialize tree \( T \) with node at \( x_{\text{init}} \)
2: \( T \leftarrow \text{ROBUST_RRT_EXPANSION}(T, x_{\text{goal}}, x_{\text{init}}, t_p) \) // Expand the tree by adding nodes (Algorithm 3) for \( t_p \) second
3: if \( \text{PATH_TO_GOAL}(T) \) then
4: \( p_i \leftarrow \text{CHOOSE_PATH_TO_GOAL}(T) \) //Choose the best path to execute
5: else
6: \( p_i \leftarrow \text{CHOOSE_PATH_TOWARDS_GOAL}(T) \) //Use a heuristic to choose a path towards the goal location
7: end if
8: while Tracking a waypath & UAV has not reached \( x_{\text{goal}} \) do
9: \( x_{\text{next}} \leftarrow \text{NEXT_WAYPOINT}(p_i) \) //Find next waypoint on this path
10: \( t_r \leftarrow \text{TIME_TO_GO}(x_{\text{next}}) \) // Estimate time to reach \( x_{\text{next}} \)
11: \( p_i \leftarrow \text{EVALUATE_PRIORITY}() \) //
12: while UAV has not reached \( x_{\text{next}} \) do
13: \( (\mathcal{P}_i, \mathcal{N}_i) \leftarrow \text{FIND_NEIGHBOUR}(i) \) // Find the neighbours of the \( i \)th UAV and paths that they are going to follow
14: \( (\mathcal{P}^c_i, \mathcal{N}^c_i) \leftarrow \text{FIND_CONFLICT}(\mathcal{P}_i, \mathcal{N}_i) \) // Find the neighbours that have conflicts with the \( i \)th UAV
15: if \( \mathcal{N}^c_i \neq \emptyset \) then
16: \( x_i \leftarrow \text{PREDICT_POSITION}(p_i) \) // Find the position on the this path where the \( i \)th starts turning
17: \( p_i \leftarrow \text{CONFLICT_RESOLVE}(\mathcal{P}_i, \mathcal{N}_i, \mathcal{P}^c_i, \mathcal{N}^c_i, T, p_i, x_i) \)
18: end if
19: \( T \leftarrow \text{ROBUST_RRT_EXPANSION}(T, x_{\text{goal}}, x_{\text{next}}, t_r) \) // Expand the tree by adding nodes (Algorithm 3) for \( t_r \) second
20: end while
21: if \( \text{PATH_TO_GOAL}(T) \) then
22: \( p_i \leftarrow \text{CHOOSE_PATH_TO_GOAL}(T) \) //Choose the best path to execute
23: else
24: \( p_i \leftarrow \text{CHOOSE_PATH_TOWARDS_GOAL}(T) \) //Use a heuristic to choose a path towards the goal location
25: end if
26: end while
Finding a path/branch for each UAV quickly: Steps 1-7

At the start of the mission, for the given obstacle map, the starting location and the goal location each UAV finds a path for the given time window using Algorithm 3. The selection of a path is done using the same heuristics as used in Algorithm 4. Note that steps 1 – 7 are the same as in Algorithm 4. This is because each UAV operates in a decentralized manner and checks for conflict only it is within communication range of another UAV.

Path execution and conflict resolution

Once a branch is selected, the UAV will start following the first segment of the selected path using the guidance law developed in Chapter 2. Similar to Algorithm 4, we predict the time to reach the next waypoint using the kinematic properties of the UAV (line 14-15); next, we further expand the tree until the UAV reaches the next waypoint. Meanwhile, the UAV checks if there is any neighbour within communication range (line 13). If there are UAVs within communication range, then they share their plans. After sharing plans, each UAV checks whether its plan is in conflict with any neighbours (line 14). This is done by checking for collision in both space and time. If they are in conflict, the conflict resolution strategy presented in Algorithm 5 is employed to resolve conflicts (lines 15-17).

As mentioned before, each UAV keeps expanding its tree while following the current waypath. Similar to Algorithm 4, given the current position of the UAV, the function $TIME_{TO\_GO}$ estimates the time required, $t_r$, to reach the next waypoint. During this time window $t_r$, the tree is further expanded in pursuit of finding better paths to the goal. If it finds a better path, then it selects it as the next waypath; otherwise it continues in the previously selected path. Either way, the tree expansion and search for a better path is continued on it reaches the next waypoint. This process is repeated until the goal point is reached. The main difference from the single UAV path planner is that the UAV continuously looks for neighbours and if there are any it checks for conflict. If there exists a conflict, it employs the conflict resolution strategy to generate locals paths around the conflict point.

This completes the description of the algorithm. With this algorithm, a UAV can avoid static and dynamic obstacles in a dynamic environment. Moreover, solutions for each UAV gets better
as the mission progresses.

### 3.5 Numerical Results

In this section, we present numerical results to demonstrate the performance of the proposed path planning algorithms. The objective is to compute feasible paths in real-time from starting locations to goal locations while avoiding static, pop-up and dynamic obstacles in an obstacle rich environment for single and multiple UAV systems. Towards this aim, we first use the robust RRT algorithm (see Section 3.3.4) to generate as off-line paths. Several key points will be demonstrated through numerical results and it is advocated that the robust RRT algorithm can be a potential candidate for real-time applications, especially in cluttered environments. Next, we use the robust RRT algorithm with a time window approach for path planning problems in real-time for a single UAV system. It is shown that the resulting path planner can find a close to optimal solution in real-time avoiding pop-up obstacles. Finally, the performance of the path planning algorithm for multiple UAVs with the proposed coordination strategy (see Section 3.4) is demonstrated. It is validated that the approach can avoid dynamic conflicts and guide each UAV towards its destination.

For simulations, we consider the following simple 2-D kinematic model for a fixed-wing UAV,

$$
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\psi}
\end{bmatrix} =
\begin{bmatrix}
v \cos \psi \\
v \sin \psi \\
\frac{a}{v} + \eta
\end{bmatrix},
$$

(3.2)

where, \((x, y)\) is the vehicle position (in m), \(\psi\) is the vehicle heading (in radians), \(v\) is the speed input (in m/s), \(a\) is the lateral acceleration input (in rad/s) (guidance command), and \(\eta \sim \mathcal{N}(0, 0.005)\) is a disturbance (in rad/s) acting on the heading dynamics, such as a wind disturbance. As already argued, when the turn radii of the UAVs are comparable with the width of a corridor of obstacles, through which they have to travel, it is important to consider the turn radius constraint to avoid collisions. The proposed algorithm can handle such a constraint while adding a node to the tree. We have taken advantage of this feature and made sure that each UAV transits smoothly from one waypath to another. In order to do this, we have considered guidance characteristics in the path planning algorithm and made sure that the actual trajectory of each
UAV does not collide with obstacles. We have considered the following values of parameters for simulations.

- UAV speed, $v = 13 \, m/sec$.
- Turn radius of a UAV = 40 m.
- Guidance update time 0.1 sec.
- Preparation time for finding a path initially $t_p = 1 \, sec$.
- Time to manoeuvre $t_m = 1 \, sec$.
- The width of a safe corridor $w = 20 \, m$.

Now we will demonstrate the effectiveness of the path planning algorithms using the following example scenarios. The first section is dedicated to show off-line performance of the proposed robust RRT algorithm. A number of examples are solved to demonstrate the effectiveness of the algorithm. Moreover, the performance of the proposed path planning algorithm is compared with the Visibility Line method and it is seen that the cost of the new algorithm is close to optimal. The second section shows how the robust RRT algorithm works when it operates in real-time. The closeness to the optimal solution and the ability to handle pop-up obstacles are demonstrated. Finally, in the last section we present few examples to show how the multi-UAV path planner resolves conflicts in the case of dynamic collisions.

### 3.5.1 Off-line Robust RRT

Initially, we consider a simple scenario of 50 polyhedral obstacles in a terrain of $1 \, km \times 1 \, km$ as shown in Figure 3.10. The position and size of the obstacles are randomly generated. However, the lengths and widths of the obstacles are kept above a prefixed minima. The starting and goal locations are marked by a black triangle and a magenta square, respectively, in Figure 3.10. In order to find a path, we modify Algorithm 4, and instead of expanding the tree by adding nodes for fixed time $t_p$, we run the algorithm until it finds at least 10 paths (which takes around 0.5 sec) from the starting location to the goal location. Figure 3.10 depicts the sample tree after the expansion, where the nodes of the tree are denoted with blue dots. As expected, it
can be observed that the tree almost covers the whole search space during the expansion. As argued before, a UAV may undergo disturbances, which move it off the waypath and into an obstacle. Hence, in order to avoid collision it is necessary to ensure that the path is safe with a corridor of width $w$ along it. Figure 3.11 shows the sample tree with safe corridors of width 20, branches of the tree are shown in blue whereas the corridor extension along the branch is shown in red. Next, in order to show the distribution of nodes, we have expanded the tree for quite a long time (3 sec) and plotted the nodes of the tree in Figure 3.12 in blue dots. It can be observed from the figure that the whole configuration space is covered by feasible nodes. Hence, it may be concluded that as time goes to infinity the configuration space is almost fully explored (probabilistic completeness).

Next, we have back tracked the path from the goal location to the starting location and plotted the paths along with its tree in Figure 3.13. The paths are shown in black whereas the branches of the tree are shown in blue. In Figure 3.14, the paths are shown without the tree. It can be observed that the algorithm finds feasible paths without considering their cost. This is a major drawback of sampling based algorithms. In this work, we have not only attempted to search feasible paths but also tried to find low cost paths using heuristics proposed in Section 3.3.3. Of the searched feasible paths over the period of 0.5 sec, we chose the best path for execution. The path tracking is done by using the guidance law proposed in Chapter 2. The reference path and tracked path are shown in Figure 3.15 in black and red, respectively. It can be seen from the figure that the UAV smoothly transits from one waypath to another without
overshooting and stays on the desired path except during the transition. We have also recorded the cost of each path searched during exploration. The minimum, maximum and average cost are $2979.56 \text{ m}$, $4443.60 \text{ m}$ and $4006.32 \text{ m}$, respectively. Next, we present a comparative analysis of the off-line Robust RRT algorithm with the Visibility Line method based on the computation time and cost to travel the path.

![Figure 3.12: Distribution of nodes in the cluttered environment](image1)

![Figure 3.13: Sample tree with 10 found paths that are shown in black](image2)

![Figure 3.14: Found paths from the starting location to the goal locations](image3)

![Figure 3.15: The shortest path among the found paths and tracked trajectory](image4)

**Comparison with Visibility Line Method**

In this section, we compare the robust RRT algorithm with the visibility line in terms of time and cost. We show that the proposed algorithm finds feasible paths in an obstacle rich envi-
ronment quickly and therefore has the potential to be applied in real-time. We obtain, through simulations, the computation time requirement of our algorithm and compare it with that of the visibility line method, which is an optimal and complete path planning method. Simulations show that the computation time of the visibility line method grows significantly with the number of obstacles when compared to our algorithm. This finding supports the use of our algorithm in obstacle rich environments. Next, we briefly describe the visibility line method.

Visibility Line Method

The visibility Line method (VLM) is one of the earliest for path planning. It applies to a two dimensional configuration space [56] and is a two step process. In the first stage of the planning, a visibility graph (VG) is created by searching line segments that connect a vertex of one obstacle to a vertex of another without entering the interior of any polygon. A vertex set of the VG consists of the vertices of obstacles including starting and goal locations of the UAVs and whose edges are line segments that are collision free pairs of vertices. Note that weights of edges of VG can be computed using different heuristics/methods; in our case we use Euclidean distance as a weight measure. After VG is built, graph search methods such as Dijkstra’s algorithm is used to find the shortest path from the starting position to the goal position. Although the VL method provides an optimal solution, it suffers from the drawback that the path is usually tangential to the obstacle.

To tackle this, obstacles are generally expanded to provide additional clearness. However,
this will reduce the effective configuration space and lead to non-optimal solutions. Moreover, it
does not incorporate kinematic constraints while planning. After finding a path, a path smoothening
method needs to be applied to satisfy kinematic constraints. Another drawback is that it is
computationally intensive, especially in an obstacle rich environment. If the number of vertices
of an obstacle is \( n \), then the computing time will be of the order \( O(n^3) \). One can find more about
the VL method in reference [56] and the references therein. A snapshot of a visibility graph for
a scenario of 5 rectangular obstacles in a terrain of \( 1 \, km \times 1 \, km \) is shown in Figure 3.16. Cyan
line segments are the edges of the VG. Note that these edges do not cross any obstacles. The posi-
tion and size of the obstacles, shown as black rectangles with numbers, are chosen randomly.
The shortest path from the starting location, blue triangle at (0,800), to the goal location, cyan
rectangle at (200,800), is shown in cyan. In Figure 3.17, the shortest path is shown for another
scenario which contains 25 obstacles.

Performance of the robust RRT Vs VL method in simple environments

As argued before, the RRT algorithm can provide solutions quickly as computational time
does not increase significantly with the number of obstacles. In order to verify this fact, we
compare the visibility line method with the RRT algorithm proposed in Algorithm 4. Unlike as
stated in Algorithm 4, we do not run the a time window approach, instead we expand the tree
until it includes the goal position and record the computational time to do this. The reason of
doing this is that at the moment we are not concerned about the optimality of solution, but rather
we are more concerned about the time required to compute the solution(s). The simulations are
carried out using MATLAB 7.0 on Intel Core2 Duo (2.40 GHz) with 2GB RAM.

<table>
<thead>
<tr>
<th>Number of the obstacles</th>
<th>Computation time (sec)</th>
<th>Visibility Line Method</th>
<th>Robust RRT method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
<td>Maximum</td>
<td>Averaged over 10 simulations</td>
</tr>
<tr>
<td>20</td>
<td>0.760</td>
<td>0.790</td>
<td>0.776</td>
</tr>
<tr>
<td>40</td>
<td>5.42</td>
<td>5.49</td>
<td>5.46</td>
</tr>
<tr>
<td>60</td>
<td>20.05</td>
<td>20.81</td>
<td>20.25</td>
</tr>
<tr>
<td>80</td>
<td>66.54</td>
<td>67.66</td>
<td>67.24</td>
</tr>
<tr>
<td>100</td>
<td>167.01</td>
<td>167.87</td>
<td>167.33</td>
</tr>
</tbody>
</table>
We consider five different scenarios in a terrain of $1 \, km \times 1 \, km$ with 20, 40, 60, 80 and 100 polyhedral obstacles, respectively. The starting and the target positions are kept the same and 10 trials are performed. The computation time to find paths is recorded and the results are summarized in Table 3.2. It can be observed that as the number of obstacles increases, the computational time required to find a path is increased significantly in the VL method compared to the robust RRT method. This is because the VL method tries to find an optimal solution whereas the RRT algorithm just tries to find a feasible path. Moreover, when the RRT algorithm finds a path the search is terminated. Next, we would like to present a cost analysis for the same scenarios as considered above.

While finding paths for the above described scenarios, we also recorded the cost of each path for each run and the results are summarized in Table 3.3. As the VL method finds an optimal path each time, the path cost for each run remains the same (for the same scenario). On the other hand, the RRT finds different paths in each run because of its random nature, and hence the cost of the path will also be differ. The minimum, maximum and average cost over 10 simulations are presented in Table 3.3. It is interesting to note the closeness of the minimum cost of the RRT with the optimal cost, and the average cost of RRT algorithm over 10 simulations is also not far way from the optimal cost. It is emphasized that if we run the robust RRT algorithm for a long time, we may get a solution which is close to the optimal solution. Moreover, the RRT algorithm does not take too long to find a solution. If we use the computation time available properly with a guided search, then we may get better solution in terms of cost.

<table>
<thead>
<tr>
<th>Number of obstacles</th>
<th>Visibility Line Method</th>
<th>Robust RRT method</th>
<th>Cost (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optimal cost</td>
<td>Minimum</td>
<td>Maximum</td>
</tr>
<tr>
<td>20</td>
<td>883.98</td>
<td>1012.95</td>
<td>1579.62</td>
</tr>
<tr>
<td>40</td>
<td>1242.46</td>
<td>1280.26</td>
<td>1658.09</td>
</tr>
<tr>
<td>60</td>
<td>1147.74</td>
<td>1172.15</td>
<td>1287.67</td>
</tr>
<tr>
<td>80</td>
<td>964.67</td>
<td>964.99</td>
<td>1228.32</td>
</tr>
<tr>
<td>100</td>
<td>1125.33</td>
<td>1131.24</td>
<td>1659.10</td>
</tr>
</tbody>
</table>

It can be observed from Table 3.2 that the maximum time to find a path is of the order of 150 ms. If we were to allow the tree to expand for longer with an emphasis on an optimization
heuristic, we may get close to optimal solutions. This idea is one of the contribution of the thesis and we used this idea to develop real-time robust RRT algorithms. Next, we discuss how the computation time requirement varies as the number of obstacles increases.

**Computational performance of the robust RRT algorithm in obstacle rich environments**

In this section, we analyze the computational performance of the robust RRT algorithm in cluttered environments. Similar to the last example, the task is to find a feasible path from a starting location to a goal location while avoiding static obstacles and satisfying a turn radius constraint. Four scenarios are considered and 20 trials are performed for each scenario with randomly generated starting and goal locations for each trial:

- 250 obstacles in a terrain of $5 \times 5$ km$^2$
- 500 obstacles in a terrain of $5 \times 5$ km$^2$
- 750 obstacles in a terrain of $7.5 \times 7.5$ km$^2$
- 1000 obstacles in a terrain of $10 \times 10$ km$^2$

Table 3.4 summarizes the minimum, maximum and average runtime per node and computation time required for each run. It can be noted that average time to find a path and average time per node do not increase significantly, although there is little increment in the former case. To compare rationally, it is logical to compare the average time required to add a feasible node in a tree rather than to compare the time required to find a path. This is because, as the terrain is enlarged, the algorithm has to cover more area than it was covering before. Hence, computation time required to find a path grows with the terrain size. The average time required to add a node provides a kind of measure for the expansion of the tree. If there are more nodes, there is more possibility that the algorithm will find a path quickly. Ideally, we would like to have this quantity as small as possible, so we can have more nodes for expansion and coverage. From Table 3.4, it is clear that the average time per node does not increase much with the number of obstacles. Hence, we can employ the proposed algorithm for real-time applications. Note that the computation time depends on a number of factors that include the number of obstacles, terrain size, starting and goal configurations, constraints, etc. We have to account for all of these when developing a path planner.
Table 3.4: Simulation results, cluttered environment

<table>
<thead>
<tr>
<th>Number of obstacles</th>
<th>Computation time</th>
<th>Time per Node (ms)</th>
<th>Time to find a path (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
<td>Maximum</td>
<td>Averaged over 20 runs</td>
</tr>
<tr>
<td>250</td>
<td>22.30</td>
<td>93.20</td>
<td>45.52</td>
</tr>
<tr>
<td>500</td>
<td>91.50</td>
<td>157.0</td>
<td>123.09</td>
</tr>
<tr>
<td>750</td>
<td>78.50</td>
<td>250.2</td>
<td>160.1</td>
</tr>
<tr>
<td>1000</td>
<td>107.2</td>
<td>422.8</td>
<td>226.79</td>
</tr>
</tbody>
</table>

Figure 3.18: A dense obstacle rich environment, in which the starting position is represented by a black triangle and the goal location is represented by a magenta square.

Note that the average time to find a path does not give the complete picture. This is because there could be a scenario as shown in Figure 3.18 where the algorithm may take longer because of the turn radius and robustness constraints compared to other scenarios. Figure 3.19 shows the sample tree (in blue) and the found path (in black) for the scenario shown in Figure 3.18; for which the path is found in 185 sec. Hence, an upper bound on computation time cannot be guaranteed, which restricts the applicability of the algorithm for real-time applications. To handle this kind of situation, we run the robust RRT algorithm in a time window approach, similar to the MPC framework. As the path planner update frequency is much lower than the state update cycle, the algorithm gets enough time to expand, and hence, a feasible path can be found quite easily. As a precaution, we define a critical number, and if the number of nodes...
in a tree exceeds this number then the UAV is asked to execute an emergency manoeuvre such as landing at a safe side. Note that the found path may be away from the optimal solution. As mentioned before, if the search is carried out for quite a long time with an emphasis on an optimization heuristic, the possibility of finding a close to optimal solution increases. Keeping this in mind, the algorithm continuously searches for lower cost paths using an optimization heuristic, which guides the search towards an optimal solution, to find a lower cost path.

![Figure 3.20](image1) ![Figure 3.21](image2)

Figure 3.20: An obstacle rich environment of 100 obstacles in an area of $1 \text{ km} \times 1 \text{ km}$

Figure 3.21: Tree after 1 sec (in blue), and a complete path to the goal location is not traced yet

### 3.5.2 On-line Robust RRT for a single UAV system

In this section, we demonstrate the performance of a real-time robust RRT algorithm in obstacle rich environments. Consider the scenario as shown in Figure 3.20, which contains 100 rectangular obstacles. The starting location is presented by the blue triangle and the goal location is represented by the magenta square. The task here is to compute a flyable path for a UAV in real-time from the starting location to the goal location while avoiding all static obstacles. As argued before, an upper bound on the computation time cannot be guaranteed, and hence the time required to find a complete path cannot be deterministically predicted. In order to do plan in real-time, as in Section 3.3.4, we initially run Algorithm 3 for 1 sec; and the corresponding tree is shown in Figure 3.21. It can be seen from the figure that the algorithm has not found a complete path for the given time window. As the UAV is in motion and it cannot stop, a path which should facilitate the search in finding the goal location should be chosen. In order to do this, we choose a node using a heuristic (line number 7 of Algorithm 4) proposed in Algorithm 4.
to backtrack a path. The backtracked path is shown in Figure 3.22 in blue. We process the found path to remove unnecessary nodes, which are important in expansion of the tree (as these act as intermediate nodes for the growth of the tree). The preprocessing involves removing unnecessary nodes while forming a path, during which the turn radius constraint should be maintained. The processed path is shown in black in Figure 3.22. It can be seen from the figure that the blue path has an additional node (waypoint) which is unnecessarily increasing the cost of the path.

![Figure 3.22: A path is chosen using heuristic (line number 7 Algorithm 4) to move towards the goal location. The back tracked path is shown in blue. The path is processed to remove unnecessary nodes, the processed path is shown in black.](image)

![Figure 3.23: Expansion of the tree during the tracking of the waypath, branches of the tree are shown in blue. When a path to the goal location is found, the algorithm uses optimization heuristic to find near optimal solutions. In this example, we consider that the UAV starts a little away from the starting location.](image)

Once a path is identified, to execute the next, the algorithm keeps expanding the tree while a UAV follows the path (rather waypath). The decision to select the next path must be taken before the UAV has to initiate a turn. As the UAV does not know exactly which waypath it is going to execute next, it cannot compute the point from where a turn is supposed to be initiated. Moreover, it could be possible that the UAV does not have a waypath to execute next, and then it does not know when a turn should be initiated. To tackle this issue, we use the following approach. If a UAV already has a path to follow next, we use (2.23) to compute the distance where the UAV would like to take a turn. If the distance computed using (2.23) is less than a predefined measure (which we call a buffer distance), then we use this buffer distance to compute the turning point. Having done this, we set the turning point as a root of the tree and the waypoint it is supposed to travel to as its child. This process does not allow the addition of nodes which are required to be reached with sharp turns; hence avoiding unwanted oscillations.
Note that the path is followed using the guidance law developed in Chapter 2. The tracked trajectory is shown in red.

As already said, the algorithm expands the tree while the UAV follows a path. If there is no path to a goal location, the expansion is done with emphasis on an exploration heuristic. However, if a path is found, and the search is carried out with an emphasis on an optimization heuristic to find close to optimal solutions. Figure 3.26 shows the tracked path and generated tree. It can be seen from the figure that almost the whole search space is covered. The branches of the tree reach each part of the configuration space. Hence, the algorithm has the ability to
identify a feasible path quickly. Next, when the UAV reaches the turning point, again a path to follow next is chosen using the heuristic. The processed path, along with tracked path, is shown in Figure 3.26. Note that while following the current waypath, the UAV does not search further. This is because the UAV has a direct path from the next turning point to the goal location. Once it reaches the next turning point, it follows the next waypath to reach the goal. Next, we present another scenario and show how a UAV improves the path cost of the path as time progresses.

![Figure 3.28: Tree expansion during preparation time (the expansion algorithm is run for 1 sec)](image1)

![Figure 3.29: Tree expansion while the UAV follows the waypath](image2)

**Closeness with the optimal solution**

There are several considerations for an ideal path planner including consideration of optimality, completeness and computational complexity, etc. In general, the optimal solution demands high computation effort, and hence cannot be computed in real-time. The RRT algorithm generates a quick solution but it is far from optimal due to its random exploration of the space. The task here is to generate close to optimal solutions in real-time while accounting for geometrical and differential constraints. We employ Algorithm 4 to achieve this objective here. The performance of the algorithm is evaluated using the scenario presented in Figure 3.28. Initially, the robust RRT algorithm is run for 1 sec and a path is selected for execution using a heuristic. While a UAV follows its path, the tree is expanded using the exploration heuristic to find a feasible path. Once a path is found, the optimization heuristic is used to expand to the tree with the intention of finding lower cost paths. Figure 3.29 shows the tree which was grown when the UAV was following the given waypath. Figure 3.30 shows the complete path that is followed by the UAV.
from the starting location to the goal location; the reference path is shown in black whereas the actual trajectory of UAV is shown in red.

Next, we would like to compare the cost of the path generated by the robust RRT algorithm in real-time with optimal cost. For this purpose, we have generated the optimal path using the VL method; the reference path is shown in Figure 3.31. The path cost using the robust RRT algorithm is found to be 1270.29 m whereas the optimal cost is found to be 1204.10 m. Note that although we have considered a one waypoint ahead search strategy, it is not always necessary for the search to be carried out up to the next waypoint. One can choose intermediate points as well and can find new paths for use, once the intermediate point has been reached. The closeness with the optimal cost is an added advantage to real-time capability. In addition, our solution satisfies the turn radius constraint and has a kind of robustness against mild disturbances, which are not easily achievable in the VL method. In order to check the closeness to the optimal solution in a general way, we perform the following exercise. We consider an obstacle rich scenario as shown in Figure 3.32 and choose a set of 20 randomly generated initial and goal locations. Then, for each pair of starting and goal locations, a path is computed using both the robust RRT and VL methods. The cost for each iteration and percentage difference from the optimal cost are shown Figures 3.33. It can be seen from the figure that the paths discovered by the new algorithm are not far from the optimal path. The second subplot shows the percentage difference from the optimal cost.
Handling pop-up obstacles

Obstacles are classified into two types: *a priori* known obstacles (e.g., from a terrain map) from which a trajectory can be generated before flight, and pop-up obstacles encountered in flight. A terrain map is usually used to generate initial waypoints, and a local path planning scheme such as bouncing [48] can be used to avoid pop-up obstacles. Here we deal with both types of obstacles using the proposed algorithm because it can provide solutions very quickly. It is assumed that the UAVs are mounted with onboard sensors which can detect obstacles if they are on a collision course and within the sensor range. As soon as a UAV detects an obstacle, the algorithm incorporates this information into the terrain map and starts searching for new paths for the given time window, the time allowed for expanding a tree is less than the time when a UAV starts manoeuvring. The additional steps for handling pop-up obstacles are given in Algorithm 8.

Detecting a pop-up obstacle: Step 3

In this step, a UAV checks using an on-board sensor, whether or not there is an obstacle on the flight path. In this work, we assume that the sensor, that is mounted on a UAV, can detect an obstacle at 70 m from the current position of the UAV. The sensor only senses obstacles that are in front of the UAV. However, one can use realistic sensor characteristics in this step. For simplicity, we decided to use 70 m for the range. The function $\text{CHECK\_POPUP\_COLLISION}$
Algorithm 8 Robust RRT Algorithm: Additional steps for avoiding pop-up obstacles

To handle pop-up obstacle, additional steps are introduced in Algorithm 4) in the while loop at line number 11. The while loop (at line number 11) contains now following code.

1: while UAV has not reached \( x_{next} \) do
2: \( \mathcal{T} \leftarrow \text{ROBUST RRT EXPANSION}(\mathcal{T}, x_{goal}, x_{next}, t_r) \) // Expand the tree by adding nodes (Algorithm 3) for \( t_r \) second
3: \( \text{CHECK_POPUP_COLLISION}(x_{current}, x_{next}) \) // Check for collision during the current waypath following with a pop-up obstacle if detected
4: if there is a collision with the pop-up obstacle then
5: \([t_m, x_{manoeuvering}] \leftarrow \text{TIME_TO_MANOEUVRE}(x_{current}, Obs_{popup})\) // Estimate time to manoeuvre so a UAV does not collide with the pop obstacle when it starts manoeuvering
6: \( \mathcal{T}.\text{ADD_VERTEX}(x_p) \) //Initialize tree \( \mathcal{T} \) with node at \( x_{manoeuvering} \)
7: \( \mathcal{T} \leftarrow \text{ROBUST RRT EXPANSION}(\mathcal{T}, x_{goal}, x_{manoeuvering}, t) \) // Expand the tree by adding nodes (Algorithm 3) for \( t < t_m \) second
8: if \( \text{PATH_TO_GOAL}(\mathcal{T}) \) then
9: \( P \leftarrow \text{CHOOSE_PATH_TO_GOAL}(\mathcal{T}) \) //Choose the best path to execute
10: else
11: \( P \leftarrow \text{CHOOSE_PATH_TOWARDS_GOAL}(\mathcal{T}) \) //Use a heuristic to choose a path towards the goal location
12: end if
13: \( x_{next} \leftarrow \text{NEXT WAYPOINT}(P) \) //Find next waypoint on this path
14: end if
15: end while

returns a collision state when it detects a pop-up obstacle on the flight path.

Predict time to manoeuvre and the state at manoeuvre: Step 5

As a UAV is subject to a turn radius constraint, it cannot make a turn instantly. Hence, the UAV has to start to turn before it reaches an obstacle otherwise it may collide with it. In order to avoid such a collision, we define the minimum distance from an obstacle boundary for manoeuvring. The distance is calculated assuming that a UAV has to turn at a right angle from its current direction. Using (2.23), the distance is found to be equal to the turn radius of a UAV. Moreover, as mentioned before we wish to maintain a safe corridor of width “w” around a path. Hence, the root tree should be at least \( w/2 \) away from the boundary of the obstacle. We take into account these facts and allow 1 sec (the same as the preparation time) for expanding the tree. The trajectory of the UAV after 1 sec is predicted assuming that it stays on the same course which it was following, which is also assigned as a new root of the tree. Note that as an obstacle appears on the flight path, the tree is not valid and hence we have to discard the existing tree.
The rest of the steps are similar to what we have used in Algorithm 4 and already explained before. For sake of completeness, the UAV chooses a new path after 1 sec of execution and a new \( x_{\text{next}} \) is predicted. This process keeps searching for low cost paths while following the given path. Next, we demonstrate the efficacy of the algorithm with two examples. In the first example, we introduce a pop-up obstacle several times and show that each time the proposed algorithm finds a path successfully. In the second example, we introduce a pop-up obstacle only once and check whether the algorithm tries to improve the existing path after avoiding the detected pop-up obstacle.

Figure 3.34: Path planning with handling of pop-up obstacles in a terrain of \( 1 \text{ km} \times 1 \text{ km} \) with 75 polyhedral obstacles (shown in blue). Pop-up obstacles are shown in red.

Figure 3.35: An example showing how a pop-up obstacle is avoided by generating a new path in real-time and then improving the existing path while following the current path.

Consider a scenario of 75 obstacles in a terrain of \( 1 \text{ km} \times 1 \text{ km} \) as shown in Figure 3.34. Initially, we run Algorithm 3 for 1 sec and the path found is shown in black. During the flight, it encounters a pop-up obstacle shown in red (bottom left corner). After the UAV detects the obstacle on the flight path, it runs the expansion steps of the robust RRT algorithm for 1 sec. The new found path is shown in magenta in the figure. The UAV again executes a manoeuvre and starts following the new path. During the flight, the pop-up obstacle is encountered again (just above the obstacle number 59). The new path is again searched, the tree is again expanded for 1 sec and the new found path is shown in blue. In the sequel, a third pop-up appears on the flight path and the algorithm again finds a new path, which is shown in cyan. The tracked path is shown in red. The blue triangle in the figure shows the waypoints from where the UAV
re-evaluates the path it is going to track next.

In the second example, we show that the algorithm finds less costly paths after avoiding pop-up obstacles. Figure 3.35 shows a scenario in which a UAV has to travel from a starting location (shown by a blue triangle) to a goal location (shown by a magenta square). The path generated at the start of the mission is shown in black. The pop-up obstacle, which is on the flight path, is shown in red. The UAV finds an alternative path, shown in magenta, to avoid the pop-up obstacle in the given time window. While following the first waypath after avoiding the pop-up obstacle, the UAV searches for less costly paths with emphasis on an optimization heuristic. During the search, it finds a new path, which is shown in blue. This demonstrates that the algorithm improves the quality of the path while following the waypath; hence, the approach facilitates finding of solutions which are close to optimal.

3.5.3 On-line Robust RRT for a multi-UAV system

Now we demonstrate the effectiveness of multi-UAV path planning by two examples: (i) the first example considers a scenario where three UAVs are simultaneously planning to reach their destination and they are never in conflict and (ii) the second example considers a scenario in which their paths are in conflict while planning. Initially a mission is considered in which three UAVs have to travel from their starting locations to their desired goal while avoiding given static obstacles and keeping a safe distance from other UAVs. In this exercise, we assume that if two UAVs are within $5\ m$ (a safe distance) to each other at some time $t$, then they are in conflict. Figure 3.36 shows planned and executed trajectories for the first example. Similar to a single UAV system, each UAV initially expands its tree for 1 sec and selects a branch to execute. For brevity, we have not shown the tree during and after execution. While executing the path, each UAV checks if there exists any UAV within the communication range as stated in Algorithm 7. Since there is no UAV in conflict, each UAV carries out its mission (to reach the destination) without changing its path. In Figure 3.36 planned trajectories for each UAV are shown as bold line whereas tracked trajectories are shown as dashed (‘-’) line. Next, we consider a scenario in which a conflict arises.

Consider a scenario as shown in Figure 3.37. All three UAVs have to travel from their given starting locations to their goal locations. The starting location of each UAV is shown by a star
Figure 3.36: Path planning for three UAVs in a terrain of $1 \text{ km} \times 1 \text{ km}$ with 50 polyhedral obstacles (shown in blue). Planned trajectories are shown as a bold line and tracked trajectories are shown as a dashed line. The planned trajectory and corresponding tracked trajectories are superimposed except at corners.

Figure 3.37: Path planning for three UAVs in a terrain of $1 \text{ km} \times 1 \text{ km}$ with 50 polyhedral obstacles (shown in blue). Trajectories of three UAVs are in conflict. The conflict point is shown as a black dot.

During the execution of paths, they find that their paths are in conflict; and the conflict point is shown by a black dot in Figure 3.37. If paths are in conflict, collisions may arise and the UAVs need to coordinate with each other to change their paths to avoid collisions. In order to avoid conflict, the conflict resolution strategy presented in Algorithm 5 is employed. As discussed before, the UAV with the highest token number continues on its original path while the other UAVs will have new paths to determine.

In this exercise, we assume that UAV 3 (with a read path) holds the highest token number, and hence it does not change its path. The paths of the other UAVs are changed to bypass conflict. Figure 3.38 shows how the path of UAV 2 (shown in magenta) is modified to avoid conflict. An intermediate node is introduced to obtain an alternative path that satisfies the turn radius constraint of the UAV. In order to shown that the path is trackable, we have plotted the tracked trajectory of the newly found alternative path as magenta coloured dashed line. It can be observed from the figure that the path is trackable. After changing its path, UAV 2 broadcasts its path to UAV 3 and UAV 1. Figure 3.39 shows the alternative path for UAV 1. The procedure for finding the alternative path is similar (the process is described in Algorithm 6). It can be seen from Figures 3.38 and 3.39 that UAV 3 continues its path whereas UAVs 1 and 2 change their paths to avoid a collision. Hence, dynamic collisions are avoided. Note that after avoiding
conflict, each UAV continues its mission in a decentralized manner. This means that while executing the current waypath, they keep expanding the tree as stated in Algorithm 7.

Figure 3.38: UAV 3 (the path of which is shown in blue) does not change its path whereas UAV 2 (path of which is shown in red) changes its path to avoid conflict.

Figure 3.39: After UAV 2 changes its path, UAV 1 (path of which is shown in blue) changes its path to avoid conflict.

3.6 Conclusions

In this chapter, path planning algorithms for single and multi-UAV systems have been proposed for navigation in obstacle rich environments. The single UAV path planner initially expands the tree for the given time window and chooses a path using heuristics. While following the given waypath, the tree is expanded with an emphasis on optimality of the path cost to obtain close to optimal solutions. The proposed path planner performs well and provides close to optimal solutions in real-time. In addition, it can avoid pop-up obstacles effectively. The multi-UAV path planner generates de-conflicting paths using the proposed single UAV path planner with the conflict resolution strategy in a framework that manages interaction among UAVs. The performance of the path planner is verified by various simulations and is found to be satisfactory.
Chapter 4

A Probabilistic Robust Algorithm using Rapidly-exploring Random Trees

For motion planning problems in the presence of uncertainty, it may not be possible to identify a feasible path, as this requires a trade-off between designing a conservative planner and the risk of infeasibility. This chapter presents a chance constrained rapidly-exploring random tree (CC-RRT) path planning algorithm for a single vehicle system operating in uncertain environments. We use chance constraint to limit the probability of constraint violation at each time step and augment this with the RRT algorithm to develop a real-time probabilistic robust path planner. By using RRT, the algorithm enjoys the computational benefits of a sampling-based algorithm, such as trajectory-wise constraint checking and the incorporation of heuristics, while explicitly incorporating uncertainty within the formulation. Under the assumption of Gaussian noise, probabilistic feasibility at each time step can be evaluated by checking the effect of tightening the linear constraint on the conditional mean, that can be propagated easily using closed-loop disturbance free vehicle dynamics. Simulation results show that the proposed algorithm can be used for efficient identification and execution of probabilistically safe paths in real-time.

4.1 Introduction

An autonomous vehicle has to deal with many forms of uncertainty when exposed to the real-world. Path planning for a vehicle under uncertainty may face the risk of failure due to unexpected events, such as unpredictable winds that might blow a vehicle into a building. To reduce
the risk of failure, the vehicle has to stay away from failure states. Designing a path planner for a UAV operating in real-time in uncertain environments while avoiding static and/or uncertain dynamic obstacles and staying away from risky states is a challenging problem.

An important ongoing topic in the motion planning literature is the identification of feasible paths for autonomous systems under many forms of uncertainty [58]. Such uncertainty may be categorized into four groups [62]: (i) uncertainty in the system configuration; (ii) uncertainty in the system model; (iii) uncertainty in the environmental situational awareness; and (iv) uncertainty in the future environment state. The first type of uncertainty could arise when the exact position of a vehicle is not known but is rather estimated using a stochastic model, inertial sensors and/or a Global Positioning System, that expresses the estimated position as a probability distribution, such as Gaussian distribution. The second type of uncertainty arises because of system modelling errors due to approximation and linearization. The third type of uncertainty enters through the perception or sensor systems. The fourth type of uncertainty is due to the problem of future dynamic agent states. To achieve safe and reliable path planning in realistic environments, where many or all these uncertainties may be present, it is expected that some knowledge of the uncertainties be incorporated into the planning problem.

There is a considerable body of work on stochastic path planning problems [29], [9], [12], [108], [81], [33]. Path planning for stochastic systems can be broadly categorized into two types: (i) systems where the uncertainties lie in a bounded set (non-deterministic systems); and (ii) systems where the uncertainties are described by probability distribution functions [58]. If the distribution of the disturbance is bounded, robustness can be achieved by sparing the safety margin between the failure states and the nominal states [53], [32]. If the disturbance is unbounded, probabilistic approaches can be taken up, which limit the probability of failure to a specific value [9], [81]. The probabilistic approach to uncertainty modelling has a number of advantages over a set bounded approach. Disturbances such as wind are best represented using a stochastic model rather than a set-bounded one [5]. Moreover, the state estimates are available as a probabilistic distribution specifying the mean and covariance of the state. In such cases the probabilistic feasibility can be easily evaluated. Furthermore, by specifying the probability that a path is executed successfully, the desired level of conservatism can be specified in a meaningful manner in the planning problem and then the performance against conservatism can be traded easily.
A particularly large subfield handles probabilistic uncertainty problems using Markov decision processes (MDP), partial observable Markov decision processes (POMDPs) and/or game theory. Recently, efforts have been made to solve POMDPs quickly for robotics problems [93], [27]. Despite recent advances in approximate POMDP solvers, uncertain environments still pose a significant challenge for POMDP planning. Much existing work on probabilistic uncertainty has focused exclusively on environmental sensing uncertainty, typically in the form of uncertainty maps, without considering configuration uncertainty. In the occupancy grid method approach, each cell of an occupancy grid is assigned the probability that the cell is occupied by an obstacle. Assuming that uncertainty in control and sensing is negligible, a path with a minimum expected collision cost using graph search algorithms can be found [56]. However, these approaches do not scale up well with increase in the dimension of the configuration space. Miralles and Bobi [72] represent the obstacle map as a sum of Gaussian probabilistic map and construct a minimum-probability-of-collision roadmap to guide the system via potential fields. Burns and Brock [15] use an exploration-based heuristic to traverse a probabilistic roadmap (PRM) with probabilistic feasibility. Several other formulations have been developed which maintain a probabilistic safety bound for a 2D vehicle avoiding obstacles represented by uncertain vertices [73],[34]. However, these approaches are not applicable to nonholonomic systems, that generally cannot track piecewise linear roadmap paths, and require a significant preprocessing phase for roadmap construction.

Chance constraints have been commonly applied in stochastic programming and stochastic receding horizon control (RHC) [9], [66], [83], [112], [114]. They have recently received much attention in the stochastic path planning problems because of their ability to capture a trade off between planner conservatism and the risk of infeasibility. Yan and Bitmead [114] developed a probabilistic model predictive control approach for designing an optimal sequence of control inputs subject to linear chance constraints, which ensures that the linear constraints are satisfied with a certain probability. The work is extended by Blackmore et al. [9] for probabilistic path planning, under the assumptions of Gaussian noise, to design an optimal sequence of control inputs for a linear system in a non-convex environment such that the probability of constraint violation with an obstacle can be upper bounded using a disjunction of linear chance constraint. The key step of the approach was to convert chance constraints into deterministic constraints
by constraint tightening\(^1\) and then to solve the problem using a standard deterministic optimal solver. Recently, several extensions to the above approach have been proposed by the authors [10], [12]. Concurrent work has extended the chance constraint optimization framework to consider other kinds of uncertainty, such as collision avoidance between uncertain agents [108]. However, such a formulation requires the use of computationally intensive optimization algorithms, such as mixed-integer linear programs or constrained nonlinear programs. For motion planning problems involving complex dynamics and/or high dimensional configuration spaces, the computational complexity of such an optimization algorithm may scale poorly. Lecchini-Visintini et al. [113] considered non-Gaussian uncertainty with chance-constrained control for conflict resolution in air traffic control problems. They use Monte-Carlo Markov chain to determine an approximate optimal control input through simulation-based optimization.

Probabilistic sampling of the configuration space is the most successful approach for overcoming the scalability issue. Sampling-based approaches have demonstrated several advantages for complex motion planning problems, including efficient exploration of high-dimensional configuration spaces, paths that are dynamically feasible by construction, and trajectory-wise (e.g., non-enumerative) checking of possibly complex constraints. The RRT algorithm has been demonstrated as a successful planning algorithm for complex real-world systems, such as autonomous vehicles [64]; however, it does not explicitly incorporate uncertainty. Recently, efforts have been made to extend the RRT algorithm to an uncertain environment incorporating uncertainty into the planned path [30], [71], [44]. In [71], the tree is extended with different likely conditions sampled from a probabilistic model, as in particles filters. Each vertex of the tree is a cluster of the simulated results. The likelihood of successfully executing an action is quantified and the probability of following a full path is then determined. Pepy et al. [83] recently proposed an algorithm that identifies a finite-series approximation of the uncertainty propagation in order to reduce model complexity and the resulting number of simulations per node. However, these approaches require many simulations per node to maintain a sufficiently good representation of the uncertainty, which limits its applicability when applied in real-time.

In this work, we propose an extension of the RRT algorithm to handle uncertainty in the system model and environmental situational awareness. Similar to Blackmore at el. [9], we consider chance constraint formulation for a linear system in a non-convex environment sub-

\(^1\)Constraint tightening refers to growing the constraint to account for uncertainty
ject to Gaussian noise to achieve probabilistic robustness. The chance constraint formulation is extended to handle uncertain and/or dynamic obstacles. Then we propose a chance constrained RRT (CC-RRT) algorithm to generate probabilistic robust paths for a single vehicle system in real-time. The proposed algorithm is similar to the robust RRT algorithm sample reference path, however a number of modifications are incorporated to ensure probabilistic feasibility of the actual path. The algorithm simulates trajectories using the closed loop system model with disturbance acting on the system. The feasibility of the generated trajectories are evaluated using chance constraint at each time step. Under the assumption of Gaussian noise, probabilistic feasibility at each time step can be established through simple simulation of the state conditional mean and the evaluation of the linear tightened constraints. A number of heuristics are augmented with the algorithm to obtain better performance.

The rest of the chapter is organized as follows. Section 4.2 formally presents the path planning problem. Section 4.3 presents a chance constraint formulation for path planning. In Section 4.4, the RRT algorithm is extended for environmental uncertainty using a chance constraint formulation for a single vehicle system. In Section 4.5, the performance of the chance constrained RRT for a single UAV system is demonstrated by numerical simulations. Finally, concluding remarks are presented in Section 4.6.

4.2 Problem Formulation

Consider the following discrete-time nonlinear stochastic system defined as,

$$x_{t+1} = Ax_t + Bu_t + Gw_t,$$ (4.1)

where, $x_t \in \mathbb{R}^{n_x}$ is the state vector, $u_t \in \mathbb{R}^{n_u}$ is the input vector, and $w_t \in \mathbb{R}^{n_w}$ is a disturbance vector acting on the system. The initial state is assumed to be a Gaussian random variable $x_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0})$. The disturbance $w_t$ has the known probability distribution $w_t \sim \mathcal{N}(0, \Sigma_{w_t})$.

The system itself is subject to two forms of uncertainty. There is uncertainty in the initial state $x_0$, corresponding to uncertain localization. The process noise may correspond to model uncertainty, external disturbances, or some combination of these, as long as they are independent. We assume that the covariance on the process noise is time-invariant, such that $\Sigma_{w_t} \equiv \Sigma_w$, $\forall t$. There are also constraints acting on the system. These constraints are assumed
to be decoupled, and can be represented as

$$u_t \in \mathcal{U}, \quad (4.2)$$

$$\Pr(x_t \notin \mathcal{X}_t) \leq \Delta, \quad (4.3)$$

where, $\mathcal{X}_t \equiv \mathcal{X} \setminus \{\mathcal{X}_t^1 \cup \mathcal{X}_t^2 \cup \cdots \cup \mathcal{X}_t^B\}$ and $\mathcal{U}$ is the set of feasible inputs. It is assumed that $\mathcal{X}, \mathcal{X}_t^1, \ldots, \mathcal{X}_t^B$ are convex polyhedra. The set $\mathcal{X}$ defines a set of time-invariant convex constraints acting on the state, while $\mathcal{X}_t^1, \ldots, \mathcal{X}_t^B$ represent $B$ convex obstacles to be avoided. The time dependence of $\mathcal{X}_t$ allows the inclusion of both static and dynamic obstacles. Equation (4.3) represents a probabilistic constraint on the states that implies that the probability of violation of the constraint at each time step should occur below a predefined value. In this case, we would like to limit the probability of the constraint violation below $\Delta$.

The location of each obstacle is assumed to be translationally uncertain, and is represented as

$$\mathcal{X}_{tj} = \mathcal{X}_{tj}^n + \delta_{tj} + c_j, \quad \forall \ j \in \mathbb{Z}_{1,B} \quad (4.4)$$

$$c_j \sim \mathcal{N}(0, \Sigma_{c_j}), \quad \forall \ j \in \mathbb{Z}_{1,B}, \quad (4.5)$$

where the $+$ operator denotes set translation and $\mathbb{Z}_{a,b}$ represents the set of integers between $a$ and $b$ inclusive. In this representation, $\mathcal{X}_{tj}^n$ is a convex polyhedron of known fixed shape (a nominal shape obstacle); $\delta_{tj} \in \mathbb{R}^{n_x}$ is a known translation at time step $t$; and $c_j \in \mathbb{R}^{n_x}$ is a fixed, unknown translation represented by a zero-mean Gaussian random variable. Note that (4.4) represents an obstacle of known shape on a known trajectory but subject to unknown obstacle boundary and translation. This representation therefore enables us to incorporate uncertainties both in detecting the boundary of an obstacle (corresponding to perception) and in translation (corresponding to movement of an obstacle). We assume that these uncertainties are independent.

The primary objective of the planning problem is to reach the goal region $\mathcal{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$ in minimum time,

$$t_{\text{goal}} = \inf \{t \in \mathbb{Z}_{0,t_f} \mid x_t \in \mathcal{X}_{\text{goal}}\}, \quad (4.6)$$

while ensuring the constraints (4.2)-(4.3) are satisfied for all time steps $t \in \{0, \ldots, t_{\text{goal}}\}$. In practice, since there is uncertainty in the state, we assume it is sufficient for the distribution to reach the goal region $\mathcal{X}_{\text{goal}}$. 

83
The secondary objective of the planning problem, given that at least one feasible path has been found, is to identify a path which minimizes some cost function. In this work, we would like to avoid some undesirable behaviours, such as limiting the expected value of constraint violation \( 1_{\mathcal{X}_t}(x_t) \) for the whole mission, which can be represented through \( \mathbb{E}(1_{\mathcal{X}_t}(x_t)) \). Here, \( 1_{\mathcal{X}_t}(x_t) \) is an indicator function and defined as

\[
1_{\mathcal{X}_t}(x_t) = \begin{cases} 
1 & x_t \notin \mathcal{X}_t, \\
0 & \text{otherwise,}
\end{cases}
\]  

(4.7)

and

\[
\mathbb{E}(1_{x_t \notin \mathcal{X}_t}) = \mathbb{P}(x_t \notin \mathcal{X}_t)
\]  

(4.8)

With this, the motion planning problem can now be defined.

**Problem 1** (Near Minimum-Time Motion Planning). Given the initial state \( x_0 \) and constraint sets \( \mathcal{X}_t \) and \( \mathcal{U} \), compute the input control sequence \( u_t, t \in \mathbb{Z}_{0, t_f}, t_f \in \mathbb{Z}_{0, \infty} \) that minimizes

\[
J(u) = t_{\text{goal}} + \sum_{t=0}^{t_{\text{goal}}} \mathbb{E}(1_{\mathcal{X}_t}(x_t))
\]  

(4.9)

while satisfying (4.2)-(4.3) for all time steps \( t \in \{0, \ldots, t_{\text{goal}}\} \).

### 4.3 Preliminaries

In this section, we discuss mathematical details of the chance constraint formulation to derive the conditions of probabilistic feasibility. The equivalent deterministic constraint related to probabilistic feasibility conditions is derived and the approach is extended to tackle uncertain dynamic obstacles.

#### 4.3.1 Probabilistic approach

In deterministic path planning, we predict the future state of a vehicle for a given sequence of control inputs. Then feasibility of the path is checked by evaluating the constraints acting on the system. In uncertain environments, the future state cannot be predicted exactly, however, the future distribution of the vehicle state can be predicted for a given sequence of control inputs.
In order to plan a path for the vehicle under uncertainty, we must predict the future distribution of the vehicle in order to check feasibility to ensure that the path is safe with some probability.

As mentioned before, we assume that the initial state has a Gaussian distribution, the system dynamics are linear and there is additive Gaussian noise acting on the system corresponding to model uncertainty and disturbances. Under these assumptions, the distribution of the future state is also Gaussian, $p(x_t|u_0, \ldots, u_N) \sim \mathcal{N}(\hat{x}_t, \Sigma_{x_t})$ [101]. The mean $\hat{x}_t$ and covariance $\Sigma_{x_t}$ can be represented either explicitly as

$$
\hat{x}_t = A^t \hat{x}_0 + \sum_{k=0}^{t-1} A^{t-k-1} Bu_k, \quad \forall \ t \in \mathbb{Z}_{0,N},
$$

(4.10)

$$
\Sigma_{x_t} = A^t \Sigma_{x_0} (A^T)^t + \sum_{k=0}^{t-1} A^{t-k-1} \Sigma_w (A^T)^{t-k-1}, \quad \forall \ t \in \mathbb{Z}_{0,N},
$$

(4.11)

or implicitly as

$$
\hat{x}_{t+1} = A \hat{x}_t + Bu_t, \quad \forall \ t \in \mathbb{Z}_{0,N-1},
$$

(4.11)

$$
\Sigma_{x_{t+1}} = A \Sigma_{x_t} A^T + \Sigma_w, \quad \forall \ t \in \mathbb{Z}_{0,N-1}.
$$

(4.12)

Note that (4.11) is equivalent to using the disturbance-free dynamics, i.e. (4.1) with $w_t \equiv 0$, and that (4.12) is independent of the input sequence and thus can be computed \textit{a priori}. Next, we present chance constraint formulation for path planning as in [11].

4.3.2 Evaluating probabilistic constraint for a single obstacle

This section reviews the chance constraint formulation of Ref. [11], in which all obstacles are assumed to be static and have known locations, i.e., $\delta_{tj} \equiv c_j \equiv 0$. Consider a convex static polyhedra as shown in Figure 4.1 with $n_j$ hypersurfaces. A vehicle collides with an obstacle at time $t$ if its position is within the obstacle. Mathematically, a collision with the $j$th obstacle is represented through the conjunction of the following linear inequality

$$
\bigwedge_{i=1}^{n_j} a_{ij}^T x_t < b_{ij}
$$

(4.13)

where, $x_t$ is the vehicle position at time step $t$, $a_{ij}$ is a unit outward vector normal to the line $b_{ij} = a_{ij}^T x_t$, and $n_j$ represents a number of linear constraints corresponding to each hypersurface. Here, $b_{ij}$ (refer to Figure 4.1) is positive if the unit vector $a_{ij}$ points away from the origin and is negative if it points towards the origin.
Suppose the objective is to ensure that the probability of constraint violation

$$Pr(x_t \notin \mathcal{X}_t) = Pr(x_t \in \neg \mathcal{X}_t) = Pr \left( \bigwedge_{i=1}^{n_j} a_{ij}^T x_t < b_{ij} \right),$$

(4.14)

for a given time step $t$ does not exceed $\Delta \equiv 1 - P_{\text{safe}}$, where $P_{\text{safe}}$ is the desired probability of constraint satisfaction. The chance constraint formulation allows us to impose such a constraint and ensures that the probability of failure (constraint violation) is below a pre-defined value (probability). We use this condition to limit the probability of the constraint violation below $\Delta$.

Next, we present the mathematical details of the chance constraint formulation to convert the probabilistic constraint into a tightened deterministic constraint.

The probability of constraint violation with an obstacle $j$ at time instance $t$ can be written as

$$E(1_{\neg \mathcal{X}_t}(x_t)) = Pr(x_t \notin \mathcal{X}_t) = Pr \left( \bigwedge_{i=1}^{n_j} a_{ij}^T x_t < b_{ij} \right)$$

In the case of constraint violation, it is true that

$$Pr \left( \bigwedge_{i=1}^{n_j} a_{ij}^T x_t < b_{ij} \right) \leq Pr \left( a_{ij}^T x_t < b_{ij} \right), \quad \forall i \in \mathbb{Z}_{1,n_j}.$$
As we want to limit the probability of constraint violation to $\Delta$, it is only necessary to show that one of the constraints for the obstacle is satisfied with probability less than or equal to $\Delta$.

$$\bigvee_{i=1}^{n_j} \Pr(a_{ij}^T x_t < b_{ij}) \leq \Delta. \quad (4.15)$$

To render this problem tractable for path planning algorithms, the key step is to convert the probabilistic constraints (4.15) into tightened, deterministic constraints. For the $i$th constraint of the $j$th obstacle at time step $t$ apply the change of variable

$$V_{ijt} = a_{ij}^T x_t - b_{ij}; \quad (4.16)$$

the probabilistic constraint is then

$$\Pr(V_{ijt} < 0) \leq \Delta. \quad (4.17)$$

Here, the random variable $V_{ijt}$ is a derived variable of the multivariate random variable $x_t$. The constraint $a_{ij}^T x_t - b_{ij} < 0$ is equivalent to the constraint $V_{ijt} < 0$, where $V_{ijt}$ is the minimum distance between the constraint and $x_t$, as shown in Figure 4.2. It can be shown that $\nu \sim \mathcal{N}(\hat{\nu}, \Sigma_v)$ is a univariate Gaussian random variable with mean $\hat{\nu}$ and variance $\Sigma_v$ [11], where

$$\hat{\nu} = a_{ij}^T \hat{x}_t - b_{ij}, \quad (4.18)$$
$$\Sigma_v = \sqrt{a_{ij}^T \Sigma_x a_{ij}}. \quad (4.19)$$

Using this, the constraints (4.15) can then be shown to be probabilistically satisfied, i.e., the probability of constraint violation does not exceed $\Delta$, through the modification

$$\bigvee_{i=1}^{n_j} a_{ij}^T \hat{x}_t \geq b_{ij} + \tilde{b}_{ijt}, \quad \forall \ j \in \mathbb{Z}_{1,B}, \forall \ t \in \mathbb{Z}_{0,N}, \quad (4.20)$$

$$\tilde{b}_{ijt} = \sqrt{2\Sigma_v \text{erf}^{-1}(1 - 2\Delta)}, \quad (4.21)$$

where, $\text{erf}(\cdot)$ denotes the standard error function. Here, the true state $x_t$, which is not known, is replaced with the conditional mean $\hat{x}_t$, which needs to be computed (see Section 4.3.1). The term $\tilde{b}_{ijt}$ represents the amount of deterministic constraint tightening necessary to ensure probabilistic constraint satisfaction. It can be seen that the chance constraint formulation provides a way to expand each obstacle by some distance (in each constraint direction). The amount by which each constraint is moved is a function of $\Delta$, the probability of constraint violation, and the uncertainty in $x_t$ which is expressed through $\Sigma_v$. 

87
4.3.3 Evaluating probabilistic constraint for multiple static obstacles

The last section presents the condition for probabilistic satisfaction using chance constraint for a single obstacle. In this section, we present the condition for probabilistic satisfaction for multiple obstacles. As stated before, the requirement is that the vehicle should violate the constraint $Pr(x_t \notin X_t)$ at time $t$ with the probability of at most $\Delta$.

Let failure $(x_t \notin X_t)$ be an event $F$, the probability of which we want to limit to a value less than $\Delta$. Let $O_i$ be an event corresponding to constraint violation for obstacle $j$ at time step $t$. Then, we can write the probability of failure as

$$Pr(F) = Pr(O_1 \cup O_2 \cup O_3 \cup \ldots \cup O_B)$$ \hspace{1cm} (4.22)

where,

$$F \equiv O_1 \cup O_2 \ldots \cup O_B$$ \hspace{1cm} (4.23)

This is because a violation of the constraint for any obstacle is considered as a failure. From Boole's bound, for any two events $A_1$ and $A_2$ we have

$$Pr(A_1 \cup A_2) \leq Pr(A_1) + Pr(A_2)$$ \hspace{1cm} (4.24)

Hence, we get

$$Pr(F) \leq Pr(O_1) + Pr(O_2) + \ldots + Pr(O_B).$$ \hspace{1cm} (4.25)

In order to constrain $Pr(F)$ to be less than $\Delta$, we have to limit $Pr(O_1) + \ldots + Pr(O_B)$ to be less than $\Delta$, which can be achieved by constraining the constraint violation probability of each event $O_i$ by $\frac{\Delta}{B}$, i.e., $Pr(O_i) \leq \frac{\Delta}{B}$, $\forall \ i \in \mathbb{Z}_{0,B}$. With this, it can be seen that,

$$Pr(F) \leq \sum_{i=1}^{B} \frac{\Delta}{B} = \Delta,$$ \hspace{1cm} (4.26)

so the probability of constraint violation at time step $t$ is constrained to utmost $\Delta$. One can assign higher probabilities to obstacles which are close to the vehicle and lower ones to obstacles which are far from the vehicle. In this work we have not considered such risk allocation but one can easily incorporate this assigning of different probabilities of failure to each obstacle. Note that, in order to ensure that the constraint violation probability is less than $\Delta$, we have to replace $\Delta$ by $\frac{\Delta}{B}$ in (4.21) to obtain the tightened deterministic bound.
4.3.4 Evaluating probabilistic constraint for dynamic obstacles

In this section, the chance constraint formulation is extended to allow for obstacles which are both uncertain and dynamic. In doing so, the chance constraint formulation can incorporate many other categories of uncertainty found in realistic path planning scenarios.

Consider adding the obstacle uncertainty (4.4)-(4.5) to the original chance constraint formulation. The constraints for an uncertain dynamic obstacle (4.13) can be equivalently written as

\[
\bigwedge_{i=1}^{n_j} a_{ij}^T x_t < b_{ijt}, \quad \forall t \in \mathbb{Z}_{0:j},
\]

(4.27)

where \(b_{ijt} = a_{ij}^T c_{ijt}\) and \(c_{ijt}\) is a point nominally on the \(i\)th constraint at time step \(t\). Note that \(a_{ij}\) does not depend on \(t\), this is because the obstacle shape and orientation are assumed to be fixed. The corresponding disjunctive constraints (4.27) are then

\[
\bigvee_{i=1}^{n_j} a_{ij}^T x_t \geq a_{ij}^T c_{ijt}, \quad \forall j \in \mathbb{Z}_{1:B}, \forall t \in \mathbb{Z}_{0:j}.
\]

(4.28)

As before, to ensure that the probability of constraint violation is less than or equal to \(\Delta/B\), it is only necessary to show that one of the constraints for the \(j\)th obstacle is satisfied with probability less than or equal to \(\Delta/B\):

\[
\bigvee_{i=1}^{n_j} Pr(a_{ij}^T x_t < a_{ij}^T C_{ijt}) \leq \Delta/B,
\]

(4.29)

where \(C_{ijt} = c_{ijt} + c_j\) is a random variable due to (4.4)-(4.5).

For the \(i\)th constraint of the \(j\)th obstacle at time step \(t\) apply the change of variable

\[
V_{ijt} = a_{ij}^T x_t - a_{ij}^T C_{ijt}.
\]

(4.30)

Here, the random variable \(V_{ijt}\) is a derived variable. The probabilistic constraint is again

\[
Pr(V_{ijt} < 0) \leq \Delta/B.
\]

(4.31)
The mean and covariance for $V_{ijt}$ are computed as follows:

$$\hat{v} = \mathbb{E}[V_{ijt}] = a^T_{ij} \hat{x}_t - \mathbb{E} \left[ a^T_{ij} (c_{ijt} + c_j) \right]$$

$$\Sigma_v = \sqrt{\mathbb{E} \left[ (V_{ijt} - \hat{v})(V_{ijt} - \hat{v})^T \right]}$$

$$= \sqrt{\mathbb{E} \left[ (a^T_{ij} (x_t - \hat{x}_t) - a^T_{ij} c_j) (a^T_{ij} (x_t - \hat{x}_t) - a^T_{ij} c_j)^T \right]}$$

$$= \sqrt{a^T_{ij}(\Sigma_{x_t} + \Sigma_{c_j})a_{ij}}.$$  \hspace{1cm} (4.33)

Using this, the constraints (4.28) can then be shown to be probabilistically satisfied, i.e., probability of constraint violation does not exceed $\Delta/B$, through the modification

$$\bigvee_{i=1}^{n_j} a^T_{ij} \hat{x}_t \geq a^T_{ij} c_{ijt} + \bar{b}_{ijt}, \ \forall \ j \in \mathbb{Z}_{1,n_{obs}}, \ \forall \ t \in \mathbb{Z}_{0,N},$$  \hspace{1cm} (4.34)

where, $\bar{b}_{ijt}$ is given as in (4.21) (with $\Delta$ replaced by $\frac{\Delta}{B}$) but uses the new definition of $\Sigma_v$ as in (4.33).

In summary, to extend the original chance constraint approach to include uncertain and possibly dynamic obstacles, it is sufficient to:

- Replace the right-hand side of the constraint inequality (4.27) with $a^T_{ij} c_{ijt}$, where $c_{ijt}$ tracks the deterministic trajectory of the obstacle; and

- Replace (4.18) with (4.32), and (4.19) with (4.33).

This corresponds to placing the mean along the possible dynamic trajectory of the constraint and adding the translation uncertainty covariance to $\Sigma_v$.

### 4.3.5 Evaluating probability of constraint violation

The chance constraint formulation presented above provides a framework to limit the upper bound of constraint violation at each time instant. Fortunately, it is possible to compute a more precise bound on the probability of constraint violation at each time step. As discussed above, the probabilistic constraint can be shown to be equivalent to a deterministic one as,

$$\Pr(V_{ijt} < 0) \leq \gamma \iff \hat{v} \geq f(\gamma),$$  \hspace{1cm} (4.35)
where, \( f(\gamma) \triangleq \sqrt{2\Sigma_v}erf^{-1}(1 - 2\Delta_B) \). The inverse error function increases monotonically and continuously from \(-\infty\) to \(+\infty\) over its domain \((-1, +1)\). Since \( \Sigma_v > 0 \), this implies that \( f(\gamma) \) decreases monotonically and continuously from \(-\infty\) to \(+\infty\) over its domain \((0, 1)\). As a result, there must exist some value \( \gamma \) such that \( \hat{v} = f(\bar{\gamma}) \). Exploiting the equivalence in (4.35), we have for some \( \epsilon > 0 \) (where \( |\bar{\gamma} \pm \epsilon| < 1 \)) that
\[
\bar{\gamma} - \epsilon < Pr(V_{ijt} < 0) \leq \bar{\gamma} + \epsilon.
\] (4.36)
As \( \epsilon \to 0 \), (4.35) becomes the desired equivalence
\[
Pr(V_{ijt} < 0) = \bar{\gamma} \iff \hat{v} = f(\bar{\gamma}).
\] (4.37)
Solving the above equation for \( \bar{\gamma} \) yields
\[
Pr(V_{ijt} < 0) = \frac{1}{2} \left( 1 - erf \left[ \frac{\hat{v}}{\sqrt{2\Sigma_v}} \right] \right).
\] (4.38)
Equation (4.38) provides a closed form expression of the probability of satisfying chance constraint corresponding to a hypersurface of an obstacle. Now consider the \( i \)th constraint of the \( j \)th obstacle at time step \( t \), with associated change of variable (see (4.30)-(4.33)). Let \( \Delta_{ijt}(\hat{x}, \Sigma_x) \) denote the probability that the constraint is satisfied for a Gaussian distribution with mean \( \hat{x} \) and covariance \( \Sigma_x \), then using (4.38) we have
\[
\Delta_{ijt}(\hat{x}, \Sigma_x) = \frac{1}{2} \left( 1 - erf \left[ \frac{a_{ijt}^T(\hat{x} - c_{ij})}{\sqrt{2a_{ijt}(\Sigma_{ti} + \Sigma_{cj})a_{ij}}} \right] \right).
\] (4.39)
The constraint violation probability for the \( j \)th obstacle at time step \( t \) is computed as
\[
\Delta_{jt}(\hat{x}, \Sigma_x) = \min_{i=1,...,n_j} \Delta_{ijt}(\hat{x}, \Sigma_x).
\] (4.40)
Now we compute the constraint violation probability for any obstacle at time step \( t \) as,
\[
\Delta_t(\hat{x}, \Sigma_x) = \sum_{j=1}^{B} \Delta_{jt}(\hat{x}, \Sigma_x).
\] (4.41)

### 4.3.6 Discussion

Although the primary objective is to reach the goal in minimum time while constraining the probability of violation below a pre-defined value at each time, in many practical applications it
is important to consider the probabilistic feasibility of the whole path rather than at each instant. However, the computation of the probabilistic feasibility over the entire path is intractable for all but the simplest configurations, mainly due to the fact that the random variables \( x_i, \forall t \in \mathbb{Z}_{0,t_f} \) are not independent. Therefore, we cannot explicitly compute the probability of failure of the entire mission. Fortunately, we can characterize the quality of path using \( \mathbf{E}(1_{(x_t \notin \mathcal{X}_t)}) \). With this, and using (4.8), the cost function can be re-written as

\[
J(u) = t_{\text{goal}} + \sum_{t=0}^{t_{\text{goal}}} \Pr(x_t \notin \mathcal{X}_t)
\]

\[
= t_{\text{goal}} + \sum_{t=0}^{t_{\text{goal}}} \sum_{j=1}^{B} \Delta_f(\hat{x}, \Sigma_x)
\]  

(4.42)

The second term in the cost function accounts for the number of steps in which constraint violations occur. With this, a quality of path can be characterized, both in terms of the time it takes to reach the goal and associated risk in executing such a path.

### 4.4 Probabilistic Robust Algorithm

In the last chapter, we proposed the robust RRT algorithm to generate waypaths (reference paths) assuming that the generated paths can be followed accurately and are safe against mild disturbances. However, when a vehicle operates under uncertainty, this assumption may not hold good. Moreover, there are disturbances acting on the vehicle, that cause it to deviate from its given (original) path. Furthermore, position and dynamic states are typically estimated from inertial sensors and/or global positioning data, and hence, they are not known exactly. They are usually estimated using estimation techniques which predict the distribution rather than the exact values [101], [69].

In this section, we extend the RRT algorithm to generate probabilistic robust paths in partially unknown environments. We incorporate the chance constraint formulation into the RRT algorithm and present the chance constrained RRT (CC-RRT) algorithm to generate probabilistic robust path(s) in uncertain environments [68]. The proposed algorithm evaluates the probability of feasibility of each node while expanding the tree and adds nodes only when they are found feasible.
**Chance constrained RRT**

The algorithm proposed in Chapter 3 grows a tree of waypoints which, when connected, form a feasible path from a starting location to a goal location. The algorithm does not explicitly consider various forms of uncertainties present in realistic environments and therefore may fail when it operates in practice. To achieve robustness against uncertainties, we grow the tree of state distributions and ensure that the probability of failure is less than a pre-defined value. To grow a tree of state distributions, it is necessary for the chance constrained RRT algorithm to have an accurate model of the vehicle dynamics. Similar to closed-loop RRT (CL-RRT) in [54], trajectories are propagated using the closed-loop vehicle model and feasibility is checked. Unlike CL-RRT, the CC-RRT propagates the mean the covariance of the state distribution using (4.11) and (4.12), respectively, as follows,

\[
\hat{x}_{t+k|t} = A\hat{x}_{t|t} + Bu_{t+k|t} \tag{4.43}
\]

\[
\Sigma_{x_{t+k|t}} = A\Sigma_{x_{t|t}}A^T + \Sigma_w, \tag{4.44}
\]

where \( t \) is the current system time step, \( \hat{x}_{t+k|t} \) is the conditional state mean at time step \( t+k \), and \( u_{t+k|t} \) is the input applied at time step \( t+k \). Note that the control input \( u_{t+k|t} \) is generated to follow a reference path using a control law, details of which is explained later in this section.

Figure 4.3: Diagram of the CC-RRT algorithm. Given an initial state distribution at the tree root (blue), the algorithm grows a tree of state distributions in order to find a probabilistically feasible path to the goal (yellow star). The uncertainty in the state at each node is represented as an uncertainty ellipse. Each state distribution is checked probabilistically against the constraints (cyan). If the probability of constraint violation is too high, the node is discarded (red); otherwise the node is kept (green) and may be used to grow future trajectories.

The chance constrained RRT algorithm grows the tree of state distributions originating from the initial state distribution while attempting to reach the goal region \( \mathcal{X}_{\text{goal}} \) and satisfying a min-
4.4.1 Tree expansion

The tree expansion step, used to grow the tree, is given in Algorithm 9. Similar to the robust RRT algorithm (Algorithm 3, Chapter 3), the CC-RRT algorithm samples the reference trajectory. However, the actual trajectories are predicted by running a forward simulation (predicting future state distribution) of the closed-loop system consisting of the vehicle dynamics model and the controller (as in Figure 4.4). The CC-RRT algorithm maintains two separate trees: one for the reference inputs, and one for the simulated trajectory (see Figure 4.5). The reference
trajectories (waypath) in CC-RRT are generated in an identical manner to the robust RRT algorithm. The generation of the actual trajectory is discussed later in this section. Next, we discuss the key steps of the algorithm to grow a tree of state distribution.

**Algorithm 9** CC-RRT Algorithm: Tree Expansion

**Input:** tree $\mathcal{T}$, a goal region $\mathcal{J}_\text{goal}$, time window for tree expansion $t_p$

```plaintext
while $t < t_p$ do
    $x_{\text{rand}} \leftarrow \text{RANDOM_VERTEX}();$ //Generate a random point in configuration space
    $(x_{\text{near}1}, x_{\text{near}2}, \ldots, x_{\text{near}N}) \leftarrow \text{NEAREST_VERTEX}(x_{\text{rand}}, \mathcal{T});$ //Find $N$ nearest vertices of $x_{\text{rand}}$ in tree $\mathcal{T}$
    for $Z=1$ to $N$ do
        $x_{\text{extend}} \leftarrow \text{EDGE_EXTEND}(x_{\text{near}Z}, x_{\text{rand}});$ //Extend the state $x_{\text{near}Z}$ along the direction of $x_{\text{rand}}$
        $x_{\text{parent}} \leftarrow \text{FIND_PARENT}(x_{\text{near}Z});$ //Find the parent node of $x_{\text{near}Z}$
        $(\hat{x}_{t+k|t}, \Sigma_{t+k|t}) \leftarrow \text{FIND_STATE}(x_{\text{parent}});$ //Find the mean and covariance corresponding to vertex of $x_{\text{parent}}$
        while $(\hat{x}_{t+k|t}, \Sigma_{t+k|t})$ is probabilistic feasible and $\hat{x}_{t+k|t}$ has not reached $x_{\text{extend}}$ do
            $u_{t+k|t} \leftarrow \text{SELECT_INPUT}(x_{\text{parent}}, x_{\text{extend}}, \hat{x}_{t+k|t});$ //Calculate the guidance command required to drive the vehicle on the desired path
            $(\hat{x}_{t+k+1|t}, \Sigma_{t+k+1|t}) \leftarrow \text{PROPAGATE_STATE}(\hat{x}_{t+k|t}, \Sigma_{t+k|t}, u_{t+k|t});$ //Propagate the mean and covariance using (4.43) and (4.44)
            $k \leftarrow k + 1;$
        end while
        if $\hat{x}_{t+k|t}$ has reached $x_{\text{extend}}$ then
            $\mathcal{T}.\text{UPDATE_COST_ESTIMATE}(x_{\text{extend}});$ //Estimate lower-bound cost using Euclidean norm and set upper-bound cost $+\infty$
        end if
    end for
    $\mathcal{T}.\text{ADD_VERTEX}(x_{\text{extend}});$ //Add vertex to the tree
    $\mathcal{T}.\text{SAVE_ACTUA_TRAJECTORY}(x_{\text{parent}}, x_{\text{extend}}, \hat{x}_{t+k|t}, \Sigma_{t+k|t});$ //Save actual trajectory corresponding to this branch
    $\mathcal{T}.\text{CONNECT_TO_GOAL}(x_{\text{extend}});$
    if a node is connected to $\mathcal{J}_\text{goal}$ then
        Update upper-bound cost-to-go of $x_{\text{extend}}$ and its ancestor
    end if
end while
```

**Generation of reference path: Steps 2-6**

These steps are the same as presented in the robust RRT algorithm. Similar to the robust RRT algorithm, Algorithm 9 takes a sample (line 2), identifies $N$ nearest nodes to connect to it (line 3), and generates potential candidate nodes (line 5) for tree expansion. For the sake of brevity,
the functions used in this procedure are not described again. After identifying potential nodes for extension, the method of selecting the input for following such a path is presented next.

**Prediction of vehicle trajectory: Steps 9-10**

This task involves two procedures; (i) for a given reference path (a straight line connecting $x_{\text{parent}}$ to $x_{\text{extend}}$) and current vehicle position $\hat{x}_t$, select a control input such that the vehicle follows the given reference path and (ii) simulate trajectories of the vehicle until $\hat{x}_{t+k} \in X_{\text{extend}}$ or the path becomes probabilistically infeasible. One can use a preferred approach for selecting the control input. In this work, we use combined pursuit plus LOS guidance law (as in (2.4)) to compute control inputs because of its simplicity and ease of implementation. As mentioned before the CC-RRT algorithm maintains two trees, and each node stores $\hat{x}_t$ and $\Sigma_x$ in addition to its location, and the mean and covariance are required to run a forward simulation. Given $x_{\text{parent}}$, the function $FIND\_STATE$ in step 8 retrieves the mean and covariance corresponding to the node location. These values are the initial conditions for the propagation of the mean and covariance. After computing the control input, the trajectory is simulated forwards using (4.11) and (4.12). Figure 4.6 shows a snapshot of the sampled reference path and a predicted future trajectory for a simple case using the vehicle model presented in (4.45).
Ensuring probabilistic feasible trajectory: Step 8

This is an important step of the algorithm in which probabilistic feasibility of the predicted trajectory is evaluated. While predicting the state distribution ($\hat{x}_t$ and $\Sigma_{x_t}$) at each time step $t$, the probabilistic feasibility is checked using the theoretical development presented in Section 4.3 (using (4.20) or (4.34)). The criterion for a probabilistically valid path is that the disjunction of the constraints $\bigvee_{i=1}^{n_j} Pr(a_{ij}^T x_t < b_{ij}) \leq \frac{\Delta}{B}$ should hold (at least one should be satisfied) for all $x_t$ and for all $j = 1, \ldots, B$. If the predicted path is found feasible, then an attempt is made to connect the extended node directly to $\mathcal{X}_{\text{goal}}$ at line 17. This is because the primary objective of the algorithm is to find quickly a feasible path to $\mathcal{X}_{\text{goal}}$.

In reality, it is useful to know the minimum probability of feasibility of the path rather than at a single time step. However, the computation of probabilistic feasibility over the entire path is intractable, mainly due to the fact that the random variable $x_t$, $\forall t \in \mathbb{Z}_{0,t_f}$, are not independent. Recent results have shown that the computation can be simplified for a linear system with Gaussian noise by “stacking” the states and constraints [12], but this is still computationally intensive for real-time operations of considerable duration. In this work, we limit the probability of failure $\Delta$ at each time step to make it equal to the maximum risk allowable for the path from the root to the goal. This idea provides a heuristic to assign a lower bound on feasibility of the entire path. Moreover, in order to obtain safer paths, we minimize the expected value of the constraint violation that occurs during the execution of entire path. This approach helps us to select less risky paths.

Branch and bound: Step 19

The main idea in branch and bound is to avoid growing the whole tree as much as possible but instead to grow only the most promising nodes of the tree. A promising node is identified by maintaining two estimates of optimal cost-to-go $J^*(u)(x, \mathcal{X}_{\text{goal}})$ 2 from itself at $x$ to the goal region [28]. The lower-bound cost-to-go underapproximates the cost using some Euclidean norm metric $\rho(x, \mathcal{X}_{\text{goal}})$, which ignores dynamic and/or avoidance constraints, with the speed

\footnote{The branch and bound method does not require to evaluate the exact value of $J^*(u)(x, \mathcal{X}_{\text{goal}})$ at $x$; rather it requires an estimator which should provide an underapproximtion of the cost of $J^*(u)(x, \mathcal{X}_{\text{goal}})$. In this work, we assume that $J^*(u)(x, \mathcal{X}_{\text{goal}})$ is the optimal solution of (4.42) from $x$ to $\mathcal{X}_{\text{goal}}$ and the underapproximation cost is calculated by dving Euclidean norm (between $x$ and $\mathcal{X}_{\text{goal}}$) by the speed of the vehicle.}
of the vehicle. The upper-bound cost-to-go identifies the lowest-cost path from the root to the goal through that node in the tree, taking the value $+\infty$ if no path to the goal has yet been found. The branch and bound method comes into picture when at least a path to the goal is identified. Each time a node is successfully connected to the goal on line 19, the upper-bound costs-to-go of that node and its ancestors should be updated if the new path has lower cost than those previously considered. Additionally, a branch-and-bound scheme is used to prune portions of the tree, unpromising nodes, whose lower-bound cost-to-go is larger than the upper-bound cost-to-go of an ancestor. These nodes are unpromising because none of these nodes could possibly lead to a better solution than the complete feasible solution we already have in hand, and so there is no point keeping them in the tree [28].

4.4.2 Execution loop

For environments which are dynamic and uncertain, the RRT tree may need to keep growing during path following to account for changes in situational awareness. As mentioned in the last chapter, it is not possible, a priori, to determine an upper bound for the time within which a feasible solution will be obtained. Furthermore, real-time computational requirements typically necessitate the use of tree information from previous cycles. Thus, it may be appropriate to add an execution loop around Algorithm 9, such that the tree of state distributions can continue to be expanded, even as the system executes a path within this tree. Algorithm 10 shows how the chance constrained RRT algorithm can be used to execute some portion of the tree while continuing to grow. Similar to the last chapter, we assume that a vehicle has only $t_p$ seconds, which is the preparation time for a mission, to find a probabilistic feasible path.

Algorithm 10 initializes the root of the search tree by assigning the initial mean and covariance of the vehicle’s starting position, which is the first node of the tree (line 1). Next, it runs Algorithm 9 for $t_p$ seconds to grow a tree of state distribution which satisfies chance constraint for probabilistic feasibility (line 2). The algorithm selects the best path to be executed by the system (lines 4-8), if there are paths to the goal, then the path which has the minimum cost is selected for execution. If a path to the goal is not found, then a branch is chosen for execution using the same heuristic as proposed in Algorithm 4 of Chapter 3. After selecting the best path, the vehicle estimates the time to reach the next waylocation (lines 9-10) and starts following by employing the guidance law proposed in Chapter 2 (2.4). While following the path, the growth
Algorithm 10 CC-RRT Algorithm : Execution

Inputs: a starting distribution \((\hat{x}_I, \Sigma_v)\), a goal region \(\mathcal{X}_{\text{goal}}\), the environment (obstacles map), mission preparation time \(t_p\), the maximum allowed failure probability \(\Delta\)

1: \(\mathcal{T}.\text{ADD\_VERTEX}(\hat{x}_I)\) //Initialize tree \(\mathcal{T}\) with node at \(\hat{x}_I\) and save associated \(\Sigma_v\)
2: \(\mathcal{T} \leftarrow \text{ROBUST\_RRT\_EXPANSION}(\mathcal{T}, \hat{x}_I, \Sigma_v, \mathcal{X}_{\text{goal}}, t_p, \Delta)\) // Expand the tree by adding nodes (Algorithm 9) for \(t_p\) second
3: \textbf{while} \(\hat{x}_t \notin \mathcal{X}_{\text{goal}}\) \textbf{do}
4: \textbf{if} \(\text{PATH\_TO\_GOAL}(\mathcal{T})\) \textbf{then}
5: \(P \leftarrow \text{CHOOSE\_PATH\_TO\_GOAL}(\mathcal{T})\) //Choose the best path to execute
6: \textbf{else}
7: \(P \leftarrow \text{CHOOSE\_PATH\_TOWARDS\_GOAL}(\mathcal{T})\) //Use a heuristic to choose a path towards the goal location
8: \textbf{end if}
9: \(\mathcal{X}_{\text{next}} \leftarrow \text{NEXT\_WAYLOCATION}(P)\) //Find next waylocation on this path
10: \(t_r \leftarrow \text{TIME\_TO\_GO}(\mathcal{X}_{\text{next}})\) // Estimate time to reach \(\mathcal{X}_{\text{next}}\)
11: \textbf{while} \(\hat{x}_t \notin \mathcal{X}_{\text{next}}\) \textbf{do}
12: \(\mathcal{T} \leftarrow \text{ROBUST\_RRT\_EXPANSION}(\mathcal{T}, \hat{x}_t, \Sigma_v, \mathcal{X}_{\text{goal}}, t_r, \Delta)\) // Expand the tree by adding nodes (Algorithm 9) for \(t_r\) second
13: \textbf{end while}
14: \text{Use measurements, if any, to re-propagate state distribution using (4.43) and (4.44)}
15: \(\mathcal{T} \leftarrow \text{CHECK\_PROBABILISTIC\_FEASIBILITY\_TREE}(\mathcal{T})\)
16: \textbf{end while}

of the tree is continued until the vehicle reaches the goal location. The operation of most of the functions used in Algorithm 10 have been explained previously in Chapter 3 and, for the sake of brevity, are not discussed here.

In the absence of feedback on sensor measurements (measurement update cycle of the state estimation), the covariance at each step may grow quite large. This will lead to unnecessary conversation in the path planner. Recent results have demonstrated the importance of incorporating feedback on such information in maintaining a manageable level of uncertainty [108]. Keeping this in mind, whenever measurements are available we update the state distribution, which is implemented in line 14-15. When measurements are received, the state distribution for the whole tree is updated. The function \(\text{CHECK\_PROBABILISTIC\_FEASIBILITY\_TREE}\) removes infeasible branches of the tree after re-propagation of the mean and covariance.

This explains the basic steps of our proposed algorithm to find a probabilistic robust path from a starting location to a given goal location using the chance constrained RRT algorithm augmented with some heuristics to obtain better performance.
4.5 Simulation Results

In this section, we present simulation results to demonstrate the effectiveness of the chance constrained RRT approach in efficiently computing paths for motion planning problems that satisfy probabilistic constraints. The objective is to compute probabilistic feasible paths in real-time from starting locations to goal locations in uncertain environments for a single UAV system. First, we show how without chance constraints the RRT algorithm, which does not incorporate knowledge of the uncertainty environment, may select paths which are excessively risky (paths that pass close to obstacles). Next, we use the chance constrained RRT algorithm to generate probabilistic robust paths in uncertain environments. We also show that as the lower bound on path safety $p_{safe}$ increases, the algorithm selects more conservative paths which are less likely to collide with an obstacle but require additional length/time to reach the goal. Moreover, we also show that unlike existing optimal planning approaches, our approach is scalable in the number of obstacles encountered. Similar to the previous chapter, we use a time window approach to generate paths for motion planning problems in real-time for a single UAV system. It is shown that the resulting path planner tries to continuously explore for improved solutions via an optimization heuristic. Finally, the performance of the CC-RRT algorithm is verified for a scenario that contains a dynamic uncertain obstacle.

4.5.1 System description

As discussed before, the objective here is to find a probabilistic robust path from the starting location to the goal location for a single UAV system. For simulations, we consider the following simple 2-D kinematic model for a fixed-wing UAV,

$$\dot{X} = f(X, u) + \eta,$$

where, $X_t \triangleq [x_t, y_t, \psi_t]^T$ and

$$f(X_t, u_t) \triangleq \begin{bmatrix} v \cos \psi \\ v \sin \psi \\ u \end{bmatrix}.$$

Here, $(x, y)$ is the vehicle position (in m), $\phi$ is the vehicle heading (in radians), $v$ is the speed (in m/s), $u$ is the steering input (in rad/s), and $\eta \sim \mathcal{N}(0, \sigma_u^2)$ is a disturbance (in rad/s) acting on the
heading dynamics, where, $\sigma_u = 0.0707$ and $\Sigma_u \triangleq \sigma_u^2 = 0.005$, such as a wind disturbance. The chance constrained formulation needs the system dynamics to be discretized. One relatively easy way to do this is to use the Euler integration technique [3]. The resulting discretized dynamic model can be written as

$$X_{t+1} = F(X_t, u_t) = \begin{bmatrix} x_t + dt \, v \cos \phi_t \\ y_t + dt \, v \sin \phi_t \\ \phi_t + dt \, (u_t + \eta_t) \end{bmatrix}$$

(4.45)

where, $dt = 0.1$ s is the time step taken for discretizing the system dynamics. The guidance command is calculated using (2.4). The bound on control is given as $|u| \leq u_{max} = v^2 / R_{min}$, where $v = 13$ m/s and $R_{min} = 40$ m; and hence $u_{max} = 4.25$ m/s².

As the system dynamics is nonlinear, the chance constraint formulation requires them to be linearized for the prediction. The system Jacobian matrices are obtained as,

$$F_t = \left[ \frac{\partial F(X_t, u_t)}{\partial X_t} \right] = \begin{bmatrix} 1 & 0 & -dt \, v \sin \phi_t \\ 0 & 1 & dt \, v \cos \phi_t \\ 0 & 0 & 1 \end{bmatrix}$$

(4.46)

$$G_t = \left[ \frac{\partial F(X_t, u_t)}{\partial u_t} \right] = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

(4.47)

and the covariance matrix of the noise is given as,

$$Q = G \Sigma_u G^T.$$  \hspace{1cm} (4.48)

With this, the mean and covariance of the linearized system can be computed at time step $t$ for the for future prediction as,

$$\hat{x}_{t+k|t} = F(\hat{x}_{t+k\mid t}, u_{t+k|t}),$$

(4.49)

$$\Sigma_{x_{t+k|t}} = F_t \Sigma_{x_{t+k\mid t}} F_t^T + Q.$$  \hspace{1cm} (4.50)

4.5.2 Off-line CC-RRT

Initially, we consider a simple scenario of 4 rectangular obstacles in a terrain of $0.5 \, km \times 0.5 \, km$ as shown in Figure 4.7. The position and size of the obstacles are randomly generated. However, the lengths and widths of the obstacles are kept above prefixed minima. The starting and goal locations are marked by a black triangle and a magenta square, respectively, as shown in Figure
4.7. In the results presented in this section, the tree is grown until a probabilistically feasible path to the goal is found. We assume that the initial state is known, but that measurement updates are not available at future time steps. We consider three cases for this scenario: (i) In the first case, we assume the RRT algorithm does not incorporate knowledge of uncertainty and finds a path, which is shown in black. It can be observed from Figure 4.7 that the path found without the CC-RRT algorithm goes near an obstacle. (ii) In the second case, we set $p_{safe} = 0.9$ which means the probability of constraint violation $Pr(c)$ should not exceed $Pr(c) < 1 - p_{safe} = 0.1$, the corresponding path is shown in blue. While following such a path the risk of collision with any obstacle is reduced because the path maintains a safe distance from the obstacles. (iii) In the third case, we set $p_{safe} = 0.8$ which means the probability of constraint violation $Pr(c)$ should not exceed $Pr(c) < 1 - p_{safe} = 0.2$, the corresponding path is shown in cyan. Although the path maintains considerable separation from the obstacles, it is more risky compared to the previous case.

![Figure 4.7: Closed-loop chance constrained RRT with 4 obstacles, all cases](image)

Next, we show the distribution of the nodes during the expansion of the tree. For this purpose, we grow the tree for a longer duration and plot the sample trees with uncertainty ellipses for two cases, as shown in Figures 4.8 and 4.9, respectively. Keeping the scenario unchanged, in Figure 4.8 the sample tree is grown with $p_{safe} = 0.5$ whereas in Figure 4.9 the sample tree is grown with $p_{safe} = 0.9$. We can make some key observations from the figures. The first observation is that in the first case the same tree has nodes closer to obstacles compared to the second case. This is because in the former case we have a less stringent requirement on safety compared to the second case. Secondly, it can be also observed that there are many
uncertainty ellipses in the passage for the first case whereas there are not in the second case. We will now show that the algorithm scales up well with the number of obstacles.

Figure 4.8: Sample tree with $p_{safe} = 0.5$ generated by CC-RRT algorithm for a simple environment. Each node corresponds to the state distribution mean; a $2\sigma$ uncertainty ellipse is centered at each node. The mean is shown by ‘x’ around each ellipse.

Figure 4.9: Sample tree with $p_{safe} = 0.9$ generated by CC-RRT algorithm for a simple environment. The tree shows forward simulated trajectory of a UAV using (4.45), the reference path is not shown.

**Computational Performance**

The computational complexity of the CC-RRT algorithm mainly depends on the number of obstacles. In this section, we show how the runtime of the proposed algorithm scales with the number of obstacles and advocate that the algorithm has the potential to be applied in real-time. We consider the following six scenarios and 10 trials are performed for each scenario with randomly generated starting and goal locations for each trial:

1. 5 obstacles, nominal RRT (no chance constraints)
2. 5 obstacles, chance constrained RRT with $p_{safe} = 0.5$
3. 5 obstacles, chance constrained RRT with $p_{safe} = 0.9$
4. 20 obstacles, nominal RRT (no chance constraints)
5. 20 obstacles, chance constrained RRT with $p_{safe} = 0.5$
6. 20 obstacles, chance constrained RRT with $p_{safe} = 0.9$

As mentioned earlier, the tree is grown until a probabilistically feasible path is found. Table 4.1 summarizes the minimum, maximum and average runtime per node and par path. The average runtime, to find a feasible path, has increased by a factor between 2 and 3 in each case and the average runtime per node has also increased by a similar factor. The runtime for the nominal RRT stays almost the same. This is mainly because the nominal RRT does not require forward simulation whereas the CC-RRT does. This analysis provides empirical evidence that the computational time needed for CC-RRT scales approximately linearly with the number of obstacles. Hence, the algorithm could appear to be suitable for real-time applications.

Table 4.1: Simulation results, cluttered environment

<table>
<thead>
<tr>
<th>Number of obstacles</th>
<th>$p_{safe}$</th>
<th>Computational time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time per Node (ms)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimum</td>
</tr>
<tr>
<td>5</td>
<td>N/A</td>
<td>10.80</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>110.60</td>
</tr>
<tr>
<td>5</td>
<td>0.9</td>
<td>119.80</td>
</tr>
<tr>
<td>20</td>
<td>N/A</td>
<td>6.9</td>
</tr>
<tr>
<td>20</td>
<td>0.5</td>
<td>238.5</td>
</tr>
<tr>
<td>20</td>
<td>0.9</td>
<td>243.6</td>
</tr>
</tbody>
</table>

4.5.3 On-line Implementation

In this section, we demonstrate the performance of the CC-RRT algorithm in real-time operation. Consider a scenario shown in Figure 4.10, which contains 25 rectangular obstacles in a terrain of $0.5 \text{ km} \times 0.5 \text{ km}$. The task is to compute a probabilistic robust path for a starting location to a goal location for a UAV in real-time. In order to check the robustness, we increased the noise in the control input while tracking the path. The trajectory is predicted while growing the tree with $\eta_t \sim \mathcal{N}(0, 0.005)$ and changed to $\eta_t \sim \mathcal{N}(0, 0.5)$ while following the path. Using the idea of the time window approach proposed in this work, we grow initially for 1 sec and choose a path to execute. The sample tree is shown in Figure 4.10, with a number of trajectories plotted. As discussed in Algorithm 9, the path planner samples paths in input space, the reference path
that a UAV is supposed to follow. The reference path (waypath) is shown in black.

After sampling reference paths, the algorithm predicts trajectories of the single UAV system using the closed loop system as shown in Figure 4.4. Each trajectory is checked for probabilistic feasibility using (4.20). The probabilistically feasible trajectory corresponding to the reference path is shown in blue. As described above, the tree is grown for 1 sec, then a path is selected using a heuristic presented in Algorithm 10 (step 8). The chosen path is shown in magenta. The first segment (waypath) of the tree is tracked using the guidance law proposed in Chapter 2 (see equation (2.4)), the tracked trajectory is shown in red.

While following the path, the tree is further expanded. The expanded tree during this step is shown in Figure 4.11. Once a path is found, the algorithm uses the optimization heuristic to guide the search towards near optimal solutions. The path chosen for execution is shown in magenta and the tracked path is shown in red. The UAV keeps expanding the tree while following the waypaths. Finally, the complete path and tracked trajectories from the starting location to goal location are shown in Figure 4.12.

The main difference between the algorithms (for a single UAV system) presented in Chapter 3 and those in this chapter is the way the trajectories are predicted and feasibility checked. In this Chapter, we simulate closed-loop future distributions of the trajectories and check for probabilistic feasibility. Whereas in Chapter 3, we checked feasibility of the waypath (reference...
to the controller) and use some heuristics to provide robustness. The approach proposed in this chapter provides a systematic way to deal with uncertainty, however it brings conservatism into path planning. To demonstrate this, we plot tightened constraints in Figure 4.13 during the feasibility check for one sample. The original constraints of avoiding obstacles are shown in green. During the feasibility check, the original obstacle is expanded in each direction by a factor, which is a function of the probability of failure and uncertainty in the position (refer (4.21)). The expanded boundaries are shown in red and black in the figure. The larger the uncertainty, the larger is the expansion and this reduces the effective size of the configuration space. Secondly, the smaller the value of probability (of failure), the larger the expansion, which again reduces the effective size of the configuration space. In addition to this, conservatism increases with the number of constraints (obstacles, because of the $\Delta B^2$ factor). One way to tackle this situation is to consider obstacles which are near sample(s). However, in doing this the guarantee of probabilistic robustness is lost, unlike in optimization based approaches it can be easily included in the proposed framework with additional heuristics.

Handling uncertain dynamic obstacles

In this section, we validate our proposed approach in a dynamic uncertain environment. We consider a simple scenario with a dynamic obstacle as shown in Figure 4.14. There are four
static obstacles which are shown in red and there is one uncertain dynamic obstacle; the initial location of this obstacle is shown in green. We assume that the obstacle is moving in a straight line (on a known trajectory) towards the right with a speed of $7 \text{ m/sec}$. As mentioned in Section 4.3, we assume that the shape and orientation of the obstacle remain fixed. However, there is uncertainty in translation, which means that the exact position of the obstacle is not known at each time step but the probability distribution of its position is known. Let $(c_{x0}, c_{y0})$ be a point on the $i$th constraint of the obstacle at time $t_0$, then the coordinates of the point on the $i$th constraint at time $t$ are given as,

$$
\begin{bmatrix}
  c_{xt} \\
  c_{yt}
\end{bmatrix} = \begin{bmatrix}
  c_{x0} \\
  c_{y0}
\end{bmatrix} + \begin{bmatrix}
  t v_{xo} \\
  t v_{yo}
\end{bmatrix} + c_t
$$

(4.51)

where, $v_{xo}$ is the speed of the obstacle in the positive $x$ direction (in m/s), $v_{yo}$ is the speed of the obstacle in the positive $y$ direction (in m/s) and $c_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_{cx}^2 & 0 \\ 0 & \sigma_{cy}^2 \end{bmatrix}\right)$.

In our case, we have considered $v_{xo} = 7\text{m/s}$, $v_{yo} = 0\text{m/sec}$, $\sigma_{cx} = 1$ and $\sigma_{cy} = 1$.

After setting up the environment, we run Algorithm 10 to plan a path in real-time. Unlike in a static environment, the obstacle changes its position with time. Hence, while predicting the future distribution of the vehicle the algorithm has to take into account expected future positions of the obstacle rather than its current position. This is possible because our proposed framework explicitly includes time in the formulation. While predicting future distributions of the vehicle, the procedure also involves time updates. Hence, we can predict the future position of the vehicle and can check probabilistic feasibility of the node with respect to a dynamic obstacle using (4.34) with $c_{ijt} = \begin{bmatrix} c_{xi} & c_{yi} \end{bmatrix}^T$ and

$$
P_{c_{ij}} = \begin{bmatrix}
  \sigma_{cx}^2 & 0 \\
  0 & \sigma_{cy}^2
\end{bmatrix}.
$$

Figure 4.14 shows the grown sample tree after 1 sec. The predicted position of the dynamic obstacle is shown at three instances, initially at time $t = 0$ sec (shown in green), then at time $t = 150$ sec (in a blue quadrilateral), and finally at time $t = 300$ sec (again in a blue quadrilateral). Note that we have marked the time on the bottom extreme vertices of the quadrilaterals. We have also noted the estimated time to reach each node and have plotted the time indices for a few nodes to maintain clarity. It can be seen from the figure that the feasible trajectories do not
collide with obstacles as they pass. The algorithm finds a few paths to the goal region during initial search of 1 sec, therefore the minimum cost path is chosen for execution. The algorithm keeps searching for lower cost paths while following waypaths in the tree. Figure 4.15 shows the searched and tracked paths in the uncertain environments. The tracked trajectory is shown in red. The algorithm successfully finds a probabilistically robust path in this uncertain dynamic environment.

Figure 4.14: Sample tree in the dynamic uncertain environment, static obstacles are shown in red. The green obstacle is dynamic and moving towards right at the speed of $7m/s$. The dynamic locations at two places shown in blue.

Figure 4.15: The completed reference path (black) and tracked path (red) in the uncertain environment.

4.6 Conclusions and Future Scope

In this chapter, we proposed the chance constrained RRT algorithm for probabilistic robust path planning in uncertain environments. The algorithm grows a tree of state distributions that satisfy the tightened deterministic constraint for probabilistic feasibility. The chance constraint formulation is extended to incorporate uncertain and/or dynamic obstacles. The proposed path planner performs well and the quality of solution is improved as the mission progresses. In addition, it can effectively avoid uncertain dynamic obstacles on known trajectories. The performance of the path planner is verified by various simulation studies and is found to be satisfactory. Future work will consider alternative representations of obstacle uncertainty, such as probability distribution on the obstacle vertices or obstacle process noise, as well as incorporation of better approximations of path feasibility.
Chapter 5

Cooperative Target-capturing with Inaccurate Target Information

This chapter presents distributed formation control of a multi-agent system to encircle a manoeuvring target using nonlinear control and graph theory with emphasis on consensus and cooperation. Initially, each vehicle in the formation computes a local control to maintain a desired distance from other vehicles and from the target, whose information is not completely known. Sliding mode control is used to compensate for partial target information. Then, using consensus theory with these local actions, a distributed control law is developed for the formation control. Under certain conditions, it is shown that the set of UAVs forms a target centred formation even when target information is not completely known. Numerical results are presented to validate the proposed approach.

5.1 Introduction

Recently, several researchers have studied multi-agent systems because they offer many potential advantages including parallelism, robustness, and scalability [35]. Multi-agent coordination and control has been applied to many challenging civilian and defence applications, for instance, surveillance [7], [31], search and rescue [6], [40], and fire monitoring [17], [106]. In this chapter, we focus on multi-agent target tracking such that the vehicles can maintain a target centred formation in order to capture a target. The target-capturing problem has received much attention in multi-agent research because of its ability to provide safety [43], [99]. In a target-centred
Figure 5.1: An unknown aircraft enters a secure zone. UAV A1, A2, and A3 are deployed to capture the target formation. Each vehicle maintains a constant distance from the target at a specified angle. The target-centred formation can be used to restrict the motion of a hostile target, and it can be used to escort a hostile target or a vehicle of importance to a desired location. Consider a case in which an unknown, and possibly hostile, aircraft enters a secure zone as shown in Figure 5.1. If the aircraft is not harmful, UAVs will escort the aircraft to a safe area and encourage it to land. If the target is hostile and uncooperative, for safety and security, its movement must be restricted in a desired manner. In this work, we design a cooperative target-capturing strategy to tackle two main problems: The first problem, due to limited sensor and communication ranges, is the availability of target information to only a subset of vehicles in the group. The second problem is the uncertainty in the target information, which is introduced because of sensor and imperfections in the target model.

Several researchers have investigated formation control using different approaches including leader-following [22], virtual structure approaches [90, 107] and behaviour-based methods [4]. A concise review of formation control techniques and stability issues can be found in [18]. Kim et al. [46] proposed a distributed cooperative control method to maintain a target-centred formation based on a cyclic pursuit strategy. However, in this method, all the vehicles require complete target information which is a limitation in practical scenarios. Consensus is a method, that can be used to propagate the target information from a subset of nodes to all the nodes in the network, and reach an agreement on some information of interest, in a distributed manner [39], [91], [79]. Ren [91] proposed a consensus-based formation control strategy for a multi-vehicle system where the states of each vehicle approach a common time-varying reference state, and only a subset of vehicles requires knowledge of the reference state. Kawakami et al.
[43] proposed a target-centred formation control strategy based on consensus seeking with a dynamic network topology. The authors show that the desired formation will be maintained if at least one vehicle in the group has the target information. Although the controllers developed using consensus in [89], [43] overcome the limitation of having information with all the agents in the group, the controllers do not take into account uncertainty in the target motion. However, if an upper bound on the uncertainty is known, then a sliding mode control can be used to drive the system states to a desired sliding manifold [45]. Similar to our earlier work [99], where we presented preliminary results for inaccurate velocity (first order controller) information, we now develop a second order controller and assume that at least one vehicle has the target information but relax the assumption of accurate information. Moreover, it is assumed that the team is time-varying but always connected. We use dynamic inversion and sliding mode to develop a robust distributive control law.

The distributed control law drives each vehicle to reach consensus on target speed and target heading. We show using graph theory that the vehicles are able to maintain a target centred formation if the graph of the communication topology unioned with the graph of the target-sensing topology has a directed spanning tree. The chapter is organized as follows is connected. In Section 5.2, the problem is formulated in a target centred geometry and the control objectives are defined. In Section 5.3, we first derive a target capturing control law using dynamic inversion and sliding mode approaches. Then we develop a distributed policy to capture cooperatively the target. We consider cases of both complete and incomplete target information. Section 5.4 presents simulation results to validate the proposed strategy and conclusions are drawn in Section 5.5.

5.2 Problem Formulation

Consider the mission scenario shown in Figure 5.2, where a fleet of UAVs is patrolling a high security area and a hostile target vehicle enters the secure zone. For security reasons the UAV team encircles the target and escorts it to a destination site. If the target does not cooperate with the UAV team, then the team will restrict the target motion.  

Let $r_t$ and $r_i$ be the position vectors of the target and the $i$th vehicle with respect to the

\footnote{The target aircraft is assumed to be similar to the UAVs in terms of manoeuvring and speed limitations.}
origin as shown in Figure 5.2. The objective of the \(i\)th UAV is to maintain persistently a relative stand-off \(\xi\) at an angle \(\alpha_i\) from the target.

### 5.2.1 UAV and target kinematics

We restrict our attention to a simple 2-D kinematic model for the UAVs and the target aircraft. The equations of motion of the \(i\)th vehicle at a constant altitude can be written as,

\[
\begin{bmatrix}
\dot{x}_i \\
\dot{y}_i \\
\dot{v}_i \\
\dot{\psi}_i
\end{bmatrix} =
\begin{bmatrix}
v_i \cos \psi_i \\
v_i \sin \psi_i \\
a_i \\
w_i
\end{bmatrix},
\]

(5.1)

where, \(r_i = [x_i \ y_i]^T\) is the position vector, \(v_i\) is the speed of the \(i\)th UAV, \(\psi_i\) is the heading angle, and \(u_i = [a_i \ w_i]^T\) is the control input for the \(i\)th UAV. The input \(a_i\) is the commanded acceleration and \(w_i\) is the commanded turn rate. The target aircraft kinematics is obtained by replacing \(i\) with \(t\). We assume that the position vector of the \(i\)th vehicle is directly measurable and is the output variable, and write the output kinematics as

\[
\dot{r}_t =
\begin{bmatrix}
\dot{x}_t \\
\dot{y}_t
\end{bmatrix} =
\begin{bmatrix}
v_t \cos \psi_t \\
v_t \sin \psi_t
\end{bmatrix}.
\]

(5.2)

As control variables do not appear in the expression of \(\dot{r}_i\), we differentiate to obtain

\[
\ddot{r}_i =
\begin{bmatrix}
\ddot{x}_i \\
\ddot{y}_i
\end{bmatrix} = M_i u_i, \quad M_i =
\begin{bmatrix}
\cos \psi_i & -v_i \sin \psi_i \\
\sin \psi_i & v_i \cos \psi_i
\end{bmatrix}, \quad u_i =
\begin{bmatrix}
a_i \\
w_i
\end{bmatrix} =
\begin{bmatrix}
\dot{v}_i \\
\dot{\psi}_i
\end{bmatrix}.
\]

(5.3)

### 5.2.2 Control objectives

Target capture is quantified by the following control objectives:

\(C_1\) \quad \lim_{t \to \infty} \|r_i(t) - r_t(t)\| = \xi.

(5.4)

\(C_2\) \quad \lim_{t \to \infty} \|\dot{r}_i(t) - \dot{r}_t(t)\| = 0.

(5.5)

\(C_3\) \quad \lim_{t \to \infty} \|\alpha_{i+1}(t) - \alpha_i(t)\| = \frac{2\pi}{n}.

(5.6)

These control objectives are similar to those proposed in Sharma et al. [99]. The control objectives \(C_1\) and \(C_2\) ensure that each UAV stays at its assigned position with respect to the target. The objective \(C_3\) ensures that a target centred formation is maintained.
Figure 5.2: Desired formation around the target. UAVs have to fly in formation with the target at relative distance 

\[ R_i = r_i - r_t = \xi (\cos \alpha_i \sin \alpha_i) T. \]

Angle of formation is 

\[ \alpha_i = \frac{2\pi n}{n}, \ i = 0, \ldots, n - 1. \]

Figure 5.3: The graph of \( n \) UAVs and target. The target is the root of the graph, which has a directed path to every UAV in the graph.

5.2.3 Communication and sensing topologies

The interaction topology of a network of \( n \) agents is represented using a directed graph \( G = (V, E) \) with the set of nodes represented by \( V = 1, 2, \ldots, n \) and with edges \( E \subseteq V \times V \). An edge \((i, j)\) in a directed graph denotes that vehicle \( j \) obtains information from vehicle \( i \). For an edge \((i, j)\) in a directed graph, \( i \) is the parent node and \( j \) is the child node. A directed path is a sequence of edges in a directed graph of the form \((i_1, i_2), (i_2, i_3), \ldots, i_j \in V \). A directed tree is a directed graph, where every node has exactly one parent, except for one node, called the root, which has no parent, and the root has a directed path to every other node. A directed spanning tree of \( G \) is a subset of edges in \( G \) that form a directed tree containing all nodes of \( G \). A directed graph has or contains a directed spanning tree if there exists a directed spanning tree as a subset of the directed graph. That is, there exists at least one node having a directed path to all of the other nodes. The neighbours of agent \( i \) are denoted by \( N_i = \{ j \in V : (i, j) \in E \} \). The adjacency matrix \( A = [a_{ij}] \in \mathbb{R}^{n \times n} \) of a weighted directed graph is defined as \( a_{ii} = 0 \) and \( a_{ij} > 0 \) if \((j, i) \in E \), where \( i \neq j \). The Laplacian \( L \) is defined as \( L = [l_{ij}] \in \mathbb{R}^{n \times n} \) where \( l_{ii} = \sum_{j \neq i} a_{ij} \) and \( l_{ij} = -a_{ij} \) when \( i \neq j \). A useful introduction to graph theory can be found in [94].
We represent the information exchange topology between \( n \) UAVs and the target aircraft by a graph \( G_{n+1} \) with \( n + 1 \) nodes as shown in Figure 5.3. The target aircraft is represented by the \((n + 1)\)th node of the graph, and is required to be the root of a directed spanning tree in \( G_{n+1} \). The associated Laplacian is represented by \( L_{n+1} \in \mathbb{R}^{(n+1)\times(n+1)} \). As the target is the \((n + 1)\)th node, and also the root of the graph \( G_{n+1} \), the last \((n + 1)\)th row of the Laplacian matrix is zero. Another important graph is \( G_n \) with only \( n \) UAVs as nodes and a Laplacian \( L_n \).

5.3 Theoretical Framework

In this section, we first derive a controller that achieves the objectives in Section 5.2.2 using the dynamic inversion approach [26], [45], [102] to keep a constant separation either between two UAVs or between a UAV and a target in the leader-follower framework. We first assume that a follower UAV has complete information about the leader UAV (or target). Then we derive a robust controller using the sliding mode approach [25], [45] to keep a constant separation between a UAV and a target whose complete information is not accessible by the follower UAV. Next, these two approaches are used to develop cooperative algorithms to capture a manoeuvring target for multiple UAVs with emphasis on consensus and coordination. Initially, we assume that at least one UAV has full access to the target information and derive a distributed control law to form a target centric formation by encircling the target. Later, in this chapter, the assumption of access to full target information is removed and we derive a distributed control law using sliding mode control. The theoretical details of the proposed cooperative strategy are presented next.

5.3.1 Target tracking with complete target information

In this section, we synthesise a controller for the \( i \)th UAV to keep a constant separation from either the \( j \)th UAV or the target vehicle. The objective is to design a controller \( u_i \) so that

\[
\begin{bmatrix}
x_i - x_j \\
y_i - y_j
\end{bmatrix} \to \xi \begin{bmatrix}
\cos \alpha \\
\sin \alpha
\end{bmatrix} \text{ as } t \to \infty
\]

(5.8)

where, subscript \( i \) stands for the follower UAV, subscript \( j \) stands for the leader UAV or the target, and \( \xi \) and \( \alpha_i \) are already defined in Section 5.2. We assume that both the leader and follower vehicles have similar manoeuvre capabilities. In order to achieve the above objective, we
first define an intermediate variable to represent position error between the $i^{th}$ and $j^{th}$ vehicles as,

$$r_{ij} = \begin{bmatrix} x_i - x_j \\ y_i - y_j \end{bmatrix} - \xi \begin{bmatrix} \cos \alpha_i \\ \sin \alpha_i \end{bmatrix}.$$  \hspace{1cm} (5.9)

Then, with (5.2) and using the chain rule for derivatives, the expression for $\dot{r}_{ij}$ can be written as,

$$\dot{r}_{ij} = \begin{bmatrix} \dot{x}_i - \dot{x}_j \\ \dot{y}_i - \dot{y}_j \end{bmatrix} = \begin{bmatrix} v_i \cos \psi_i - v_j \cos \psi_j \\ v_i \sin \psi_i - v_j \sin \psi_i \end{bmatrix}.$$  \hspace{1cm} (5.10)

As control does not appear in the error dynamics, we again use the chain rule and write the expression for $\ddot{r}_{ij}$ as,

$$\ddot{r}_{ij} = M_iu_i - \ddot{r}_j.$$  \hspace{1cm} (5.11)

in which we assume that $\ddot{r}_j$ is known to the $i^{th}$ vehicle. Next, the controller is synthesized such that the following stable linear error dynamics are satisfied

$$\ddot{r}_{ij} + K_\zeta \dot{r}_{ij} + K_{wn} r_{ij} = 0$$  \hspace{1cm} (5.12)

where,

$$K_\zeta = \begin{bmatrix} 2\zeta_1 w_{n1} & 0 \\ 0 & 2\zeta_2 w_{n2} \end{bmatrix}, \quad \text{and} \quad K_{wn} = \begin{bmatrix} w^2_{n1} & 0 \\ 0 & w^2_{n2} \end{bmatrix}$$

are gain matrices, with $0 < \zeta_i < 1$ and $w_{ni} > 0,i = 1,2$. Then, using the definition of $r_{ij}$ and substituting the expression for $\dot{r}_{ij}$ from (5.10) and $\ddot{r}_{ij}$ from (5.11) in (5.12), one can carry out the necessary algebra to obtain the control solution as,

$$u_i = M_i^{-1}(\ddot{r}_j - K_\zeta \dot{r}_{ij} - K_{wn} r_{ij}).$$  \hspace{1cm} (5.13)

This completes an overview of the basic steps of designing a controller using the approach of dynamic inversion. The controller designed in this section is used to compute the local control action required to keep constant relative distance with either other UAVs or with the target when employed for multi-agent formation. Note that this approach requires complete target information. In the next section, we synthesise a controller when the target information is incomplete.
5.3.2 Target tracking with incomplete target information

In this section, we synthesise a controller to keep a constant separation between the \textit{i}th UAV and a target. Here, we assume that the \textit{i}th vehicle obtains complete information from the other vehicles and gets only incomplete information from the target. The control objective remains the same as defined in the previous section. However, the follower UAV does not have complete access to target inputs. The bound on the target’s input signal is given as

\[ |M_j u_j| \leq \beta. \]  

(5.14)

The controller derived in the previous section does not account for uncertainty in the target input information. We need a controller which can assure tracking even in the presence of uncertainty. We will use sliding mode techniques because they are robust with respect to matched uncertainty. Taking \( r_{ij} \) and \( \dot{r}_{ij} \) as state variables, the error dynamics can be written in the following form

\[
\begin{bmatrix}
\dot{r}_{ij} \\
\ddot{r}_{ij}
\end{bmatrix} =
\begin{bmatrix}
0 & 1 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
r_{ij} \\
\dot{r}_{ij}
\end{bmatrix} +
\begin{bmatrix}
0 \\
M_i
\end{bmatrix} u_i +
\begin{bmatrix}
0 \\
-M_j u_j
\end{bmatrix}.
\]  

(5.15)

The objective is to select \( u_i \) such that

\[
\begin{bmatrix}
r_{ij} \\
\dot{r}_{ij}
\end{bmatrix} \rightarrow \begin{bmatrix}
0 \\
0
\end{bmatrix}.
\]  

(5.16)

To achieve this objective, we first define the sliding manifold as,

\[ s = \dot{r}_{ij} + K_s r_{ij}. \]  

(5.17)

Note that when \( s = 0 \) the motion of the system is governed by \( \dot{r}_{ij} = -K_s r_{ij} \). By choosing \( K_s > 0 \) (positive definite), it is guaranteed that \( r_{ij} \rightarrow 0 \) as \( t \rightarrow \infty \). Initial conditions do not usually lie on the sliding surface. In order to reach the sliding surface in finite time, we need a controller which drives the vehicle towards the sliding surface quickly. To design such a controller, we define the candidate Lyapunov function as

\[ V = \frac{1}{2} s^T s. \]  

(5.18)
The time derivative of the Lyapunov function \( V \) along the trajectories of the system satisfies
\[
\dot{V} = s^T \dot{s} \\
= s^T (\dot{r}_{ij} + K_s \dot{r}_{ij}) \\
= s^T (\dot{r}_{ij} + K_s \dot{r}_{ij}) \\
\dot{V} \leq s^T M_i u_i + |s|^T (K_s \dot{r}_{ij} - M_j u_j) \\
\dot{V} \leq s^T M_i u_i + |s|^T (K_s |\dot{r}_{ij}| + \beta) \tag{5.19}
\]
We choose
\[
u_i = -M_i^{-1} (K_s |\dot{r}_{ij}| + \beta + \eta) \quad \text{sgn}(s). \tag{5.20}\]
where, \( \eta \) is a small positive constant. Next, substituting the expression for \( u_i \) from (5.20) in (5.19), one can carry out the necessary algebra to obtain
\[
\dot{V} \leq -|s|^T \eta. \tag{5.21}
\]
The differential inequality (5.21) is the standard \( \eta \)-reachability condition, which implies that the sliding motion is maintained for all time. One can find extensive details on sliding mode control in [25], [45] and the references therein. The control in (5.20) exhibits chattering. To avoid chattering, we use a saturation function in place of the \( \text{sgn} \) function as described in [25], [45]. With the saturation function, the control is given by
\[
u_i = -M_i^{-1} (K_s |\dot{r}_{ij}| + \beta + \eta) \quad \text{sat}(s). \tag{5.22}\]
The controller designed in this section keeps a constant separation between the target and follower UAV even when the latter does not have complete access to target information. We use this controller to compute the local control action required to keep constant relative distance from a target when employed in a multi-vehicle formation. The next section develops a cooperative strategy to form a target centred formation using the approaches developed in this and the previous sections.

### 5.3.3 Cooperative control for target-capturing with accurate target information

As stated before, the objective is to develop a distributed control strategy for \( n \) UAVs to form and maintain a target centred formation. In this work we will not use a leader-follower strategy.
since in such topologies, information only flows from the leader to the follower. In the case where a follower experiences a disturbance, the leader cannot respond and the formation cannot be maintained. To tackle this problem, we develop an egalitarian cooperative strategy, in which information can flow in any direction, with emphasis on coordination and consensus to capture a target by encircling it.

We synthesise a distributed controller assuming that at least one of the UAVs knows the target position and velocity. The information exchange between UAVs is modelled using a graph, denoted by $G_n$. The weighting factors of information exchange between $n$ vehicles is given by the adjacency matrix $A = [a_{ij}] \in \mathbb{R}^{n \times n}$, where $a_{ij}$ is an information weighting factor representing information flowing from the $j$th UAV to the $i$th UAV. The aim is locally to compute controls $u_i, i = 1, \ldots, n$ such that the control objectives $C1, C2$, and $C3$ defined in Section 5.2.2 are achieved. Towards this direction, we first write the second order output dynamics for the $i$th UAV as

$$\ddot{r}_i = M_i u_i. \tag{5.23}$$

Assume that the $i$th UAV is supposed to maintain a constant separation $R_i$ with the target vehicle. Suppose, however, that it does not have direct access to the target’s position and velocity, but instead it has access to the $j$th UAV’s information, which is supposed to keep a constant separation $R_j$ with the target vehicle. In order to maintain the formation, we write the control required to maintain a constant separation $R_{ij} = R_i - R_j$ between the $i$th and $j$th UAVs as

$$u_{ij} = M_{ij}^{-1} \left( \dot{r}_j - K_\zeta (\dot{r}_i - \dot{r}_j) - K_w (r_i - r_j - R_{ij}) \right). \tag{5.24}$$

Note that the constant separation $R_{ij}$ is achieved with respect to the target vehicle. Similarly, if the $i$th vehicle has access to the target’s position and velocity, then the control required for the $i$th UAV to maintain a constant separation $R_i$ with the target, assuming that the $i$th UAV has full information of the target, is obtained using (5.13) as

$$u_i = M_i^{-1} \left( \dot{r}_i - K_\zeta (\dot{r}_i - \dot{r}_j) - K_w (r_i - r_j - R_i) \right). \tag{5.25}$$

We wish to find a control $u_i$ which satisfies (5.24) and (5.25), for which there are $n$ equations and one variable (an over determined system). As we know, there is no $u_i$ which satisfies all equations simultaneously. The nearest solution will be the average of the $u_i$ which satisfies these
equalities separately. Hence, the control solution is computed by taking the weighted average (to account for information exchange weighting) as follows:

\[ u_i = \frac{1}{\sum_{j=1}^{n} a_{ij} + a_{ii}} \left( \sum_{j=1}^{n} a_{ij}u_j + a_{ii}u_i \right), \]

\[ = \frac{M_i^{-1}}{|N_i|} \left[ \sum_{j=1}^{n} a_{ij} \left( \dot{r}_j - K_{\zeta} (\dot{r}_i - \dot{r}_j) - K_{w_n}(r_i - r_j - R_{ij}) \right) + a_{ii} \left( \ddot{r}_i - K_{\zeta} (\dot{r}_i - \dot{r}_i) - K_{w_n}(r_i - r_i - R_{ii}) \right) \right]. \]

(5.26)

where, \(|N_i|\) represents the in-degree of the \(i\)th UAV. Substituting the value of \(u_i\) from (5.26) to (5.23), we get

\[ \ddot{r}_i = \frac{1}{|N_i|} \left[ \sum_{j=1}^{n} a_{ij} \left( \dot{r}_j - K_{\zeta} (\dot{r}_i - \dot{r}_j) - K_{w_n}(r_i - r_j - R_{ij}) \right) + a_{ii} \left( \ddot{r}_i - K_{\zeta} (\dot{r}_i - \dot{r}_i) - K_{w_n}(r_i - r_i - R_{ii}) \right) \right]. \]

(5.27)

Next, we arrange the above equation in the following form

\[ |N_i| \ddot{r}_i - \sum_{j=1}^{n} a_{ij} \ddot{r}_j = \left( -K_{\zeta} \sum_{j=1}^{n} a_{ij}(\dot{r}_i - \dot{r}_j) - K_{w_n} \sum_{j=1}^{n} a_{ij}(r_i - r_j - R_{ij}) \right) - a_{ii} K_{\zeta} \dot{r}_i - K_{w_n}(r_i - R_{ii}) + a_{ii} (\ddot{r}_i + K_{\zeta} \dot{r}_i + K_{w_n} r_i). \]

(5.28)

Defining \(r = [r_1, \ldots, r_n]^T\) and writing (5.28) in vector form, we get,

\[ [(L_m + T) \otimes I_2] \ddot{r} = -K_1[(L_m + T) \otimes I_2] \dot{r} - K_2[(L_m + T) \otimes I_2] (r - R) + [t \otimes I_2] (\ddot{r}_i + K_{\zeta} \dot{r}_i + K_{w_n} r_i). \]

(5.29)

where, \(\otimes\) is the Kronecker product, \(L_m \in \mathbb{R}^{n \times n}\) is the Laplacian of the graph consisting of \(n\) UAVs, and \(T \in \mathbb{R}^{n \times n}\) is the target information diagonal matrix which is given as

\[ T = diag(t) = \begin{bmatrix} a_{1t} & 0 & \cdots & 0 \\ 0 & a_{2t} & \cdots & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & a_{nt} \end{bmatrix}, \quad t = [a_{1t}, \ldots, a_{nt}]^T. \]

and \(R = [R_1, \ldots, R_n]^T\). The gain matrices are defined as \(K_1 = diag(K_{\zeta}), \ K_2 = diag(K_{w_n}).\)

Equation (5.29) represents the collective second order dynamics for \(n\) UAVs. Next, we derive the conditions to achieve the desired formation around the target. To that end, we first prove the following Lemma.
Lemma 1: Rank of matrix \( L_n + T \) is \( n \) if and only if \( G_{n+1} \) has a directed spanning tree.

Proof: In order to prove that the rank of the matrix \( L_n + T \) is \( n \), we first include the target vehicle in the information graph and denote its graph as \( G_{n+1} \). The Laplacian of graph \( G_{n+1} \) can be written as

\[
L_{n+1} = \begin{bmatrix} L_{n \times n+1} & 0 \\ 0 & I_{n+1} \end{bmatrix}.
\]

The last row, \( n+1 \), corresponds to the target and there is no information flow from the agents to the target, hence all entries of the last row are zero. The last column again corresponds to the target and its entries dependant on the information flow from the target to an agent. If there is an information flow from the target to an agent \( i \), then \( L_{i \times n+1} = -a_{it} \), depends on the information weighting factor, otherwise \( L_{i \times n+1} = 0 \). With this, we can write the first \( n \) rows of the Laplacian matrix \( L_{n+1} \) as,

\[
L_{n \times n+1} = \begin{bmatrix} L_n + T \end{bmatrix} - t \begin{bmatrix} -1 \\ 0 \end{bmatrix} = L_n + T - tI.
\]

Noting that \( L_{n \times n+1} \) has \( n+1 \) columns and each of its row sums is zero, it follows that the last column of \( L_{n \times n+1} \) depends on its \( n \) columns, where \( t = (L_n + T)1_n \). This means that the last column is a linear combination of the first \( n \) columns, and hence \( \text{Rank}(L_n + T) = \text{Rank}([L_n + T | -t]) \), which is equal to \( n \) if and only if \( G_{n+1} \) has a directed spanning tree. ■

Theorem 1: (With complete target Information) When the target-capturing algorithm (5.29) is applied \( r_i(t) - r_i(t) \rightarrow R_i \) and \( \dot{r}_i(t) - \dot{r}_i(t) \rightarrow 0 \), \( \forall \ i \in [1, n] \) if and if only the graph \( G_{n+1} \) has a directed spanning tree and the root of the graph \( G_n \) has the target information.

Proof: Lemma 1 provides the condition for invertibility of matrix \( L_n + T \). Assuming the invertibility condition holds, we can write (5.29) as,

\[
\dot{r} = -[(L_n + T) \otimes I_2]^{-1} [K_1 (L_n + T) \otimes I_2] \dot{r} - [(L_n + T) \otimes I_2]^{-1} K_2 ((L_n + T) \otimes I_2) (r - R) + ((L_n + T) \otimes I_2)^{-1} \]

\[
(L_n + T) \ast 1_n \otimes I_2 \left( \ddot{r}_i + K_\xi \dot{r}_i + K_{\omega r} \mathbf{r}_i \right),
\]

\[
\ddot{r} = -K_1 \dot{r} - K_2 (r - R) + (\dot{r}_i + K_1 \dot{r}_i + K_2 \mathbf{r}_i),
\]

\[
\ddot{r} - \dot{r}_i 1_n = -K_1 (\dot{r} - \dot{r}_i 1_n) - K_2 (r - r_i 1_n - R).
\]
This represents second order error dynamics with gain matrices $K_1 > 0$ and $K_2 > 0$. From linear control system theory [78], we can say that each state $r_i$ will converge to the corresponding assigned position $R_i$ with respect to the target, i.e., $(r_i - R_i) \to 0$. Moreover, because of the second order error dynamics we will achieve $\dot{r}_i \to \dot{r}_t$. This holds true if and only if the directed graph $G_{n+1}$ has a directed spanning tree. Note that convergence does not depend on the topology of the network because invertibility of matrix $L_n + T$ does not depend on the topology as long as $G_{n+1}$ has a directed spanning tree.

Next, we assume that the exact target input information is not available and propose a cooperative control strategy to achieve the target centred formation.

### 5.3.4 Cooperative target capturing with incomplete target information

In this section, we assume that exact target information is not available and derive a robust distributed control law to capture the target using a sliding mode design discussed in Section 5.3.2. Initially, as in the last section, we assume that the $i$th UAV does not have direct access to the target. Instead, it has full access to the $j$th UAV. The control required to maintain a constant separation with the $j$th UAV is calculated as in (5.24). Next, we assume that the $i$th UAV has inaccurate target information and write the control using (5.20) to maintain a constant separation $R_i$ from the target as

$$u_{it} = M_{i}^{-1} \left( K_s |\dot{r}_i - \dot{r}_t| + \beta + \eta \right) sgn(s_i).$$

(5.31)

where $s_i = (\dot{r}_i - \dot{r}_t) + k_s (r_i - r_t)$ is a sliding manifold for the $i$th UAV. Again, there is one control and $(n + 1)$ equations to be satisfied. As before, we compute the control taking the weighted average of all controls as

$$u_i = \frac{1}{\sum_{j=1}^{n} a_{ij} + a_{it}} \left( a_{it} u_{it} + \sum_{j=1}^{n} a_{ij} u_{ij} \right),$$

$$= \frac{M_{i}^{-1} \left( K_s |\dot{r}_i - \dot{r}_t| + \beta + \eta \right) sgn(s_i)}{|N_i|} \sum_{j=1}^{n} a_{ij} \left( \ddot{r}_j - K_\zeta (\dot{r}_i - \dot{r}_j) - K_w (r_i - r_j - R_{ij}) \right).$$

(5.32)

The distributed control law in (5.32) is similar to (5.26) except the first term. The first term now is derived using sliding mode control law to account for the target uncertainty. As the control
action contains both linear and nonlinear expressions, we refer to it as a hybrid distributed control law. The distributed control law does not require the target input exactly but is still able to achieve a target centred formation. Next, we write the second order error dynamics in vector form as

\[ [(L_n + T) \otimes I_2] \ddot{r} = -K_1 [L_n \otimes I_2] \dot{r} - K_2 [L_n \otimes I_2] (r - R) \]
\[ + [T \otimes I_2] (\dot{r}_{max} + \beta + \eta) \text{sgn}(S), \]  
(5.33)

where, \( \dot{r}_{max} = \max[|r_1 - \bar{r}_1|, \ldots, |r_n - \bar{r}_n|] \) and \( S = (\dot{r} - 1 r_1) + (\dot{r} - 1 \bar{r}_1), \dot{r} = r - R. \) Now, we prove that the error dynamics in (5.33) is stable, which means that each UAV maintains the desired separation from the target. We choose a Lyapunov candidate function as

\[ V = \frac{1}{2} (\dot{r} - 1 r_1)^T L_n (\dot{r} - 1 r_1) + \frac{1}{2} S^T (L_n + T) S. \]  
(5.34)

Note that here we present the proof for the first component of \( r_i \) for simplicity. However, the proof is also valid for the second component. The time derivative of the Lyapunov function \( V \) along the trajectories satisfies

\[ \dot{V} = (\dot{r} - 1 r_1)^T L_n \dot{r} + S^T (L_n + T) S \]
\[ = \ddot{r}^T L_n (-\ddot{r} + S) + S^T ((L_n + T) \ddot{r} - (L_n + T) 1 \dot{r}_i + (L_n + T) \dot{r} - (L_n + T) 1 \dot{r}_i), \]
\[ = -\ddot{r}^T L_n \ddot{r} + S^T (L_n \ddot{r} + (L_n + T) \ddot{r} - (L_n + T) 1 \dot{r}_i + (L_n + T) \dot{r} - (L_n + T) 1 \dot{r}_i), \]  
(5.35)

Substituting the expression of \((L_n + T) \ddot{r}\) from (5.33) in (5.36), we get

\[ \dot{V} = -\ddot{r}^T L_n \ddot{r} + S^T (L_n \ddot{r} - T (\dot{r}_{max} + \beta + \eta) \text{sgn}(S) - K_2 L_n \dot{r} - K_1 L_n \dot{r} - T 1 \dot{r}_i + (L_n + T) \dot{r} - T 1 \dot{r}_i), \]
\[ = -\ddot{r}^T L_n \ddot{r} + S^T (-T (\dot{r}_{max} + \beta + \eta) \text{sgn}(S) + (I - K_1) L_n + (I - K_2) L_n \dot{r} - T 1 \dot{r}_i + T (\dot{r} - 1 r_1)). \]  
(5.36)

Choosing \( K_1 = K_2 = I \) and using the fact that \( T (\dot{r}_{max} + \beta + \eta) |S| > 0 \) and \( T (\dot{r}_{max} + \beta + \eta) |S| > S^T T((\dot{r} - 1 r_1) - 1 \dot{r}_i), \) we can write

\[ \dot{V} \leq -\ddot{r}^T L_n \ddot{r} - (\dot{r}_{max} + \beta + \eta) |S|. \]  
(5.37)

As \( L_n \geq 0 \) (due to the definition of the Laplacian matrix), we have \( \ddot{r}^T L_n \ddot{r} \geq 0, \) then if at least one UAV can see the target then \(-(\dot{r}_{max} + \beta + \eta) |S| < 0 \) and therefore

\[ \dot{V} \leq -\ddot{r}^T L_n \ddot{r} - (\dot{r}_{max} + \beta + \eta) |S| < 0. \]  
(5.38)
Hence, the second order error dynamics in (5.33) is globally asymptotically stable, which implies $r - 1r_t \rightarrow R$ and $\dot{r} \rightarrow 1r_t$. Note that for global asymptotic stability we need the matrix $L_m + T$ to be full rank, because only then the condition of positive definite will be satisfied. The distributed control law developed in this section will enable a group of UAVs to form a desired formation around the target even if the target input is not known completely. Hence, the law is robust to target input uncertainty.

5.4 Numerical results

In this section, we present simulation studies to demonstrate the performance of the proposed distributed control laws to capture a target. Consider a team of fixed wing-UAVs attempting to capture a sinusoidally manoeuvring target. The simulation parameters are

- Speed constraint on the UAV ($v_{\min} = 5m/s, v_{\max} = 25m/sec$).
- Desired distance from the target= 50 m.
- Desired angle from the target ($\alpha_i = \frac{2\pi i}{3}$).

The target vehicle is assumed to be executing a sinusoidal manoeuvre with input $U = [0 \ 0.5\sin(\frac{2\pi t}{50})]^T$ from the point $(0,0)$ with airspeed of $10 m/sec$ and initial heading of $0^\circ$.

The performance of the target-capturing cooperative strategy is demonstrated using the following example scenarios. The first scenario will show the ability to track a target by a single UAV with and without complete target information. The second scenario demonstrates the target-capturing strategy assuming that at least one of the UAVs in the group has complete target information and the graph including the target has a directed spanning tree. Finally, the second scenario is modified so that complete target information is not available to the follower UAV(s) and it is demonstrated that a target centred formation is still achieved.

5.4.1 Single vehicle target tracking

In this section, we demonstrate target tracking capability for a single UAV using the control laws proposed in Sections 5.3.1 and 5.3.2. If an individual UAV cannot follow the target or
leader UAV, then tracking a target using multiple UAVs with the same control structure cannot be envisaged. Hence, we first check the tracking performance of a single vehicle. Towards this end, we validate sinusoidal trajectory tracking with and without complete target information.

![Figure 5.4: Trajectories of the UAVs while maintaining a constant separation from the target](image)

![Figure 5.5: Tracking errors](image)

![Figure 5.6: Speed and heading convergence](image)

![Figure 5.7: Target centric trajectory in polar coordinates](image)

The follower UAV is required to maintain a distance of $\xi = 100$ m at angle $\alpha_i = 45^0$ from the target vehicle in the target’s frame of reference. The design parameters are chosen as $K_\zeta = 0.7I_2$ and $K_{w_{ji}} = 1I_2$ for the controller as in (5.13) and $K_\psi = 2I_2$ and $\eta = 2I_2$ for the controller as in (5.22). Note that the controller in (5.22) does not require the target position and velocity. Instead, the bounds on the target acceleration and turn rate need to be known. Figure 5.4 depicts the target and follower UAV trajectories and Figure 5.5 shows the associated tracking errors. It can be observed that after the initial transient the tracking errors are regulated to zero. Hence,
the formation is maintained for all time, which is clear from Figure 5.4. In order to check whether the control objective \( C_3 \) is met, we have plotted speed and heading of the target and follower UAVs in Figure 5.6. It can be observed that the airspeed and heading of the follower UAV attain the same values as the speed and heading of the target. Hence, the control objective \( C_3 \) is met. Next, we show the trajectory of the follower UAV in the target frame using polar coordinates in Figure 5.7. It can be seen from the figure that the follower UAV tracks the target maintaining the desired separation at the desired angle. The above results corroborate the theoretical developments and encourage us to use a similar approach for the tracking of multiple UAVs.

5.4.2 Target-capturing with complete target information

In the last section, we validated the controllers in (5.13) and (5.22) for a single vehicle tracking. In this section, we check the performance of the cooperative strategy for target-capturing. The information exchange topology is shown in Figure 5.8, with \( a_{ij} = 1 \) if there is an information flow from the \( j \)th UAV to the \( i \)th UAV, otherwise \( a_{ij} = 0 \). We assume that only UAV 3 has complete target information. The Laplacian matrices of the graphs, excluding the target (denote as \( L_n \)) and including the target (denoted as \( \bar{L}_n \)) are given by

\[
L_n = \begin{bmatrix}
2 & -1 & 0 & -1 \\
0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1
\end{bmatrix}, \quad \bar{L}_n = \begin{bmatrix}
2 & -1 & 0 & -1 \\
0 & 1 & -1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1 & 1
\end{bmatrix}
\]
The rank of $L_n$ is three, whereas the rank of $\bar{L}_n$ is four. The full rank of matrix $\bar{L}_n$ satisfies the condition of target-capturing in Lemma 1. As the topology of the UAVs has a directed spanning tree and the root of the graph has access to the target, the condition of target-capturing is met, as evaluated by the rank of $\bar{L}_n$ (which is full rank). It can also be observed from Figure 5.8 that all the UAVs get information of the target through UAV 3. For example, UAV 1 gets information of the target through UAV 2. Note that here we do not mean that the target information is passing through the link as it does in a communication link.

Next, we again assume that the target is manoeuvring in a sinusoidal manner with the same initial conditions as considered in the last section. The task is to capture the target by satisfying the control objectives mentioned in Section 5.2.2. These are achieved by applying the distributed control law proposed in (5.26). The simulations are carried out with the following initial conditions for follower UAVs.

Table 5.1: Initial conditions

<table>
<thead>
<tr>
<th>UAV ID</th>
<th>x</th>
<th>y</th>
<th>v</th>
<th>$\psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>50</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-50</td>
<td>50</td>
<td>8.5</td>
<td>$\frac{\pi}{4}$</td>
</tr>
<tr>
<td>3</td>
<td>-50</td>
<td>-50</td>
<td>9</td>
<td>$\frac{\pi}{2}$</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>-50</td>
<td>9.5</td>
<td>$\frac{3\pi}{4}$</td>
</tr>
<tr>
<td>Target</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.9 depicts the target and UAV trajectories around the target. The formation geometry is plotted in Figure 5.9 at five instances in magenta colour. It can be observed that the initial
formation does not satisfy the target-capturing conditions. Then, the cooperative strategy drives each UAV such that the desired separation at the given orientation with the target is maintained. The formation geometry at the second instance in Figure 5.9 shows the desired formation around the target. It can be further noted that the formation geometry is maintained throughout the mission. The associated formation error for each UAV is shown in Figure 5.10. As the follower UAVs start from random positions, there is a relatively large formation error at the beginning. The formation error for each UAV quickly settles to zero which results in the desired formation around the target. It can be noticed that the formation error settles to zero in around 15 sec, which is similar to what a single UAV takes to track the target with a constant separation. The speed and heading convergence are shown in Figure 5.11. Each UAV attains the same speed and heading with respect to the target and maintains these for all time. This means that they reach a consensus using the distributed law. Figure 5.12 shows the target centred trajectory for each UAV. It can be noted that each UAV maintains the desired separation at the given orientation to capture the target.

5.4.3 Target-capturing with incomplete target information

In the last section, we assume that complete target information was available to each one of the UAVs to achieve the target centred formation. In this example, we assume that only one UAV has the target information. Moreover, it does not have complete access to the target input; instead, it has a bound on the target input. To capture the target, we employ the robust distributed
control law proposed in (5.32), which is a combination of a linear policy (a dynamic inversion approach) and a nonlinear policy (sliding mode control). We perform simulations with the same initial conditions (both for the target and the UAVs) and information exchange topology as considered in the last section. Figure 5.13 shows trajectories of the target and follower UAVs around the target. The dashed magenta lines show the formation geometry around the target. The UAVs start with a random formation and achieve the desired formation quickly. The formation tracking error for each UAV is shown in Figure 5.14. It can be observed again that the formation errors settle to zero in time similar to that in the last example. The speed and heading convergence are shown in Figure 5.15. The convergence in this case confirms that all the UAVs arrive at a consensus quickly using the robust distributed control law (5.32). The target centred formation is depicted in the polar plot in Figure 5.16. It can be observed again that each vehicle maintains the desired separation at the given orientation to capture the target.

Figure 5.13: Trajectory of the UAVs while maintaining a constant separation with the target

Figure 5.14: Tracking errors

5.5 Conclusions

This chapter presents distributed formation control of a multi-agent system required to capture an aggressively manoeuvring target using either a dynamic inversion approach or sliding mode control. There is an emphasis on consensus and cooperation. The proposed method integrates a consensus algorithm with a robust controller in order to capture a target with incomplete information. The cooperative strategy in conjunction with the controller enables a fleet of UAVs to control the target movement by restricting its trajectory. Hence, safety of a zone is assured.
with the proposed method. The simulation results demonstrate the efficacy of the approach. In this work, the target is assumed to be similar to the UAVs in terms of manoeuvring and maximum speed capabilities. Designing a controller to capture a more agile and faster target than the UAVs will be a challenging problem for future work. Another challenging problem will be to capture multiple targets with a cooperative approach.
Chapter 6

Conclusions and Future Scope

This Chapter summarizes the contributions of the thesis and provides suggestions for future research. Initially we outline the principle contributions of each chapter; we then discuss possible new avenues for research.

In Chapter 2, combining pursuit guidance with a line-of-sight guidance law, a novel guidance law was proposed for path following under cluttered and/or windy conditions. The stability conditions of the linearized system are provided. Performance of the guidance law to follow standard manoeuvres like straight lines, circular paths or combinations of both are validated through numerical simulations.

In Chapter 3, a real-time path planning strategy using Rapidly-exploring Random Trees was proposed for single and multi-UAV systems in obstacle rich environments. The underlying idea was to use a time window approach to find solutions for the given time. While following the generated path in the given time window, the search is continued to find a complete path or to improve the existing solution. The algorithm scales well with the number of obstacles and effectively avoids pop-up obstacles. Next, a multi-UAV path planner was developed using a single UAV path planner in a framework that manages interaction among UAVs and uses a coordinate strategy to resolve conflicts. The multi-UAV path planner allows each vehicle to operate in a decentralized manner when they are not within communication range, therefore each UAV can improve its path when are not in communication range. The performance of the path planner is verified by various numerical simulations and is found to be satisfactory.

In Chapter 4, a probabilistic approach for path planning is developed to trade off between
planner conservatism and risk of infeasibility, especially for systems subject to many or unbounded forms of uncertainty. The approach uses explicit constraints to determine the feasibility of a node by computing the probability of failure, which provides the operator with direct control over the level of conservatism in the planning approach. We have demonstrated the approach through numerical simulation, it is scalable with increase in the number of obstacles and it allows for efficient computation of safe paths even in heavily cluttered environments.

Chapter 5 presents a practical application of multi-agent systems in which a group of UAVs captures a target by forming a target centred formation. Nonlinear control theory, dynamic inversion approach and a sliding mode control scheme are used to derive the local control action. This local control action is then used to compute a distributed control law for each vehicle to reach a consensus on the target information. We have proved using graph theory that the UAVs are able to maintain the target centred formation given that the target information is available to at least one of the vehicles. To account for target uncertainty, the controller is augmented with a sliding mode controller.

The studies carried out in this thesis on various applications of single and multi-agent systems open up avenues for further research. We will discuss some of them below.

In Chapter 3, we have resolved conflicts assuming that UAVs exchange information between themselves. There is also the possibility of adversary obstacles being present in the environment which would the need to be considered for safe travel between locations. Secondly, we have not tested the path planning algorithm on real systems. These remain future areas of research.

In the probabilistic path planning approach (Chapter 4), we have not explicitly considered the probability of the failure of the mission. In many practical applications, it may be useful to limit the probability of failure throughout the mission rather than at each time instant. This is an important direction for further research.

In Chapter 5, to capture a target we have defined a target centred formation a priori, which might not always be achievable or optimal. This aspect needs to be studied to obtain truly optimal target capturing strategies. The analysis of the behaviour of an adaptive topology in the presence of noise and delays in information is another important direction for further research.

Finally, some of the contributions of the thesis have already been published in the following
[50], [49], [51], [68], [86], [99].
Appendix A

A Robust Target Tracking Algorithm

In this appendix, we summarize the results of a seedcorn project “Novel experiments of real-
time reconfiguration of networked systems on collaborative missions in uncertain environ-
ments”, supported by the EPSRC/BAE Systems funded project NECTISE - Network Enabled
Capability Through Innovative Systems.

The research objectives were:

1. To configure ConSERT \(^1\) and to assess its appropriateness as a demonstration environment
   for network enabled systems.

2. To test and improve real-time task allocation and trajectory assignment algorithms.

3. To investigate issues related to the formation, communication and reconfiguration of col-
laborative systems (in this case, robots).

A.1 Introduction

The increasing use of autonomous vehicles (AV) in target tracking and formation keeping tasks
has introduced the need to improve sensor accuracy and performance. These sensors serve the
purpose of information acquisition and enable the vehicle to know about its external environ-
ment. Once information about the external world is known, appropriate decisions are taken by

\(^1\)ConSERT stands for Configurable Systems Engineering Research Tool and is an experimental laboratory housed of the
BAE Systems/Loughborough University Systems Engineering Innovation Centre (SEIC). It is used to demonstrate new results
in systems engineering.
vehicles to complete the mission at hand. Detection of sensor faults and corrective action is therefore becoming an integral part of modern AVs.

Sensors are vulnerable to internal faults and adverse external conditions such as weather and physical obstacles. Sensory information can also be corrupted by Gaussian or non-Gaussian noises. Kalman filters and other schemes are available in the literature for sensor fault identification. In practice, such schemes will have to be incorporated with backup sensors or a second information source in order to obtain correct information and to achieve fault tolerance. On the other hand, in many circumstances, military as well as civil, a group of vehicles is required to operate in a cooperative manner to accomplish the mission. In this exercise, we make use of cooperation between vehicles to deal with sensor failures and with erroneous sensory data. The algorithm developed and tested in the experiment shares and updates information obtained by each individual vehicle and makes judicious decisions to guide the movement of all the vehicles.

Multi-agent target tracking, which is the mission considered, is the same in spirit as multi-sensor target tracking, which is already a popular topic in the literature, e.g. [103]. Among many target tracking schemes, we are particularly interested in decentralized target tracking combined with fault detection because of its flexibility, cost-effectiveness and robustness. Among decentralized target tracking schemes, it is popular to use consensus type algorithms. These algorithms basically allow each agent (sensor, AV, etc) to gather information from its immediate neighbours and to update its information, so that every agent agrees on common or very similar information. Depending on the final common information $f$, these algorithms are called average-consensus, median-consensus, or $f$-consensus algorithms. Several recent papers propose decentralized consensus algorithms in Gaussian noisy environments and use Kalman-filter type estimation rules, e.g. [92], [105], or low-pass or high-pass filters, e.g. [80], [104].

The main aim of the present project is to demonstrate the efficiency of a fault detection and data fusion algorithm developed in [47]. For this purpose, the ConSERT robots available at SEIC are used to conduct experiments. Under mild assumptions the proposed decentralized scheme in [47] is capable of almost detecting faulty sensors, even if half of the neighbouring sensors are faulty. The scheme uses two measures, $d_{ij}(t)$ and $\Delta d_{ij}^{\Delta t}$, where $d_{ij}(t)$ is the difference between the values of two sensors $i$ and $j$ at time $t$ and $\Delta d_{ij}^{\Delta t}$ is the change of $d_{ij}(t)$ over a fixed time period $\Delta t$. If more than half of the neighbouring sensors $j \in N_i$ are such that both
measures are less than predefined threshold values, then the sensor $i$ is diagnosed as good and is subsequently used for diagnosing other sensors as good or faulty. Although this scheme is claimed to be probabilistically attractive, it can be noted that the two measures are not sufficient for detecting a group of faulty sensors all together in a faulty zone. For instance, one can easily consider a situation in which a faulty sensor $i$ has its neighbouring sensors $j$ all faulty, and therefore $d_{ij}(t) = \Delta d_{ij} = 0$ for a certain period of time and diagnosing the faulty sensor as good. The scheme in [47] is decentralized but guarantees perfect fault detection and isolation if there exist any faulty sensors, so that the central objective of target tracking is successfully achieved. In A.2, we clearly state the problem under consideration and in A.3 we present two algorithms which are developed in [47]. Experimental results and concluding remarks are then given in A.4 and A.5, respectively.

A.2 Problem Formulation

As stated earlier, the aim is to design a robust fault-tolerant scheme for $n$ AVs such that they acquire a desired formation in spite of faults or bad weather. Consider a formation of $n$ AVs which track the position $p_T(t)$ of a moving target in an uncertain area $X$. We assume that $p_T(t)$ is smooth, i.e. differentiable, with respect to time $t$. Each AV’s sensor senses the target position $p_T(t)$, and its corresponding reading is denoted by $p_{Ti}(t), (i = 1, 2, \ldots, n)$. When the position sensor is not faulty, its reading follows a Gaussian distribution with a standard deviation of $\sigma$ with respect to the nominal value $p_T(t)$. We assume that sensor readings are uncorrelated with each other over time. The $i$th AV shares $p_{Ti}(t)$ with other AVs within the sensor range (a sphere of radius $\rho_i$ centred at the $i$th AV) in order to keep a fixed formation $F$ even in the presence of sensor faults. The operational area $X$ may contain a faulty (such as bad-weather) zone where some AVs (sensors) may not correctly measure the target position. Assuming that the communication network of AVs is connected in a standard graph theoretical sense and there are less than $q(< n/2)$ AVs incorrectly sensing the target position all the time, our objective is to design a robust fault-tolerant target tracking scheme such that all the AVs always keep the initial formation $F$ within a prescribed fixed tolerance throughout the mission. We assume that once the processed (estimated) target location $p_{Ti}(t + \Delta t)$ is known to the $i$th AV, the $i$th AV is capable of changing its current location $p_i(t)$ to $p_i(t + \Delta t)$ with little effort such that
\[ p_i(t + \Delta t) = p_{Ti}(t + \Delta t) + p_i(0) - p_{Ti}(0), \text{ i.e. it keeps the initial relative position with respect to the target location.} \]

### A.3 Solution Approach

The fault-tolerant target tracking scheme has two embedded algorithms: (i) a semi-decentralized dynamic data fusion algorithm and (ii) a fault detection algorithm.

#### A.3.1 Algorithm I – Semi-decentralized dynamic data fusion

The semi-decentralized data fusion algorithm adopts the following update rule to estimate the target position by \( i \)th AV

\[
\begin{bmatrix}
x_{Ti}(k+1) \\
y_{Ti}(k+1) \\
z_{Ti}(k+1)
\end{bmatrix} = 
\begin{bmatrix}
(1 - \gamma \Delta t)x_{Ti}(k) + \sum_{j \in \eta_i} b_{ij}(k)p_{xj}(k) \\
(1 - \gamma \Delta t)y_{Ti}(k) + \sum_{j \in \eta_i} b_{ij}(k)p_{yj}(k) \\
(1 - \gamma \Delta t)z_{Ti}(k) + \sum_{j \in \eta_i} b_{ij}(k)p_{zj}(k)
\end{bmatrix}
\]

where

\[
b_{ij}(k) = \frac{\gamma e^{-\alpha|\bar{p}_i(k) - p_j(k)|}}{\sum_{j \in \eta_i} e^{-\alpha|\bar{p}_i(k) - p_j(k)|}}
\]

Here \( x_{Ti}(k+1), y_{Ti}(k+1), z_{Ti}(k+1) \) are the x, y and z position estimated position of the target by the \( i \)th AV in global coordinate system, \( p_{xj}(k), p_{yj}(k), p_{zj}(k) \) are the measured target position in x, y and z direction respectively, \( \eta_i \) is the set of the \( i \)th AV’s (non-faulty) neighbours including itself and \( b_{ij} \) is the information index which determines the trueness of the measured quantity in each direction. Moreover, \( \alpha > 0 \) and \( 0 < \gamma < 1 \) are constant parameters and \( \bar{p}_i(k) \) is the median of the positions of the \( i \)th AVs neighbours and itself.

#### A.3.2 Algorithm II – Fault detection and isolation

The fault-tolerant scheme work in two steps: (i) it finds global median from non-faulty AVs (AVs which have measurement within prescribed tolerance); (ii) propagating of find global median to entire network if they have discrepancy with neighbours (the extended neighbor provide global median position to save communication time and to avoid centralized scheme) to detect
the faulty AV. Once the faulty AV is identified, its measurement are discarded for the rest of mission, because it is obvious faulty unit can be repaired online.

The first step of the fault detection algorithm makes use of $\eta_i$ neighbours. Each AV gathers target information from its immediate neighbours and compares the information with its own. If the difference in target position between the three AVs exceeds a fixed tolerance level $\sigma$, the value of global median is sought from extended neighbors. This process is repeated for each AV in the formation until a global median can be reached. In other words, each AV gathers enough information from (up to 2q) other AVs over a time period until it can determine a global median of $p_i, i = 1, 2, \ldots, n$. Since there are at most q faulty sensors, the almost global median can be determined if the gathered information contains at least $(q + 1)$ similar sensor readings, i.e. $|p_i - p_j| < 2\sigma$.

The second step of the fault detection algorithm is the propagation of the determined almost global median to every AV for detecting and isolating faulty sensors. Once an AV notices that one of its immediate neighbours knows the global median, it compares the global median with its target estimation at the time when the fault detection algorithm was initially applied. If the difference is beyond $2\sigma$, the AV’s sensor is diagnosed as faulty. Since the proposed data fusion algorithm is now equipped with the fault detection algorithm, it can be assured that faulty readings cannot enter the data fusion equation A.1.

### A.4 Implementation of the Algorithm

In real life, it is expected that external disturbances, including bad weather zones and internal electronic malfunctions can lead to sensor failure. Moreover some unexpected situation may interrupt sensor vision. It is imperative to design a robust fault tolerant algorithm which keeps the AVs in a formation even in the presence of sensor failure and other cases where target information cannot be obtained accurately from onboard sensors. To check the concept, an experiment was designed and conducted with a set of robots in the ConSERT laboratory of SEIC. A set of four follower robots, named as Sylvester, Garfield, Felix and Tom, were used in a chosen configuration. As mentioned before, the aim of the mission was to maintain constant relative distance between the follower robots and the target robot while maintaining a formation of the follower robots in the presence of sensor failures. It has been assumed that a lower level
(inner loop) controller tracks accurately the commanded position generated by a data fusion algorithm for each follower robot.

A.4.1 Robot information

The type of robot used in the experiment is the KOALA, which is a mid-sized robot designed for real world applications. It is quite powerful and capable of carrying large accessories (such as sophisticated battery management systems), and rides on 6 wheels for indoor terrain operations. The Koala and its modules are easy to operate through connections to very standard and well known tools.

Each wheel is driven by a DC motor, coupled through an $84:1$ reduction gear. An incremental encoder is placed on the motor axis and gives 100 pulses per revolution of the motor. This gives 8400 pulses per revolution of the wheel, which corresponds to $32.5$ pulses per millimeter of forward displacement of the robot. The motor power supply can be adjusted by the main processor by switching it ON and OFF at a given frequency and for a given time.

The sensor used in our experiments is a RANGE-FINDER type Laser Scanner URG-04LX. The light source of the sensor is an infrared sensor laser with wavelength $785$ nm. The scanning area is $270$ degrees with $0.36$ degree/pitch and it can detect the distance and direction to the object. The maximum detectable distance is $4$ m. Each robot is powered by a $5V$ DC supply and for connectivity it uses RS-232 USB. Laser sensors whose range is limited to $180$ degrees field of view are mounted on the robots. All the parameters of each robot are set by means of a small laptop mounted on each vehicle. Another way to configure the robot’s initial status is by means of a central hub computer. In this case, it is still not a centralized scheme, because each robot transmits information to other robots via the central computer. Communication between the central computer and a Koala robot is made by sending and receiving ASCII messages. Every interaction is composed of

- A command, sent by the central computer to the KOALA robot and followed by a carriage return or a line feed.
- When required, a response, is sent by the KOALA to the central computer.
In all communications the central computer plays the role of master and each Koala plays the role of a slave. All communications are initiated by the master. The different parameters that can be set before the start of a mission (tracking a target) are:

- Bearing angle of the robot
- Laser angle
- Respective initial coordinates
- Laser Range (we have set this to 3m)

A.4.2 Experimental setup

The aim here is to design a robust fault tolerant scheme for n robots such that they all maintain a specific formation in spite of sensor failures. The goal is to test the algorithm presented in Section A.3. In the first experiment, each robot first tracks the target and maintains its relative position with respect to the target in a fault-free case. When a fault is introduced in the environment, i.e. a sensor communication is blocked, then the consensus algorithm comes into play. The parameters used in the algorithm are:

- \( n = 4 \) is the number of robots
- \( p_T(k) = [x_T(k), y_T(k)]^T \) is the position of the target at time step ‘k’
- \( X \) is the uncertain target area
- \( \sigma \) is a positive constant
- \( \rho_i \) is the \( i \)th sensor radius used to estimate the target position
- \( x_{Ti}, y_{Ti} \) are estimated target positions by the \( i \)th robot sensor
- \( q \) is the number of faulty sensors \(< n/2\)
- \( x_i(k), y_i(k) \) are the position coordinate of the \( i \)th robot
- \( \mathcal{N}_i^\kappa \) is the set of \( i \)th robot’s non-faulty neighbours
Four robots are placed in the configuration as shown in Fig. A.1 and they are tasked to track the position $p_T(k)$ of a mobile robot (target) in a designated area. Each robot senses the target position $p_i(k)$. This $p_i(k)$ is shared with other robots within the sensor range to estimate the target position for keeping a fixed formation $F$. The estimated target position by the $i$th robot is denoted by $p_T = [x_T, y_T]^T$ (since it is in two dimensions, we have dropped the $z$ coordinate). For the validation of the algorithm, the number of faulty sensors $q < 2$ i.e. less than half the sensors are allowed to be faulty. Our objective is to design fault tolerant information sharing rules such that all robots always keep the initial formation $F$ (within a prescribed fixed tolerance). Once the estimated target position $p_T$ is known to the $i$th robot, the $i$th robot changes its current position $p_i(k)$ to $p_i(k+1)$, so as to keep its initial relative position with respect to the target location, i.e.

$$p_i(k+1) = p_T_i + p_i(0) - p_T(0)$$

(A.2)

The underlying idea is that each robot tracks the target by employing the median consensus algorithm and shares this information with the other robots. The algorithm detects faulty sensors if they exist. This is done in two steps: (i) find the global median within the fixed tolerance; and (ii) propagate the median value to every robot to diagnose sensor health. Once all the robots are diagnosed as faulty or non-faulty via the second algorithm, $\mathcal{N}_i^{E}$ is updated accordingly.
In finding the global medians the neighbourhood is extended i.e. not only the immediate neighbours are considered but the next-to-the-immediate neighbours are also taken into account, up to a certain number of robots. For example in Figure A.1, if the sensor reading of Robot 2 is corrupted, Robot 2 needs to gather information not only from Robots 1 and 3 but also from Robot 4. This is because it needs at least three values (from more than half of all the robots) to be similar to each other to identify the current global median within a fixed tolerance. Since there is no direct communication between Robot 2 and Robot 4, it will take two time units for Robot 2 to operate this. The second step of Algorithm II is the propagation of the global median to every robot in order to identify faulty sensor(s). Once a robot notices that one of its immediate neighbours knows the global median, it can compare the global median (of the target information) with the information it has at the time. If the difference is beyond the prescribed tolerance level of $2\sigma$, the robot sensor is diagnosed as faulty.

Three experimental tests were conducted. Initially, the robots’ manoeuvring and movement were examined and calibrated, and communication and control through the central computer were analyzed. In the first test, a collective manoeuvring test, the robots tracked the target without any faults in the environment. Since every robot tracks the target by itself and also shares the information of the current location of the target with the other robots (to find the global median), to confirm this another test was carried out. One of the robots had its tracker OFF (and locator left ON); so it couldn’t track the target and was left behind. In this way, the target went beyond the sensor range of the “faulty” robot. Then the tracker was switched ON for the robot in question. This robot knows its initial position and receives information of the current position of the target from the other robots. Based on this information, the robot can find the required direction and adjust itself until it reaches the (prescribed) relative position with respect to the target and its position in the formation (with fellow follower robots). In the third test, all the follower robots track the target. A fault was then introduced in real-time in one of the robots (In this exercise, a plastic cup was placed over the sensor of each of the robots in turn). The cup caused the particular robot’s sensor reading to drop to some arbitrary value. Although the ‘Fault detection algorithm’ was in operation throughout all the experiments, it was only at this point that the fault detection and isolation algorithm started to work. The faulty robot was diagnosed, and non-faulty measurements are being made available to it so that it can track the target.
A.4.3 Experimental results

As mentioned earlier, sensors mounted on each robot give radial distance from the target and heading angle from vehicle body frames. Once the relative distances in X and Y directions are known, then after transforming polar measurements into Cartesian measurements, the actual position of the target is calculated using the follower robot position. Each follower robot’s position is determined by GPS; in our case, we have used a wireless network.

The target position for each follower robot is estimated by a data fusion algorithm. Before fusion the algorithm checks for faults. If faults are spotted the relevant measurements are not considered in estimating the target position. Moreover, the position of the target is provided to the faulty vehicle by a non-faulty neighbour of the robot using the consensus algorithm as described in Section A.3.

Figures A.2 and A.3 shows the relative positions X and Y of the target from each follower robot measured by the mounted sensors. Faults in sensors are introduced by covering a sensor with a plastic cup so that it cannot detect the actual target. During the interval of covering, “malicious” target positions are measured by the covered sensor. The spikes in Figures A.2 and A.3 represent this scenario. Trajectories of each robot and of the target are plotted in Figure A.4. It can be observed from the figure that in spite of sensor failures the follower robots maintain the formation while keeping constant relative distance with the target. Experiments are recorded and the video of the experiments is attached with the thesis.

![Figure A.2: Relative X position of each robot](image_url)
Figure A.3: Relative Y position of each robot

Figure A.4: Trajectories of the robots in the formation

A.5 Conclusions and Future Scope

The fault-tolerant scheme proposed in [47] has been tested on a set robots in the ConSERT Laboratory. The scheme uses a two-step procedure to identify faulty sensors and then to use the information to take corrective action. The concept of extended neighbours is employed to obtain a global median of target positions and this value is used by the sensors in the network to identify faulty ones. Another part of the scheme uses the information on faulty sensors to estimate the target position from a data fusion algorithm. Sensor faults are simulated in the ConSERT robot formation by means of vision obstruction. Results from the experiments demonstrate the convergence and filtering properties of the algorithm.
Bibliography


