Moving Obstacle Avoidance for Unmanned Aerial Vehicles

by

Yucong Lin

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2015 by the
Graduate Supervisory Committee:

Srikanth Saripalli, Chair
Paul Scowen
Georgios Fainekos
Jekanthan Thangavelautham
Cody Youngbull

ARIZONA STATE UNIVERSITY

December 2015

ABSTRACT

There has been a vast increase in applications of Unmanned Aerial Vehicles (UAVs) in civilian domains. To operate in the civilian airspace, a UAV must be able to sense and avoid both static and moving obstacles for flight safety. While indoor and low-altitude environments are mainly occupied by static obstacles, risks in space of higher altitude primarily come from moving obstacles such as other aircraft or flying vehicles in the airspace. Therefore, the ability to avoid moving obstacles becomes a necessity for Unmanned Aerial Vehicles.

Towards enabling a UAV to autonomously sense and avoid moving obstacles, this thesis makes the following contributions. Initially, an image-based reactive motion planner is developed for a quadrotor to avoid a fast approaching obstacle. Furthermore, A Dubin's curve based geometry method is developed as a global path planner for a fixed-wing UAV to avoid collisions with aircraft. The image-based method is unable to produce an optimal path and the geometry method uses a simplified UAV model. To compensate these two disadvantages, a series of algorithms built upon the *Closed-Loop Rapid Exploratory Random Tree* are developed as global path planners to generate collision avoidance paths in real time. The algorithms are validated in Software-In-the-Loop (SITL) and Hardware-In-the-Loop (HIL) simulations using a fixed-wing UAV model and in real flight experiments using quadrotors. It is observed that the algorithm enables a UAV to avoid moving obstacles approaching to it with different directions and speeds.

ACKNOWLEDGEMENTS

There are a number of people without whom this thesis might not have been written and to whom I am greatly indebted.

First I would like to thank my advisor Prof. Srikanth Saripalli for giving me an opportunity to work on robotics and patiently guiding me for the last five years. His patient and in-detail supervision successfully help me become a newbie roboticist from an astrophysics betrayal. I also would like to thank my thesis committee Paul Scowen, Georgios Fainekos, Jekan Thanga, and Cody Youngbull. Their insightful suggestions made this thesis more complete.

I am extremely grateful to my current fellow labmates: Aravindhan Krishnan, Sai Vemprala, Ben Stinnet. They constantly provide technical support and proof-reading support. I also thank former lab members or alumni: Andres Mora, Shatadal Mishra, Ganesh Kumar, Anandrao Biradar, Patrick McGarey, Brance Hudzeitz, and Collin Ho. They helped me a lot when they were in the lab. I also apprieciate visiting students from Universidad Politecnica de Madrid in spain including Changhong Fu, Jesus Pestana, and Adrian Carrio. They helped me learn a lot of knowledge that I cannot learn within the lab.

My internship in SRI International is an unforgetable experience in my Ph.D. I learned a great deal of knowledge and skills from Xun S. Zhou, Aveek Das, Philip Miller, Mikhail Sizintsev, Han-Pang Chiu, Supun Samarasekera, and Rakesh (Teddy) Kumar. I also thank other interns including Fan Yang, Willian Hua, and Tejas Mathai. We spent a good time together.

I will never forget my seniors and peers in School of Earth and Space Exploration of ASU. It was great to be in the same school with them. They are Chunpeng Zhao, Zhengya Zheng, Patty Lin, Jun Wu, Mingming li, Jingping Hu, Ruirui Han, Shule Yu, Guang Zhai, Tiantian Xiang, Qian Zhang. Communications with my friends since University of Science and Techonology of China are important part of my Ph.D years. These Friends are Ning Jiang, Hua Wen, Yiqiu Ma, and Zhou Lin. My friends outside school gave me a lot of support in life during these five years. They are Yian Song, Zhaofeng Gan, Chufeng Li, Qiushi Mou, and Le Lu. I also thank my roommates Qi Dong and Da Guo during these few years. Da Guo taught me a lot in driving.

Closer to home, I would like to thank my parents for their love, support and encouragement. Last but not least, I thank my wife, Shuo Huang, who appeared in my life in my most desperate period in my Ph.D years. She walked me through the time I was in most need of emotional and spiritual support. Without her, this thesis will be completed with much lower quanlity.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

An unmanned aerial vehicle (UAV) is defined as an aircraft with no onboard human pilot. It has been almost a century since the first use of UAVs in history–the Hewitt-Sperry Automatic Airplane in Word War I. UAVs have evolved from a radio remote controlled airplanes to fully autonomous platforms. They have also expanded their applications from aerial targets and reconnaissance for military purpose to many different civilian areas. To list a few, they are used in:

- **Aerial photography**: Using an UAV significantly reduced the cost of aerial photography. Besides shooting images of a large building, UAVs are recently more and more used in taking pictures of human or individual behavior due to the reduced size, increased easiness of operation, and lower costs. For example, *DJI Phantom* is almost ubiquitous for aerial photography in different events all around the world.

- **Infrastructure monitoring**: UAVs provide a new and unique perspective to inspect infrastructures that is difficult to access for human-being. Using UAV in infrastructure inspection is much less risky than traditional inspection in which human needs to closely access the infrastructures. UAVs have been used in bridge (Metni and Hamel (2007)) and powerline (Sampedro *et al.* (2014)) inspections.

- **Search and Rescue**: Search and rescue missions are time-consuming, expensive, and often dangerous for the people involved. The use of well-equipped drones is increasing for search and rescue and could soon become a standard way to cover large areas of inaccessible terrain, even at night. For example, a Dragan Flyer X4-ES drone with heat-sensing equipment, launched by the Canadian Mount Police, found a disoriented driver from a late-night rollover in a remote location of Saskatchewan, Canada, in May 2013.

- **Wildlife protection**: UAVs are already widely used in wildlife protection. In Colorado, the U.S. Geological Survey has mounted a thermal imaging camera on an AeroVironment Raven to count sandhill cranes. Drones have been used in Indonesia to monitor wildlife in forests. In Namibia, UAVs have been purchased to monitor game parks and to track poachers.

- **Agriculture**: UAVs can be used in checking and spraying crops, finding lost cattle. Yamaha RMAX has been flown in Japan for 20 years to treat the farmlands on steep hillsides.
- **Cargo transport**: Logistics related companies such as Amazon, DHL, UPS, and FedEx are looking into transportation or delivery using UAVs. Amazon has demonstrated its concept of *Amazon Prime Air*, a light UAV platform for short distance delivery. DHL has used a quadrotor to carry medicine from the harbor town of Norddeich, Germany, to the small island of Juist.

Given the wide applications mentioned above, there is a strong wish to integrate UAVs into the National Airspace. In the recently released Unmanned Aerial System Integration Roadmap (FAA (2013)), the Federal Aviation Association (FAA) raised two technological challenges for the integration: 1.Control and Communications (C2) system performance requirements, 2. "Sense and Avoid" (SAA) capability. This thesis aims to provide a solution to the "Sense and Avoid" challenge. So, what are the challenges for the sense-and-avoid problem for UAVs?

1.1 Challenges

- **Moving obstacle detection**: A sensor used in moving obstacle detection should be able to continuously provide the 3D position of a moving obstacle. It is difficult to find an appropriate sensor for UAVs to detect moving obstacles onboard. Airborne Radar can detect the relative location of other aircraft, and has been in military use since World War II. While larger civil aircraft carry weather radar, sensitive anti-collision radar is rare in non-military aircraft. For small UAVs, radar is expensive, power-hungry, and adds more pay-load. Commercial RGB-D sensors such as *Kinect* and *Xtion* are able to provide depth information but their sensing range is insufficient for long range obstacle detection. Heavy lidars such as Velodyne are able to detect the distance to the obstacle, but to carry them in an UAV will significantly add the the payload. A monocular camera has been used to detect and tracking moving obstacles, but it relies heavily on the image processing and computer vision algorithms in detection and tracking. To develop a reliable algorithm for the purpose is still a research problem till today. As well as active sensing, cooperative communication has also been proposed for collision avoidance. Examples of the technologies are the *Traffic alert and Collision Avoidance System* (TCAS) and *Automatic Dependent Surveillance - Broadcast* (ADS-B). In fact, ADS-B is an element of the US Next Generation Air Transportation System (NextGen) (FAA (2010)) and is a requirement for all commercial aircraft by 2020. ADS-B receiver is light to carry. Therefore, ADS-B can be an ideal technique to detect other aircraft in the national airspace.

However, the downside is that vehicles except for aircraft such as airship may not be equipped with an ADS-B transmitter.

- **Motion prediction for moving obstacles**: In moving obstacle avoidance, the motion of moving obstacles needs to be predicted to estimate the possibility of its collision with the UAV. In Shim and Sastry (2007); Lin and Saripalli (2014), obstacles motion is predicted by linear extrapolation based on the obstacle's current position and velocity. It is highly approximated prediction because the obstacle may change its moving direction and velocity at any time. Gaussian process and its extensions are used in Fulgenzi *et al.* (2008a); Luders *et al.* (2011); Aoude *et al.* (2013) to predict the motion of dynamic obstacles with uncertain motion patterns. However, learning is needed to create the Gaussian model of the moving obstacles. Such learning is feasible in a known environment with small area, but is difficult in the whole national airspace due to the large variety of many different aircraft.

- **Generation of avoidance paths**: Moving obstacle avoidance can be formulated as a path planning problem. A solution to the problem should satisfy the following requirements: 1) a path satisfying obstacle avoidance constraints should be fast generated. 2) the generated path must be dynamically feasible for the UAV to execute. 3) the path must be optimal in terms of some target function such as path length. Geometry methods (Yang *et al.* (2013); Mejias and Campoy (2011); Melega *et al.* (2011)) are fast but optimality is not guaranteed. In Patel *et al.* (2009); Lai *et al.* (2011); Liu and Hwang (2014); Richards and How (2002), aircraft collision avoidance is formulated as pure optimization problems. Such solution does make sure optimality but they are not fast and only uses simplified UAV and aircraft models. Therefore, the generated path may not be dynamically feasible for real UAVs. Many kinds of path planning methods are developed for static obstacle avoidance (Scherer *et al.* (2008); Kobilarov (2012); Hrabar (2011)), but it is unclear if the methods are applicable for moving obstacle avoidance.

1.2 Contributions

While there are continuous advancement in research on Sense and Avoid for UAVs, we put our focus on the development of a path planning method that satisfy all the three requirements in the section 1.1. We also develop a strategy that helps the UAV to compensate the inaccurate motion prediction of the moving obstacles.

- **Development of real time path planner for UAV moving obstacle avoidance:** This planner is able to generate in real time a path that satisfies obstacle avoidance constraints, accommodate the UAV's dynamics, and is optimal in path length.

- **Design of a execution loop for the UAV to compensate the inaccurate prediction of obstacles' motion:** In the execution loop, collision prediction is always being performed, even when the UAV is following the generated path for avoidance. If the path is predicted to induce collision when the obstacle changes its motion, a new path will be generated.

- **Experimental Validation of path planners:** To validate the path planning algorithm and the execution loop, the following experiments are performed:

  – A quadrotor flying in partially open area (a parking structure) to avoid one or multiple moving obstacles. The moving obstacles includes actual and virtual flying quadrotors. Cooperative communication was used in the experiments.

  – A fixed wing UAV to avoid other aircraft in Software-In-the-Loop Simulation. More than one obstacle aircraft were involved. The UAV encountered them either simultaneously or in sequence. The obstacle aircraft approached the UAV in different directions. Logged ADS-B data of commercial aircraft are used in the experiments.

  – A fixed wing UAV to avoid other aircraft in Hardware-In-the-Loop Simulation. The system setup is closer to actual flight experiments.

  – A quadrotor flying in outdoor environment. Another quadrotor is used as the actual moving obstacle. GPS is used as the positioning system.

1.3 Outline of the Thesis

The thesis is organized as follows:

- Chapter 2 describes prior work in moving obstacle detection by UAVs, motion prediction of moving obstacles, and path planning for moving obstacle avoidance.

- Chapter 3 describes the system setup for all experiments used in the following chapters.

- Chapter 4 describes an image based reactive motion planner for moving obstacle avoidance. This is our first trial to solve the problem.

- In chapter 5, we explore the possibility to achieve moving obstacle avoidance using a simple geometry based avoidance algorithm using Dubins curve.
- The application of *Closed-Loop Rapidly Exploratory Random Tree* is introduced and extended in chapter 6 as the path planner for the UAV to avoid moving obstacles. The loop of execution is also presented in this chapter.
- Chapter 7 extends the method in chapter 6. The path planner is improved to create more candidate paths in the same amount of time and to deal with motion uncertainty of moving obstacles.
- The four types of experiment validations are described in chapter 8. We detail experiment setups and results.
- Chapter 9 summarizes the contributions and suggest the future work.

Chapter 2

RELATED WORK

Unmanned Aerial Vehicles have been an active area of research for several years. This chapter reviews existing sense and avoid techniques for UAVs. The literature review aims to evaluate their applicability to moving obstacle avoidance for UAVs. Sensing-detection of obstacles will first be discussed and avoiding-motion planning will follow.

2.1 Obstacle Detection

Obstacle detection methods can be categorized based on the sensors that are used.

2.1.1 Cameras

As a passive sensor, a camera needs to be combined with image processing algorithms to detect obstacles. The morphological filter is used in Wainwright and Ford (2012); Gandhi *et al.* (2003); Carnie *et al.* (2006) to detect aircraft under different backgrounds. However, a big number of false positives are generated. Dey *et al.* (2010) utilizes shape descriptor and SVM-based classifier to achieve aircraft detection of low false positives. The classifier should be trained offline with hand-labeled sample image data. McCandless (1999) proposes moving object detection with thresholding on optic flow of feature points of each image frame of a video. Multiple-Instance learning approach, Multiple-Classifier voting mechanism, and Multiple-Resolution representation are used in Fu *et al.* (2014) to achieve precision aircraft tracking but offline training is required for initial detection. The distance to the aircraft can be estimated using the aircraft's size in the image and the pinhole camera model. An initial guess of the distance to the obstacle and Extended Kalman Filter is used in Watanabe *et al.* (2007) to estimate the distance to the obstacle but it is only applicable to static obstacles.

Stereo cameras acquire depth information using triangulation and are used widely in mapping and navigation for UAVs (Shen *et al.* (2012); Hrabar (2008); Heng *et al.* (2011); Andert *et al.* (2011)). These works do not require long range of detection. However, the depth accuracy from triangulation drops significantly for longer range. Therefore, image processing is needed for stereo cameras if the obstacle is far away.

2.1.2 Optic Flow

Optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene. Range detection using optic flow captures the idea that when you move closer to an object its size in your eye expands. Such fact is quantitatively described in Green and Oh (2008). Optics flow sensors are used in Green *et al.* (2003, 2004) to perform autonomous take off, obstacle avoidance, and landing of a fixed-wing indoor aircraft. Optic flow and stereo vision are combined in Hrabar and Sukhatme (2009) for obstacle avoidance through urban canyons. The approach is tested on an autonomous helicopter and tractor. To apply optic flow to estimation of the distance to a moving obstacle, we must be able to accurately measure the velocity of the obstacle.

2.1.3 Lidar

LIDAR-Light Detection And Ranging detects objects by scanning a laser beam in the environment and measuring distance through time of flight or interference. Off the shelf LIDAR systems primarily use a single-line scan that produces 2-dimensional information. Such LIDAR is used in indoor mapping and navigation of small UAVs (Shen *et al.* (2012); Bachrach *et al.* (2009)) and UGVs Thrun *et al.* (2002). In all of them, an occupancy grid map is built based on the range data. It is also used in Merz and Kendoul (2011) for obstacle avoidance of a slowly flying UAV and in Hrabar (2012) combined with stereo camera to construct a 3D occupancy map for reactive obstacle avoidance. 2D lidar is light and suitable as the payload of small UAVs. But it only provides 2D laser scanning that is unable to track a moving obstacle whose motion is not restricted to a 2D plane. A pannable LIDAR in Scherer *et al.* (2008); Geyer and Johnson (2006); Tsenkov *et al.* (2008) is able to map the 3D environment but the weight and size of such machinery are formidable to small UAVs. HELLAS-A [1] is an awareness system in real service that uses in conjunction with other systems to give early warning of obstacles in the flight path. But it is manufactured for real helicopters to carry.

2.1.4 Other Sensors

Other sensors used in obstacle avoidance includes ultrasonic sensors (Borenstein and Koren (1989, 1991)) and RGB- D sensor (Henry *et al.* (2010); Biswas and Veloso (2012)). These two types of sensors

[1] http://www.fairchildcontrols.com/products/electronics-and-avionics/hellas-awareness/

have short detection range therefore cannot be applied to detection of moving objects approaching from far away. Radar is a reliable sensor to detect and track other aircraft. Larger UAVs like the Navy Tier III MQ-8B Fire Scout do have high performance Radar systems that could potentially be used for collision avoidance. For avoidance of collision in air traffic, radar technology provides a legitimate solution. However, it is relatively expensive, heavy, power hungry, and not accessible for civilian domains. This limits its application in UAVs with small payloads and short power endurance.

Some cooperative sensing system for aircraft collision avoidance are presented in Contarino and Scire Consultants (2009). Examples of them are The Traffic alert and Collision Avoidance System (TCAS) and Automatic Dependent Surveillance - Broadcast (ADS-B). TCAS monitors the airspace around an aircraft for other aircraft equipped with a corresponding active transponder, independent of air traffic control, and warns pilots of the presence of other transponder-equipped aircraft which may present a threat of mid-air collision. ADS-B relies on the Global Positioning System (GPS) and onboard systems to determine the aircraft's position in space. It is automatic because it periodically transmits information without pilot, operator or external interrogation input. It is dependent because its position is obtained from GPS. It is surveillance because it is a method of determining position of aircraft, vehicles or other assets. Finally, broadcast is the ability of anyone with appropriate receiving equipment to utilize the transmitted information. These sensing systems are designed for large aircraft for commercial flight and they are the potential sensor for sensing and avoiding other aircraft in air traffic.

2.2 Obstacle Avoidance

Existing works in obstacle avoidance can be categorized into global path planning methods and reactive avoidance methods.

2.2.1 Global Path Planning methods

Robotic path planning seeks to find a solution to the problem "Go from the start to goal while respecting all of the robot's constraints."

Graph Search Method

One way of path planning is to discrete the obstacle-free portion of the workspace into cells and model the cells as a graph or roadmap. Utilizing this graph, the problem of robot path planning becomes a graph search problem. A^* is used online to search a path to the goal from a roadmap constructed by

OBB-Tree (Pettersson and Doherty (2006)). A dynamic version of A^* called D^* is proposed in Stentz (1994) for optimal and efficient re-planning in partially known environments. In stead of re-calculating the optimal path of the entire map when new obstacle is detected, D^* only checks a reduced set of cell and incrementally update the robot's optimal pose. Focused D^* (Stentz (1994)) improved on D^* with the addition of a heuristic focusing function so that the total time for re-planning is reduced. D^* Lite (Koenig and Likhachev (2002)), built on Lifelong Planning A^* (LPA^*) (Koenig and Likhachev (2001)), implements the same navigation strategy as Focused D^* but is algorithmically different. The authors showed that D^* Lite can be rigorously analyzed and experimentally appears to be even slightly more efficient than D^* . These incremental planners substantially speed up the planning cycles by making use of the results of the previous plans to generate a new plan. However, finding an optimal plan within the available time is difficult. Anytime algorithms (Zilberstein and Russell (1995)), in contrast, try to find the best plan within the given available time. Likhachev *et al.* (2008) presents an A^*-based anytime search algorithm, which is able to incrementally search the most optimal plan within available time. Graph-based method has its disadvantage in: 1) It is not clear how to move the robot from one cell to an adjacent cell when it has complex non-linear dynamics. 2) Partitioning a high-dimensional state space of a robot into cells is difficult and impractical.

Sampling-based Methods

Sampling-based planning algorithms are proposed to deal with high dimensional configuration space.

Probabilistic Roadmap (PRM) (Kavraki *et al.* (1996)) divides planning into two phases: the learning phase, during which a roadmap in free configuration space is built; and the query phase, during which a path is constructed with the precomputed roadmap. Pettersson and Doherty (2006) applies PRM in path planning for an unmanned helicopter. In the paper, PRM is used to construct the roadmap offline with OBB-tree (Gottschalk *et al.* (1996)) as the collision checker. Sekhavat *et al.* (1998) deals with non-holonomic constraints in the probabilistic roadmap. The approach can be applied in path planning of a robot car. Missiuro and Roy (2006) extends PRM to compute motion plans that are robust to environment uncertainty. They modify randomized sampling methods in order to minimize the number of samples required to express good plans and then evaluate PRM actions efficiently in the context of uncertainty and generate motion plans with minimal expected cost. Time is added as an extra workspace dimension in Hsu *et al.* (2002); Van Den Berg *et al.* (2006) to the probabilistic roadmap to deal with moving obstacles. The computation load is also increased.

Rapidly-exploring Random Trees (RRT) (LaValle and Kuffner (2001)) handles systems with differential constraints effectively. RRT selects a random state x_{rand} and then find the nearest node to the x_{rand} in terms of distance metric on the state space. Next select the control input u_{new} that minimizes the distance from x_{near} to x_{rand} and finally check for collision. If there is no collision, x_{new} , u_{new} are added as a new vertex in the tree. Redding *et al.* (2007) applies dual RRT to generate waypoints to the goal. They can be pruned using one or more iterations of Dijkstra's algorithm, resulting in shorter paths that approach the optimal. Kuwata *et al.* (2009) extended the RRT to Closed-loop RRT (CL-RRT) algorithm by making use of a low-level controller and planning over the closed-loop dynamics. It was used in MIT's vehicle as the entry to DARPA Urban Challenge in 2006. To perform navigation in dynamic environment, Fulgenzi *et al.* (2008b) extends RRTs by taking into account the likelihood of the obstacles trajectory and the probability of collision. Moving obstacles in the work are supposed to move on typical patterns which are pre-learned and represented by Gaussian processes. Karaman *et al.* (2011) proposes RRT* as a modification of RRT to achieve a path that surely converges to an optimal solution by using a new nearest node searching method. Examples of other tree-based algorithms are *Expansive-Spaces Tree* (EST) (Hsu *et al.* (1997)) and *Sampling- Based Roadmap of Trees* (Bekris *et al.* (2003)).

Other Planning Methods

Pivtoraiko *et al.* (2009) proposes a state lattice motion planning method. It is based on deterministic search in a specially discretized state space. A set of elementary motions are computed for discrete state connection. In Weiss *et al.* (2006), the kinematic of the UAV is reduced to a set of feasible trim trajectories and maneuvers. The local operating Iterative Step Method (ISM) sequentially determines the next best trim trajectory, which minimizes a local cost-function. In Kobilarov (2012), path planning is performed in trajectory space. A probability distribution over the set of feasible path is constructed and the search for an optimal trajectory is performed through importance sampling. In Templeton *et al.* (2007), *Model predictive control* (MPC) is used to generates a safe vehicle path from an online map built by a laser scanner. A cost function that penalizes the proximity to the nearest obstacle is adopted.

2.2.2 Reactive Methods

In contrast to planning algorithms, reactive methods (local approach) use a simple formula to react to obstacles as they appear, which have low demanding of computing resources, making them very suitable to real-time robot navigation. In these methods, only the nearest portion of the environment is used and

the world model is updated according to the current sensor observation. Only the next input control instead of a whole path to the goal are generated.

Velocity-based Methods

The dynamic window approach (Fox *et al.* (1997)) deals with the constraints imposed by limited velocities and accelerations of the robot. It consists of generating a valid search space and selecting an optimal solution in the search space. Simmons (1996) proposed Curvature-Velocity Method (CVM), where local obstacle avoidance is treated as constrained optimization problem in the velocity space of the robot. Constraints that stem from physical limitations (velocities and accelerations) and the environment (the config- uration of obstacles) are placed on the translational and rotational velocities of the robot. Ko and Simmons (1998) combines CVM with the Lane Method, which divides the environment into lanes and then chooses the best lane to follow to optimize travel along a de- sired heading. These methods do not take into account the dynamic information of the environment, considering all the obstacles as static ones.

On the other side, the Velocity Obstacles approach (Fiorini and Shiller (1998)), the Inevitable Collision States concept (Fraichard and Asama (2003)) use a deterministic knowledge about the velocity of the obstacles to compute collision-free controls. A velocity obstacle (VO) is the set of a robot's velocities that will result in a collision with other moving objects at some moment if both of them maintain current velocities. A velocity inside VO will result in collision with the objects while one outside VO will avoid a collision. The extension to nonlinear VO is proposed in Shiller *et al.* (2001). Inevitable collision states for a given robotic system are states for which, no matter what the future trajectory followed by the system is, a collision eventually occurs with an obstacle of the environment. Owen and Montano (2006) maps the dynamic environment into a velocity space, using the concept of estimated arriving time to compute the times to potential collision and potential escape. Damas and Santos-Victor (2009) presents a computationally fast algorithm that allows a robot to avoid col lision with multiple moving obstacles. Instead of performing an exhaustive search on the robot velocity space, the proposed method computes the Forbidden Velocity Map, a union of polygonal zones corresponding to the non admissible velocities.

Potential Field Methods

In potential field methods, an obstacle exerts a repulsive force pushing the robot away from the obstacle, while an attracting force pulls the robot toward the target. *Virtual Force Field* (VFF) is proposed

in Borenstein and Koren (1989) and *Vector Field Histogram* (VFH) in Borenstein and Koren (1991). Hydrodynamic potentials are applied in Sugiyama *et al.* (2010) for mobile robots to avoid moving obstacles, where static and moving obstacles are represented with different hydrodynamic potentials. One disadvantage of potential field methods is the risk to get stuck in a local minima.

Other Reactive Methods

Minguez and Montano (2004) develops Nearness Diagram (ND) navigation algorithm for driving a robot in obstacle-dense space. The proximity of obstacles and areas of free spaces are constructed from the depth map sensor information with a nearness diagram, and the best collision free path is chosen. In Watanabe *et al.* (2007), A collision cone approach is used as a collision criteria to detect any obstacle that is critical to the vehicle. The Minimum-effort guidance is used as the collision avoidance strategy, in which the helicopter lateral accelerations are minimized. Collision cone approach is also used in Chakravarthy and Ghose (1998) for detection and avoidance of collision with moving obstacles. Saunders *et al.* (2005) proposes generating avoidance triangles around the obstacles and using the sides of the triangles as possible waypoint paths. This is used as the local path planner of a fixed wing UAV. In Hrabar (2011), an expanding elliptical search is performed to find an Escape Point - a waypoint which offers a collision free route past obstacles and towards a goal waypoint. The reactive planner of the UAV in Scherer *et al.* (2008) is based on a control law for obstacle avoidance in people avoiding point obstacles studied in Warren *et al.* (2001), in which the UAV tries to avoid obstacles and reach a goal by turning in the direction where the obstacle repulsion and goal attraction are at an equilibrium.

Chapter 3

HARDWARE AND SYSTEM DESCRIPTION

3.1 Overview

In this chapter, I describe all the hardware and system that are used in next few chapters. Software architectures are also included. Readers can skip this chapter and use it as the reference when reading the next few chapters. In the rest part of this chapter, each section corresponds to the system used in each chapter that follows.

3.2 Systems in Chapter 4

In chapter 4, an image based reactive avoidance method was developed. In the experiments, videos of the moving obstacle were collected using an forward facing camera attached to the bottom of a quadrotor. The videos were used to generate avoidance action for the quadrotor in ROS simulation. The action was executed by the quadrotor in ROS simulation environment.

3.2.1 Video Capture

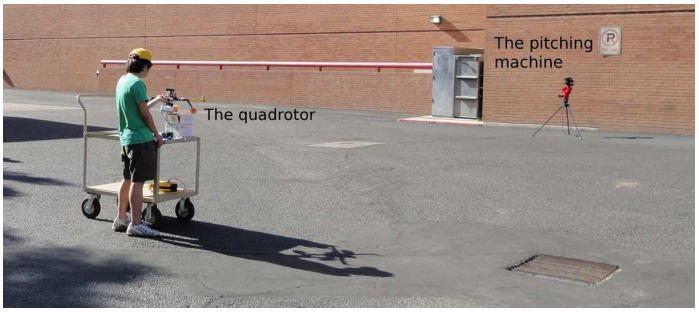
Our experiment setup is in Figure 3.1. A ball was delivered from a pitching machine towards a quadrotor, which was manually held to emulate hovering. The forward facing camera at the bottom of the quadrotor captured images of the moving ball. *Ptgrey CMLN-13S2C-CS* was used as the camera. It captured images at 18 FPS. The pitch machine delivered a color ball with speeds up to 45mph (20m/s). The diameter of the ball is approximately 70 mm. Figure 3.2 shows two captured images with the ball in it. The quadrotor was separated from the pitching machine by about 18m. We chose such distance so that the ball may reach the dangerous ambience of the quadrotor after traveling over the distance.

3.2.2 ROS Simulation

Velocity command for the quadrotor to avoid the incoming moving obstacle were generated by a method that will be described in chapter4. The block diagram of the system for validating the algorithm is in Figure 3.3. The algorithm calculates the avoidance action for the quadrotor from obstacle images. The actions are converted to velocity command and sent to the quadrotor simulation node in ROS to

(a) the quadrotor and the camera

(b) the pitching machine



(c) the arrangement of the quadrotor and the pitching machine

Figure 3.1: The Experiment Setup for Capturing Images of a Moving Obstacle.

(a)

(b)

Figure 3.2: Images of the Ball Launched by the Pitch Machine.

execute. *hector_quadrotor* [1] ROS package are used for the simulation. The *Gazebo* environment for

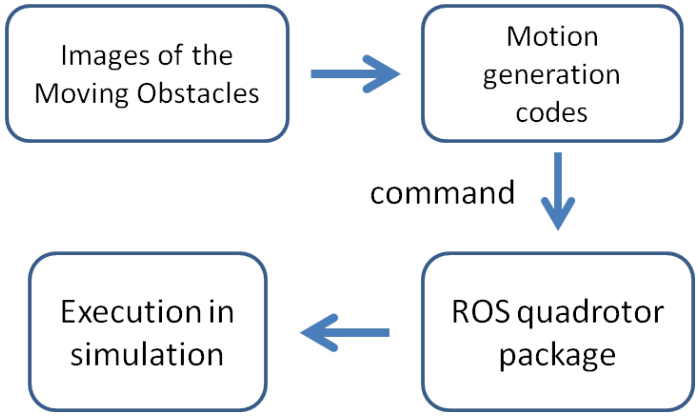


Figure 3.3: The block diagram of the system for validating the potential field based avoidance method. Avoidance motions are generated from images of the moving obstacle and converted to velocity command for the quadrotor. The ROS node of the quadrotor simulator will receive and execute the command in simulation.

visualizing the simulation is shown in Figure 3.4. The environment includes a quadrotor, a simplified pitch machine and a ball, and a background object.

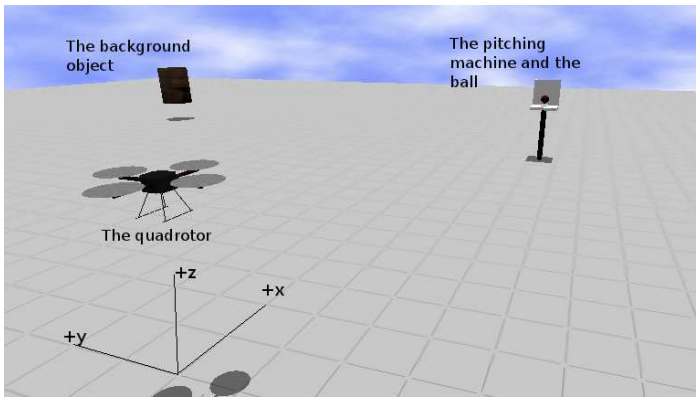


Figure 3.4: The ROS Gazebo simulation environment to validate the potential based method. It includes the quadrotor, the pitching machine and the ball, and a background object.

3.3 Systems in Chapter 5

This section describes ADS-B data collection used in chapter 5. ADS-B data collection is also needed in chapter 8. Chapter 5 uses a low cost ADS-B receiver that is able to receive only 1090 MHZ ADS-B transmission. Chapter 8 uses an aviation level ADS-B receiver that works for both 978 MHZ and 1090 MHZ. The receiver is much more expensive and is supposed to mount on a real aircraft. In chapter 5,

[1] `http://wiki.ros.org/hector_quadrotor`

(a) (b)

Figure 3.5: GNS 5890 ADS-B Receiver USB-Stick and Its Antenna.

a GNS 5890 ADS-B Receiver USB-Stick [2] is used to receive ADS-B data transmitted from flying-by aircraft around PHX Sky Harbor international airport. The USB device and its antenna are shown in Figure 3.5. The ADS-B data are decoded and logged using modified open-source code [3] . The data of a single aircraft includes:

1 the GPS position, latitude and longitude: y, x.

2 the speed over ground (the speed in $x - y$ plane): v_g.

3 the course over ground (the heading in $x - y$ plane): θ, defined in respect to positive y axis (north).

4 the altitude: z.

5 the vertical speed: v_h.

The logged GPS trajectory of aircraft with Flight Number UAL1479 on Dec 10, 2012 is plotted overlapped on Google Map, as shown in Figure 3.6. The trajectory is not continuous due to data transmission dropouts caused by blocking of buildings, absorption by dust or molecules in the air, etc. The length of the trajectory is limited by the receiving range of the device but sufficient for algorithm validation in chapter 5. In the experiments the ADS-B data are read from a previously logged file to emulate ADS-B data flow. In figure 3.6, the gap of the aircraft trajectory due to data dropouts is observed. Extrapolation

[2] http://www.amazon.com/GNS-5890-ADS-B-Receiver-USB-Stick/dp/B006VI3WAK

[3] http://code.google.com/p/adsb-pgr/

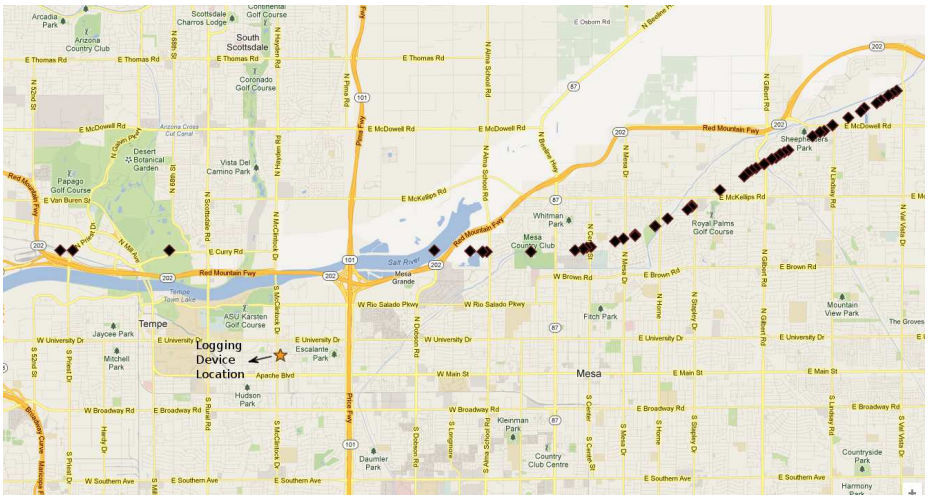


Figure 3.6: The logged trajectory of Flight UAL1479 on Google Map. The star indicates the device's location (33.4190018,-111.9082609). The leftmost (33.4408, -111.963) and rightmost (33.4747,-111.753) of the trajectory spans by 19.547km East-to-West (horizontal) and 3.498km South-to-East (vertical).

is used to fill in the data dropouts gap. Details will be covered in chapter 5. The extrapolated trajectory is shown in Figure 3.7.

3.4 Systems in Chapter 8

3.4.1 Systems Used in Indoor Experiments

In the indoor flight experiments to validate the Closed-Loop RRT based algorithm, an off-the-shelf Parrot AR.Drone was used as the UAV platform. The Parrot AR.Drone has the following specifications:

1 Powered with brushless engines with three phases current controlled by a micro-controller. Using a LiPo battery to fly.

2 Sensors: IMU for automatic stabilization. An ultrasound telemeter providing with altitude measures. A 3-axis magnetometer and a pressure sensor to allow altitude measurements at any height.

3 A camera aiming downwards: a field of view of $47.5° \times 36.5°$ and resolution of 176×144. The other camera is aimed forward: a field of view of $73.5° \times 58.5°$ and a resolution of 320×240. Only images from either of the camera can be transmitted to the laptop.

4 Open source ROS driver [4] for controlling.

Pictures of an AR.Drone are shown in Figure 3.12 An AR.Drone's downward-facing camera is able to

[4] `https://github.com/AutonomyLab/ardrone_autonomy`

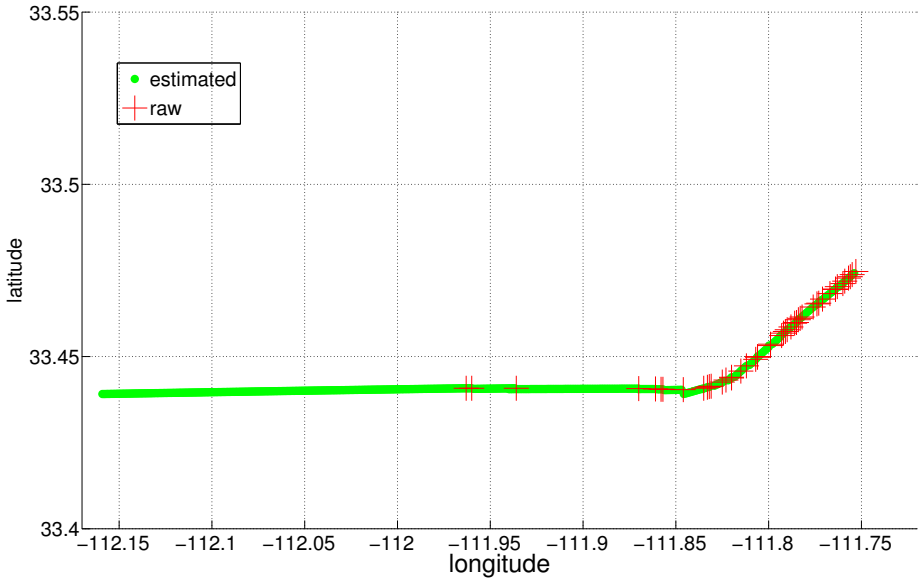


Figure 3.7: The trajectory of UAL479 after extrapolation. The green line shows the extrapolated data and the red cross shows the raw data.

(a) (b)

Figure 3.8: An AR.Drone from the Front and Bottom View.

provide the optical flow. Optical flow combined with the sonar provides the velocity measurement for the AR.Drone. In indoor environment, we do not have GPS or motion capturing system like vicon to provide positioning system. Instead, we use odometry,i.e.,integration of the velocity for navigation. More technical details of the AR.Drone can be found at Bristeau *et al.* (2011). An AR.Drone receives commands from a laptop and sends its onboard sensor readings to it. Wifi is used as the communication. Computation for path planning is performed offboard by the laptop.

The block diagram for the software architecture used in the experiment is shown in Figure 3.9. The path planning node generates the path and sends it to the path executing node. The latter creates

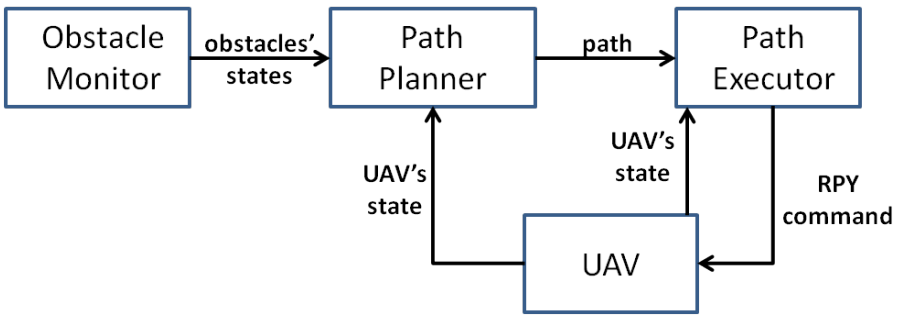


Figure 3.9: The Software Architecture for Indoor Flight Experiments of UAV Moving Obstacle Avoidance.

the velocity command based on the UAV's position relative to the path using the vector field method in Beard and McLain (2013). The velocity command is converted to roll-pitch-yaw command by the *ardrone_autonomy* ROS package and sent to the AR.Drone. The path planning node also receives the AR.Drone's state for path planning. It holds an obstacle monitor to acquire updated obstacles' states. Both actual and virtual moving obstacles are used in the experiments. As demonstrated in Figure 3.10 another AR.Drone is used as the actual obstacle. It sends its velocity and position in real time to another laptop (laptop B) that controls it. Laptop B then sends the information to the laptop that controls the main AR.Drone. This way the path planning node obtains the states of the moving obstacle. Virtual

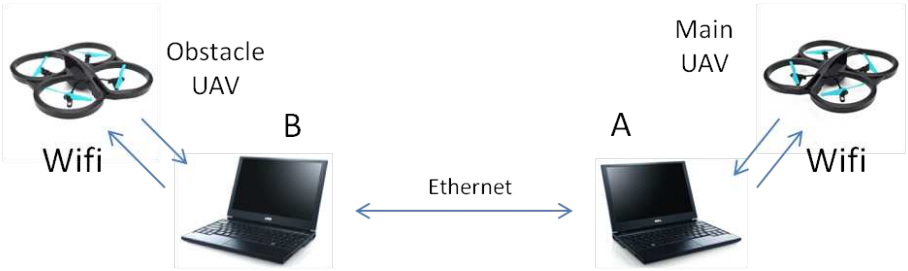


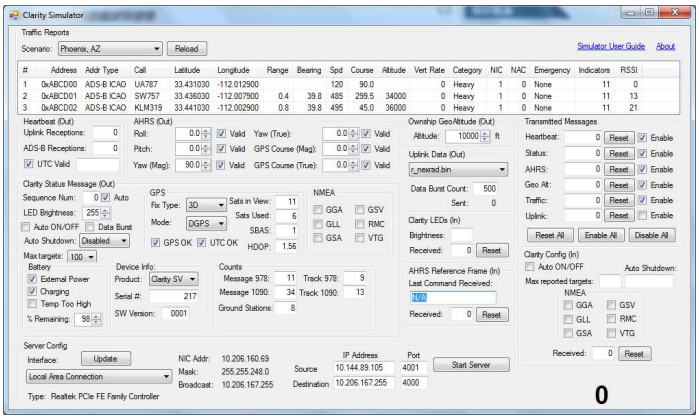
Figure 3.10: The System Setup for Indoor Flight Experiments.

moving obstacles are logged data for quadrotors flying in *hector_quadrotor* ROS simulator. The same simulator is used in section 3.2.2. The path planner reads the data from the logged file to obtain the states of virtual moving obstacles.

3.4.2 Systems in Software-in-the-Loop Experiments

In the experiments, a commercial ADS-B receiver is used to receive ADS-B data of aircraft flying by Phoenix Sky Harbor international airport. Software In the Loop (SITL) simulation is used to validate the algorithm.

(a) Sagetech Clarity ADS-B receiver

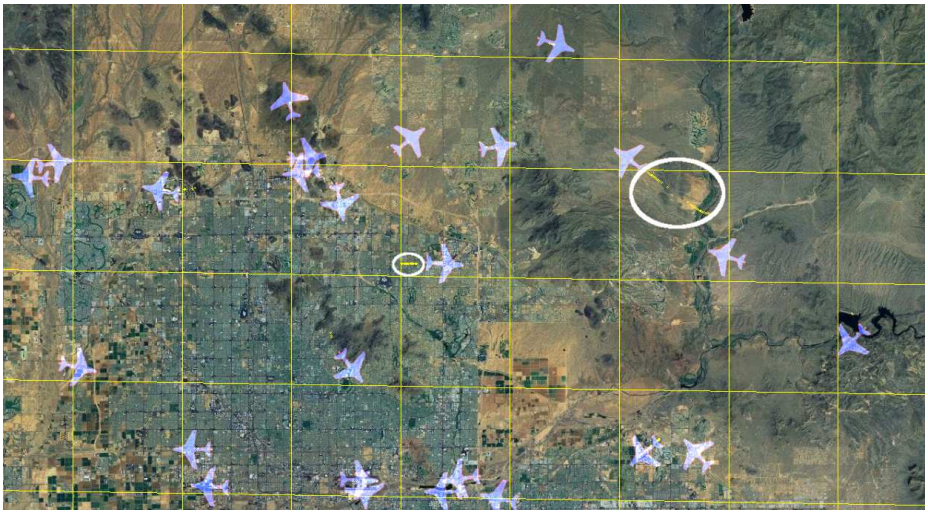


(b) ADS-B simulator interface

Figure 3.11: The Sagetech Clarity ADS-B Receiver and ADS-B Data Simulator.

Receiving ADS-B Data

Section 3.3 also describes ADS-B data acquisition but a low cost receiver is used. This section uses the same ADS-B receiver as is adopted in civil aviation. Figure 3.11a shows the ADS-B receiver. It is supposed to mount on a real aircraft like Cessna 172 to receive ADS-B data sent from other aircraft. The ADS-B receiver receives ADS-B bytes stream and sends the stream to a computer through UDP via Ad Hoc Wireless. The computer runs the software to decode the bytes stream and translate it to human readable data format. We created our own software to decode the bytes stream based on the interface control document provided by the vendor. For SITL experiments, we use the ADS-B receiver to log data and play them back in the simulation to emulate actual aircraft. Figure 3.12a shows the received aircraft ADS-B data in the neighborhood of Sky Harbor airport in a certain time instance. The receiver was



(a) ADS-B aircraft trajectories received by the Clarity receiver.

(b) ADS-B aircraft trajectories from the simulator.

Figure 3.12: Trajectories (yellow dots) of received ADS-B aircraft data. Figure 3.12a shows actual ADS-B from the receiver and Figure 3.12b shows ADS-B data from the simulator software. Only a few aircraft in Figure 3.12a generate yellow dots trajectories. That means they are not being tracked due to poor communication. In 3.12b, all three aircraft from the simulator are being well tracked.

placed at the roof of a building at Arizona State University. The tracked trajectory of each aircraft is shown in yellow dots. We observe that most aircraft were not being tracked and there was data dropout for the ones being tracked. According to *Sagetech*, such phenomenon was caused by block of buildings around and the data communication worked well when the device was mounted in an aircraft flying in the air. We are currently not doing real flight experiments and therefore we selected only aircraft that were tracked by the ADS-B receiver and filled in the data dropout gap by extrapolation based on the heading and velocity. We obtained three aircraft trajectories. Apart from real ADS-B data, we also used the ADS-B simulator software (Figure 3.11b) that came along with *Sagetech Clarity* receiver to obtain three more aircraft. They are shown in Figure 3.12b. The receiver runs in a separate laptop and sends data bytes stream in the same way as the Clarity ADS-B receiver. ADS-B data contain latitude, longitude, altitude, heading, and horizontal and vertical velocities of the aircraft.

Software-in-the-Loop Setup

Software In the Loop (SITL) simulation was used as the first step test before real flight tests. In SITL, the key low level hardware drivers (such as gyros, accelerometers and GPS) run in the same way that they would run in a real flight. The architecture of SITL setup is shown in Fig 3.13. The SITL setup

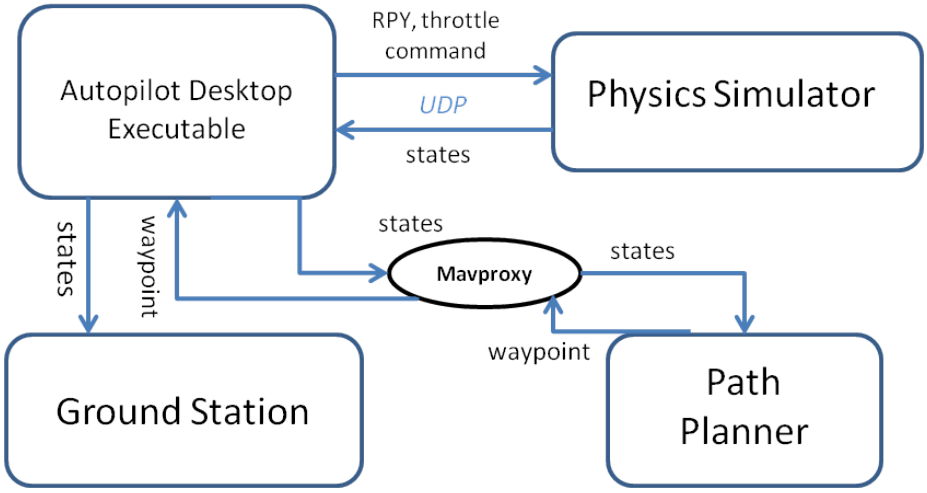


Figure 3.13: Architecture of Software In the Loop simulation. The architecture consists of an autopilot executable, a ground station, a flight simulator and the path planner. The path planner receives the UAV's states from the autopilot and sends a generated waypoint for collision avoidance to the autopilot. The ground station receives the UAV's states from the autopilot to visualize the UAV's states in real time. The flight simulator receives roll-pitch-yaw and throttle command from the autopilot and flies the UAV in simulation.

consists of an autopilot executable, a ground station, a flight simulator, and the path planner. The autopilot executable is the equivalent of onboard autopilot in real flight. It is able to receive high level

commands such as waypoints and converted it low level command such as roll-pitch-yaw and throttle commands. The executable sends the commands to the physics flight simulator. The simulator will fly the UAV according to the commands. The path planner receives the UAV's states from the autopilot and sends the generated waypoint for avoidance to it. The communication is facilitated by *Mavproxy* [5] . It is a Mavlink [6] based communication proxy. It also has a ground station interface. The autopilot sends its states to the ground station in real time for visualization. In the experiment, *APM::Plane* is used as the autopilot and *JSBSim* is used as the physics simulator. Figure 3.14 shows the simulation environment.

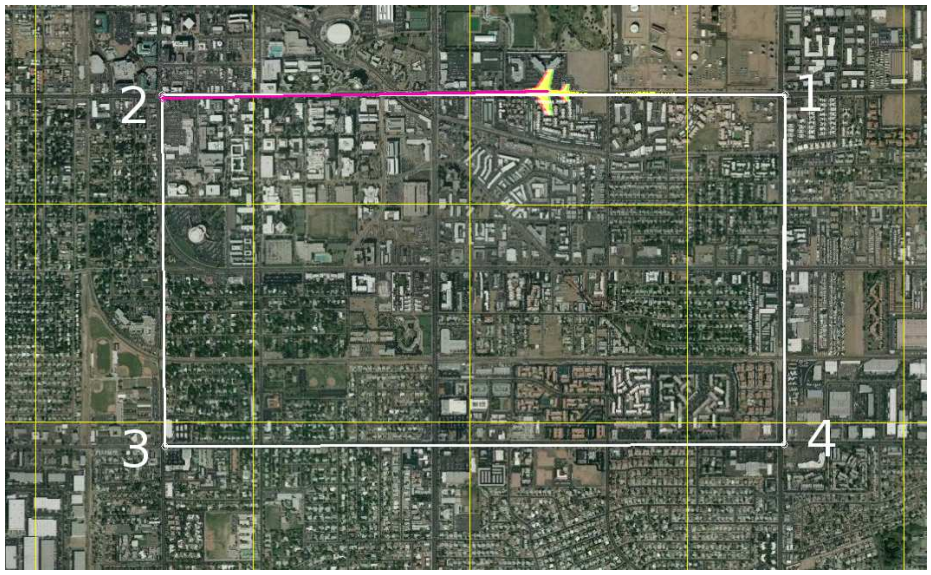


Figure 3.14: The UAV (green icon) in Software In the Loop Simulation.

3.4.3 Systems in Hardware-in-the-Loop Experiments

The HIL setup used in the experiment is shown in Figure 3.15. It consists of a flight simulator, an autopolit board, a computer that runs the HIL software and bridges the simulator and the autopilot, and a computer that generates the avoidance waypoint. The simulator provides the simulated sensor data. The data are fed to the autopilot via the HIL computer. It provides the path planning computer the UAV's states. The path planning computer sends an avoidance waypoint to the autopilot. The autopilot converts the waypoint command into control output and send them to the simulator through the HIL computer. In the experiment, *X-plane 10* is used as the flight simulator, *qgroundcontrol* the HIL software,

[5] `http://tridge.github.io/MAVProxy/`

[6] `http://qgroundcontrol.org/mavlink/start`

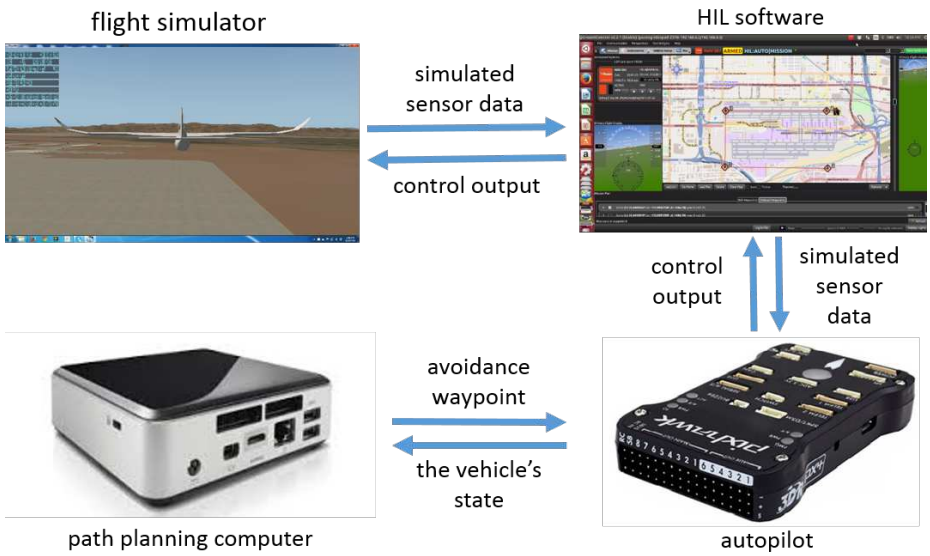


Figure 3.15: The system setup for the Hardware-In-the-Loop experiment. The autopilot board received simulated sensor data from the flight simulator and sends its control output to it. They are bridged by the computer that runs the HIL software. The computer running the path planning programme receives the vehicle's states from the autopilot and send waypoint command for collision avoidance to it.

and *pixhawk* the autopilot board. Compared to Software-In-the-Loop simulation, the system setup of Hardware-In-the-Loop is closer to the system setup used in real flight.

3.4.4 Systems in Outdoor Flight Tests

(a) AR.Drone with GPS receiver: (b) The GPS receiver:

Figure 3.16: The AR.Drone used in the real flight experiments 3.16a and the GPS receiver attached to it 3.16b.

Two Parrot AR.Drones with GPS receivers (the orange device in Figure 3.16b) connected to them via USB are used in the experiment. One of them plays the role as the obstacle aircraft and the other acts as the host UAV to avoid it.

Figure 3.17 explains the setup of the real flight experiments. The host UAV and the obstacle UAV are controlled by two laptops respectively. The laptops also receives the states such as position, velocity, and heading from the UAVs. A laptop communicates with an AR.Drone through wifi. The two laptops are connected by Ethernet. This way the laptop running the path planning algorithm is able to receive the obstacle UAV's states.

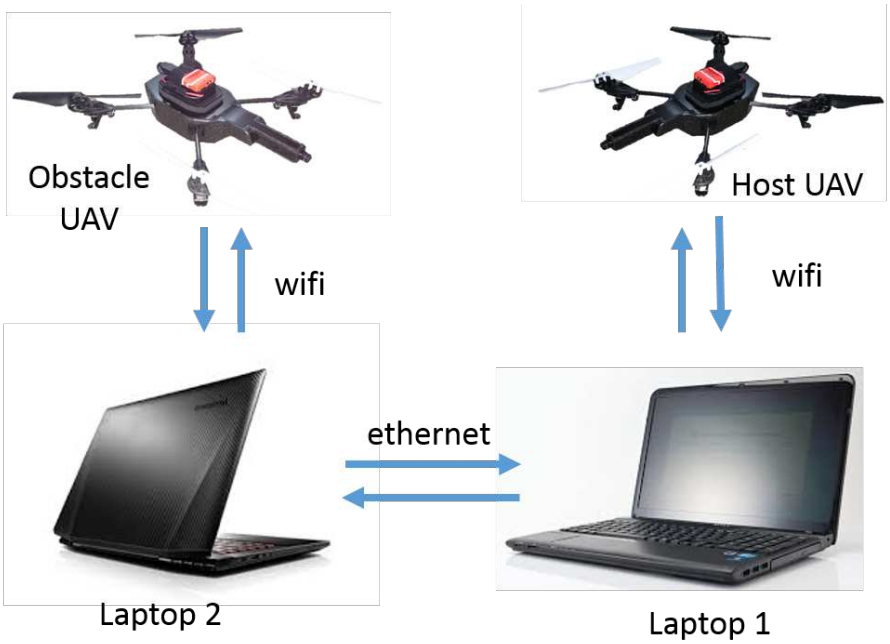


Figure 3.17: The experiment setup for the outdoor real flight tests. AR.Drones are used as the UAV platform. They are attached with a GPS receiver. One AR.Drone is used as the obstacle aircraft and the other one acts as the host UAV to avoid it. They are controlled by respective computers through wifi. Two computers communicate through Ethernet. This is how the path planner receive the states of the obstacle UAV.

Chapter 4

IMAGE BASED REACTIVE AVOIDANCE

4.1 Overview

In this chapter, an image-based reactive method for moving obstacle avoidance is developed. It can be used as the local planner to compensate the global planner in case of its failure.

4.2 Algorithm Description

4.2.1 Moving Obstacle Detection

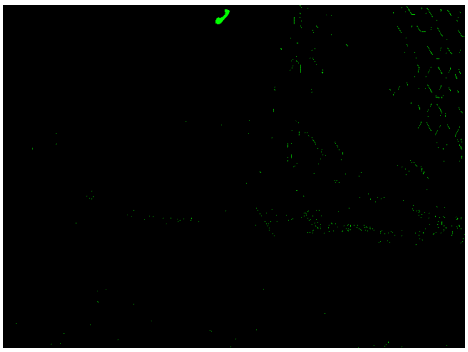
Jung and Sukhatme (2004); Lieb *et al.* (2004) propose two methods of detecting moving objects using a single camera. The basic idea is that pixels or local features indicating possible moving objects are returned from difference images and a particle filter is applied over sequence of difference images to finally detect the moving objects. A number of frames are needed for the convergence of the particle filter so that a moving object can be confirmed. If we use their methods based on our experiment setup, the obstacle may already be quite close to the quadrotor when detected from images. In this case, there will not be sufficient time for the quadrotor to react. Therefore we simplify the detection by assuming known color of the obstacle. Such approach is also used in Lippiello and Ruggiero (2012), where the detection of the moving object was not the main focus of the paper neither. In detail, we go through the following steps:

- Transforming each image into the *Hue, Saturation and Value Color Space (HSV)* to reduce the affect caused by the variations of the environmental lightness.
- Binarization:
  for any pixel (x, y) with color value (H, S, V), if
  $H_{lower} \leqslant H \leqslant H_{upper}$ and $S_{lower} \leqslant S \leqslant S_{upper}$
  and $V_{lower} \leqslant V \leqslant V_{upper}$,
  (x, y) is assigned 1, otherwise 0
- Performing blob detection Chang *et al.* (2004) on the binarized images

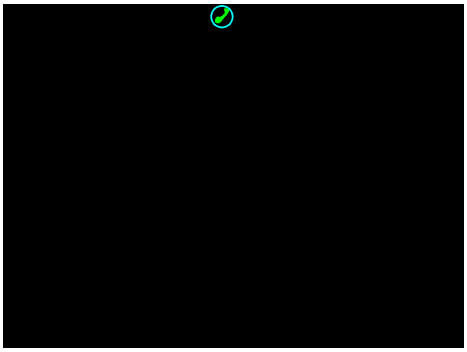
(a) The original image

(b) The image in HSV space

(c) image binarization



(d) blob detection and thresholding



(e) obstacle detected

Figure 4.1: Moving obstacle detection from an image. The obstacle is marked with a circle in image (e).

- A threshold for the blobs' area is applied and the centroid of the blob with the largest area in the remaining ones is is evaluated as an ideal approximation of the obstacle's center.

where H_{lower}:H_{upper}, S_{lower}:S_{upper} and V_{lower}:V_{upper} are pre-determined based on the ball's color in HSV color space. The threshold in the last step is necessary because if we pick the blob with the largest area without thresholding, there will be false positives in images without the obstacle. Figure 4.1 shows the output after each step in the algorithm. Firuge 4.1b is the image in HSV color space. Figure 4.1c gives us a large white spot (which is the obstacle) with some noise blobs. Figure 4.1d marks the blobs detected in green. Figure 4.1e shows the obstacle that has been detected. Moving obstacles in 97 frames are detected out of total 142 frames with the obstacle. The obstacle was missed in images where the obstacle was too much far away from the UAV and therefore too small in the image or the strong illumination from the sun made the ball lose its original color in the image. There is no false detection since we deliberately chose the background with different color from the ball.

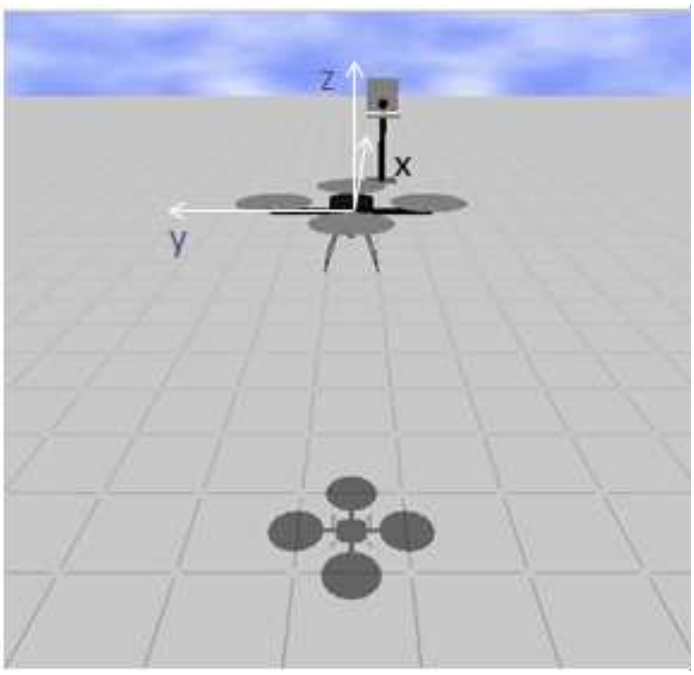


Figure 4.2: Demonstration of UAV's reacting plane (the y-z plane). The xyz coordinate system is fixed w.r.t the world but coincident with the quadrotor's center at the moment.

4.2.2 Potential Field Based Avoidance

Generation of the trajectory to avoid the moving obstacle A monocular camera is unable to provide the actual distance between the obstacle and the quadrotor. All it can provide is the 2D projection of

the obstacle's motion onto the camera's image plane. We restrict the UAV's motion in a plane parallel parallel to the image plane to make use of such projection. Referring to Figure 4.2, if the camera faces along $+x$ direction, the quadrotor will react in y-z plane. Next we discuss how to generate the avoiding trajectory in the plane.

Transforming the obstacles image position to an global reference frame We use the image frame where the ball was initially detected as the reference frame where the trajectory will be generated and denote it as $frame\,0$. Image positions of obstacles detected in later frames need to be evaluated in $frame\,0$ by transformation. Supposing that the obstacle in $frame\,i$(where the obstacle was detected for the ith time, $i > 0$) is $\mathbf{x}_b^i$, it can be transformed into $frame\,0$ via $\mathbf{x}_b^{i\prime} = H\mathbf{x}_b^i$ where H is the homography from $frame\,i$ to $frame\,0$, calculated as follows:

- Applying Shi & Tomasi's corner detector Shi and Tomasi (1994) to $frame\,i$, returning corner feature points $\mathbf{X}_f = [\mathbf{x}_1, \mathbf{x}_2 ... \mathbf{x}_n]$
- $\mathbf{X}_f$ are tracked in $frame\,0$ as $\mathbf{X}'_f = [\mathbf{x}'_1, \mathbf{x}'_2 ... \mathbf{x}'_n]$, with a pyramidal implementation of the Lucas-Kanade optical flow method Lucas *et al.* (1981).
- H is calculated by $H = \arg\min_H \left\| \mathbf{X}'_f - H\mathbf{X}_f \right\|_F$, where F means *Frobenius* matrix norm.

With $\mathbf{x}_b^{i\prime} (i = 1, 2, 3...)$, we next generate the avoiding trajectory.

Trajectory Generation The image center O_c(from camera calibration) of $frame\,0$ indicates the camera's position in that image. We can approximate it as the image position of the UAV in that image since the camera is close to the mass center of the UAV. Imagining that we initially place a mass point P (which represents the UAV) at O_c, $\mathbf{x}_P^0 = O_c$ is the position of P. P will be driven to a new point $\mathbf{x}_P^i$ incrementally when the obstacle is detected for the ith time. We design that $\mathbf{x}_P^i (i \geqslant 0)$ is updated to $\mathbf{x}_P^{i+1}$ as follows:

$$d\mathbf{x}_a = attr(\mathbf{x}_P^i, O_c) \tag{4.1}$$

$$d\mathbf{x}_r = rep(\mathbf{x}_P^i, \mathbf{x}_b^{i\prime}) \tag{4.2}$$

$$d\mathbf{x}_g = gnd(\mathbf{x}_P^i) \tag{4.3}$$

$$d\mathbf{x} = d\mathbf{x}_a + d\mathbf{x}_r + d\mathbf{x}_g \tag{4.4}$$

$$\mathbf{x}_P^{i+1} = \mathbf{x}_P^i + d\mathbf{x} \tag{4.5}$$

Where $attr(\mathbf{v}, \mathbf{v}_o)$ represents the attractive potential field on $\mathbf{v}$ by source $\mathbf{v}_o$; $rep(\mathbf{v}, \mathbf{v}_o)$ is the repulsive potential field exerted on $\mathbf{v}$ by the source $\mathbf{v}_o$; $gnd(\mathbf{v})$ is the repulsive potential field from the ground. We have equation (1) because the UAV has the trends to return to its original location. (2) means that UAV needs to avoid the obstacle. (3) is introduced to prevent the UAV from crashing the ground.

Assuming $\mathbf{v} = (x, y)$, $\mathbf{v}_o = (x_o, y_o)$, the mathematical expression (modified from Goodrich (2002)) of the 3 types of potentials are:

(1) attractive potential:

$attr(\mathbf{v}, \mathbf{v}_o) = (attr\_1d(x, x_o), attr\_1d(y, y_o))$,

where $attr\_1d(x, x_o) =$

$$\begin{cases} 0, \quad |x - x_o| < d \\ -\alpha * (|x - x_o| - d) * sign(x - x_o), \\ \qquad |x - x_o| \geqslant d \quad and \quad |x - x_o| \leqslant d + t \\ -\alpha * t * sign(x - x_o), \quad |x - x_o| > d + t \end{cases} \tag{4.6}$$

(2) repulsive potential:

$rep(\mathbf{v}, \mathbf{v}_o) = (rep\_1d(x, x_o), rep\_1d(y, y_o))$,

where $rep\_1d(x, x_o) =$

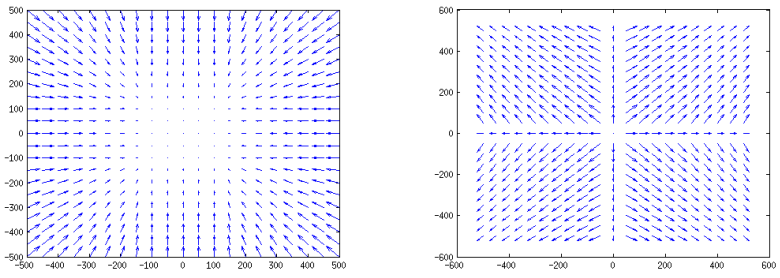
$$\begin{cases} sign(x - x_o) * 100, \quad |x - x_o| < r \\ \beta * (s + r - |x - x_o|) * sign(x - x0), \\ \qquad |x - x_o| \geqslant r \quad and \quad |x - x_o| \leqslant r + s \\ 0, \quad |x - x_o| > r + s \end{cases} \tag{4.7}$$

(3) ground effect potential:

$gnd(x, y) = (0, dy)$, where

$$dy = \begin{cases} dy = -\kappa(y - h), \quad y \leqslant h \\ 0, \quad y > h \end{cases} \tag{4.8}$$

Attractive and repulsive force are sketched in Figure 4.3, where $d = r = 100, s = t = 900, \alpha = 0.1, \beta = 0.1$.



(a) the attractive force field (b) the repulsive force field

Figure 4.3: The Attractive and the Repulsive Force Fields.

The process of updating $\mathbf{x}_P^i$ is demonstrated in Figure 4.4. When

$$|\mathbf{x}_P^i - O_c| \geqslant L \tag{4.9}$$

(where L is a preset threshold), the vector $\overrightarrow{\mathbf{x}_P^i - O_c}$ (the yellow line in Figure 4.4l is the trajectory for the quadrotor. A velocity command $\mathbf{Vel} = k\overrightarrow{(x_q^i - \mathbf{x}_q^0)}$ is sent to the quadrotor followed by a stop command after a time interval. The command is executed in ROS simulation using the *hector_quadrotor* package, which is in charge of lower level dynamic control of the quadrotor.

4.3 Experiments

The ROS simulation environment is shown in Figure 3.4. The quadrotor, the pitching machine (simplified for drawing) with the ball and a background object are shown. We place the quadrotor at $(0, 0, 1.5)$ in gazebo, around which it hovers. The quadrotor is not static due to dynamics. The ball is designed to launched from (x_s, y_s, z_s) to (x_f, y_f, z_f) (using the coordinate system in Figure 3.4). They are specified as follows: supposing m frames are cost in total to track the ball from its appearance till it disappears and the kth$(k > 0)$ frame is the one where the ball is detected for the first time , we have $x_s = \frac{m-k}{m}S$ where S is the measured actual distance along the ground from the pitching machine to the quadrotor. If x_s, the depth according to our coordinate system, is known, y_s and z_s can be calculated with the ball's image coordinate in kth frame and the calibration of the camera. By specifying like this, we actually imagine that the ball is launched from the moment and the location it is detected for the first time. $x_f = f\frac{D}{d_{img}}$, where f is the camera's focal length and D and d_{img} are the ball's actual and image diameter. Knowing x_f, y_f and z_f can be calculated with the ball's image coordinate in mth frame and the calibration of the camera. The magnitude of the velocity of the ball is 20.1168m/s (the maximal speed that the pitching machine provides). The quadrotor and its control is provided by

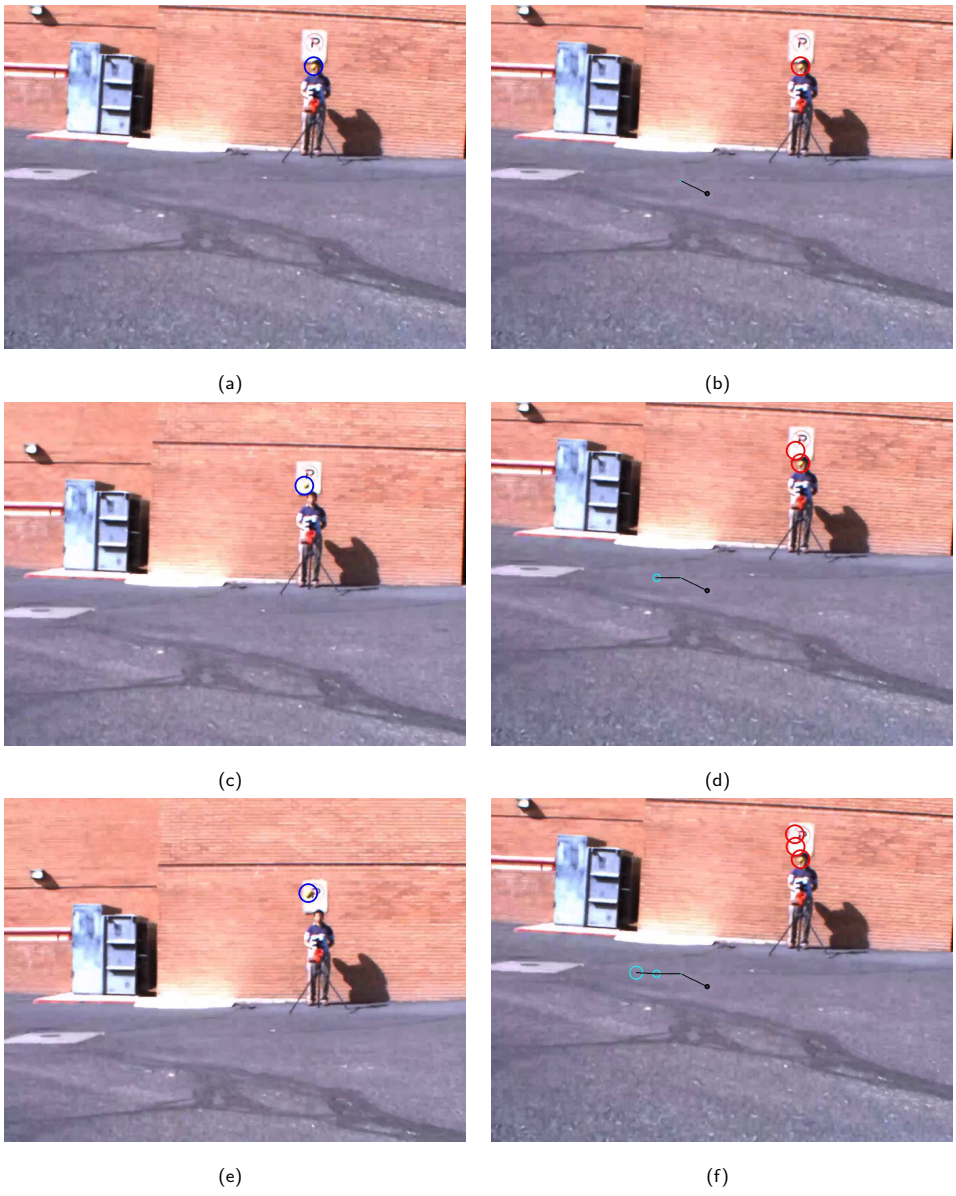
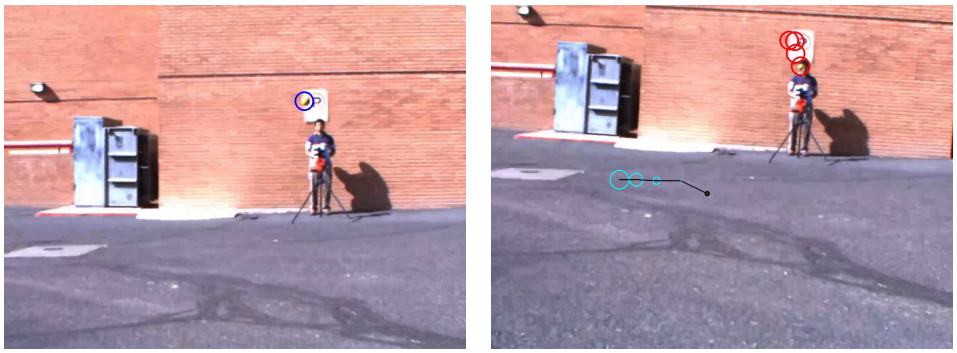
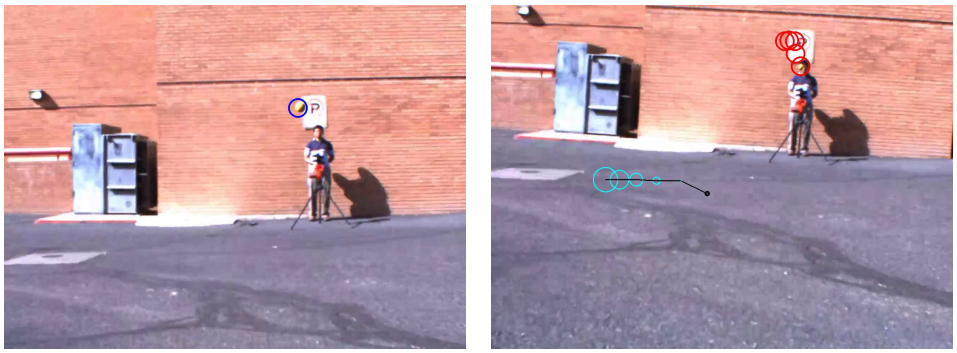


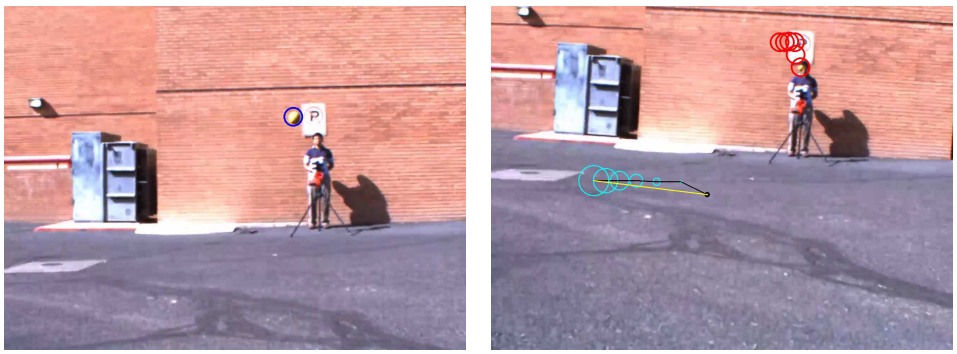
Figure 4.4: Obstacle avoiding trajectory generation for the qudrotor. The left column shows the original images where the detected obstacle is circled with blue. The right shows how $\mathbf{x}_P^i (i = 1, 2, ...)$ (the center of the largest white circle) is updated. The black dot is the image center O_c. The red circles indicate the obstacles' position in $frame\,0$ after homography transform. The yellow line is the generated trajectory (starting from the image center).



(g) (h)



(i) (j)



(k) (l)

Figure 4.4: Obs-avoiding trajectory generation for the qudrotor. The left column shows the original images where the detected obstacle is circled with blue. The right shows how $\mathbf{x}_P^i (i = 1, 2, ...)$ (the center of the largest white circle) is updated. The black dot is the image center O_c. The red circles indicate the obstacles' position in $frame\,0$ after homography transform. The yellow line is the generated trajectory (starting from the image center).

Throw	$detect\_dis$	dis0	dis1
1	11.87	0.476	0.531
2	14.03	0.305	0.983
3	12.95	0.386	1.162
4	15.11	0.373	0.923
5	12.95	0.493	0.943
6	11.87	0.293	0.759
7	7.82	0.395	0.410
8	10.80	0.401	1.063
9	7.82	0.32	0.5006

Table 4.1: The closest distance between the quadrotor and the ball with (dis1) and without (dis0) avoiding reaction. The detect_dis means is the distance between the quadrotor and the ball when the ball is detected from the images for the first time.

hector_quadrotor ROS package. The maximal extension of the quadrotor from its mass center is 0.38(x), 0.38(y) and 0.35(z) meter. When define the safety volume for the quadrotor as a sphere of radius 0.5m. Table 4.1 summarizes the closest distance (dis1) between the quadrotor and the approaching ball during the procedure the quadrotor tries to avoid the ball when L in equation 4.9 is set as 150. $detect\_dis$ means the detection distance, which is defined as the distance between the quadrotor and the ball when the ball is detected from the images for the first time. We have $detect\_dist = x_s$ according to last paragraph. The closest distance when the avoiding commands are not sent are also listed for comparison (dis0). We see only in the 7th throw the ball goes into the safety volume. It is because the detection distance is relatively small and the ball is closer than other throws when the quadrotor starts the avoiding maneuver. In the 9th throw where the detection distance is the same as that of 7th, the closest distance is just 0.0006 meter larger than the safety volume. The closest distance in 1st throw is not much larger than 5m. In that throw, the ball is detected relatively far away initially but it is not detected in next 3 frames due to strong illumination of sunlight. Therefore, the condition $|\mathbf{x}_P^i - O_c| \geqslant L$ will be satisfied relatively late and the command is sent late too.

Based on Ebdon and Regan (2004), the required range for detection of moving obstacle can be formulated as

$$R_{detect} = |\mathbf{V}_{uav} - \mathbf{V}_{obstacle}| * (t_{react} + t_{maneuver}) \tag{4.10}$$

Where $|\mathbf{V}_{uav} - \mathbf{V}_{obstacle}|$ means the magnitude of relative velocity between the UAV and the obstacle. t_{react} represents the time consumed from initial detection of the obstacle to sending the reaction command. $t_{maneuver}$ is the time for execution of the command. We examine the efficacy of our obstacle detection method by calculating the required detection range using equation 4.10. The results are in table 4.2. In our case, $\mathbf{V}_{uav} \approx 0$ since it is hovering. We only examine the obstacle (the ball)'s velocity in x-y plane since its z-velocity is not constant. $|\mathbf{V}_{uav} - \mathbf{V}_{obstacle}|$ equals to the magnitude of the

Throw	$detect\_dis$	t_{react}	$t_{maneuver}$	Vb_{xy}	R_{detect}
1	11.87	0.197	0.307	19.8199	9.9892296
2	14.03	0.163	0.326	19.7241	9.6450849
3	12.95	0.145	0.268	19.7239	8.1459707
4	15.11	0.170	0.351	19.697	10.262137
5	12.95	0.122	0.217	19.8224	6.7197936
6	11.87	0.084	0.357	19.8293	8.7447213
7	7.82	0.036	0.352	19.8715	7.710142
8	10.80	0.035	0.220	19.7811	5.0441805
9	7.82	0.030	0.315	19.8717	6.8557365

Table 4.2: Table for Evaluating the Efficacy of Moving Obstacle Detection.

projection of the obstacle's velocity in x-y plane (Vb_{xy}). $t_{maneuver}$ is specified as the time needed to achieve the half of the quadrotor's maximal velocity during the whole avoiding maneuver. We observe that $detect\_dis > R_{detect}$ for all 9 throws but in 7th throw the two are quite close which results in the incursion of the ball into the quadrotor's safety volume.

Chapter 5

GEOMETRY METHOD BASED ON DUBINS CURVE

5.1 Problem Formulation

To avoid moving obstacles, we must ensure that the obstacles do not enter the UAV's safety volume. In this chapter, a sphere is used as the safety volume as the same as in Hrabar (2011). As well as maintaining a safe separation to the obstacle, the global planner aims at minimizing the path's length and control efforts. The path length reflects the deviation from the original route. Based on the discussion above, the formulation of the global path planning problem is:

$$\begin{aligned} \mathbf{v}^*(t) \ = \quad & \underset{\mathbf{v},t_f}{\operatorname{argmin}} \int_0^{t_f} \left(\|\mathbf{v}\| + \|\frac{d\mathbf{v}}{dt}\|^2 \right) dt, \\ \text{subject to} \quad & \text{kinematic constraints of } \mathbf{v}, \\ & \forall t \in [0, t_f], \|\mathbf{x}(t) - \mathbf{x}_o(t)\| \geqslant L, \\ & \mathbf{x}(0), \mathbf{x}(t_f) \in \mathcal{X}_{ori}. \end{aligned} \tag{5.1}$$

In the problem, we calculate optimal velocity $\mathbf{v}^*(t)$ instead of direct control input $\mathbf{u}^*(t)$, assuming that the UAV's low level controller is able to execute the velocity. This approach has the advantage of being universal to different types of UAVs while calculation of control input is more specific to an UAV's dynamic model, therefore more restrictive. We restrict the initial and final position of the UAV to the original route $\mathcal{X}_{ori}$ so that the UAV commences the avoidance from the original route and return to it when completion. We also add the restriction that the UAV's position $\mathbf{x}(t)$ should at least separate from the obstacle's position $\mathbf{x}_o(t)$ by distance L, the radius of the UAV's safety separation zone. It is determined by the position estimation accuracy of both UAVs and obstacles. In this work, we assume that the UAV's state information is accurate and only deal with uncertainty in the position of the obstacle. The kinematic constraints are specific to different types of vehicles. For example, the kinematic constraint of a fixed wing UAV in 2D is

$$\begin{aligned} v_x &= \|\mathbf{v}\| cos\psi \\ v_y &= \|\mathbf{v}\| sin\psi \\ -c \leqslant \dot{\psi} \leqslant c, \ & v_{min} \leqslant \|\mathbf{v}\| \leqslant v_{max} \end{aligned} \tag{5.2}$$

in which v_x and v_y are velocities in x and y direction respectively. $\|\mathbf{v}\|$ is the magnitude of velocity and ψ is the heading. c is maximal yaw rate and v_{min}/v_{max} is the maximal/minimal velocity. The cost function $\|\mathbf{v}\| + \lambda \|\frac{d\mathbf{v}}{dt}\|^2$ multiplied by dt represents the path length plus the curve smoothness at time t in interval dt, where $\lambda > 0$ is a chosen weight. We also seek an optimal finish time t_f to finish the avoidance procedure as soon as possible so that total deviation and curve smoothness can be minimized. The whole avoidance procedure is composed of collision detection , collision avoidance and return to route. In this work, collision is detected based on ADS-B data from the moving obstacle (aircraft). Details of ADS-B data collection are covered in section 3.3.

5.2 Algorithm Description

5.2.1 Collision Detection

Moving Obstacle State Estimation

To detect collision of the obstacles with the UAV, we need to estimate the position and velocity of the obstacles at every moment. W use an Extended Kalman Filter:

$$\hat{x}_{i+1} = x_i + v_{g_i} * sin(\theta_i) * dt \tag{5.3}$$

$$\hat{y}_{i+1} = y_i + v_{g_i} * cos(\theta_i) * dt \tag{5.4}$$

$$\hat{\theta}_{i+1} = \theta_i + (\widetilde{\theta}_{i+1} - \widetilde{\theta}_i) \tag{5.5}$$

$$\hat{v_{g_{(i+1)}}} = v_{g_i} + (\widetilde{v_{g_{(i+1)}}} - \widetilde{v_{g_i}}) \tag{5.6}$$

$$\hat{z}_{i+1} = z_i + v_{hi} * dt \tag{5.7}$$

$$\hat{v_{h(i+1)}} = v_{hi} + (\widetilde{v_{h(i+1)}} - \widetilde{v_{hi}}) \tag{5.8}$$

$$\hat{P}_{i+1} = F_i * P_i * F_i^T + Q_i \tag{5.9}$$

$$S_{i+1} = H_{i+1} * \hat{P}_{i+1} * H_{i+1}^T + R_{i+1} \tag{5.10}$$

$$K_{i+1} = \hat{P}_{i+1} * H_{i+1}^T * S_{i+1}^{-1} \tag{5.11}$$

$$X_{i+1} = \hat{X}_{i+1} + K_{i+1} * (Z_{i+1} - \hat{X}_{i+1}) \tag{5.12}$$

$$P_{i+1} = (I - K_{i+1} * H_{i+1}) * \hat{P}_{i+1} \tag{5.13}$$

where $X_i = (x_i, y_i, \theta_i, v_{g_i}, z_i, v_{hi})^T$ is the obstacle (in this case the aircraft) estimated state including the position, heading, horizontal and vertical velocities. $\hat{X}_i$ is the predicted state with the obstacle's kinematic model in equations 5.3-5.8 and $Z_i = (\widetilde{x}_i, \widetilde{y}_i, \widetilde{\theta}_i, \widetilde{v_{g_i}}, \widetilde{z}_i, \widetilde{v_{hi}})$ is the measurement of the state

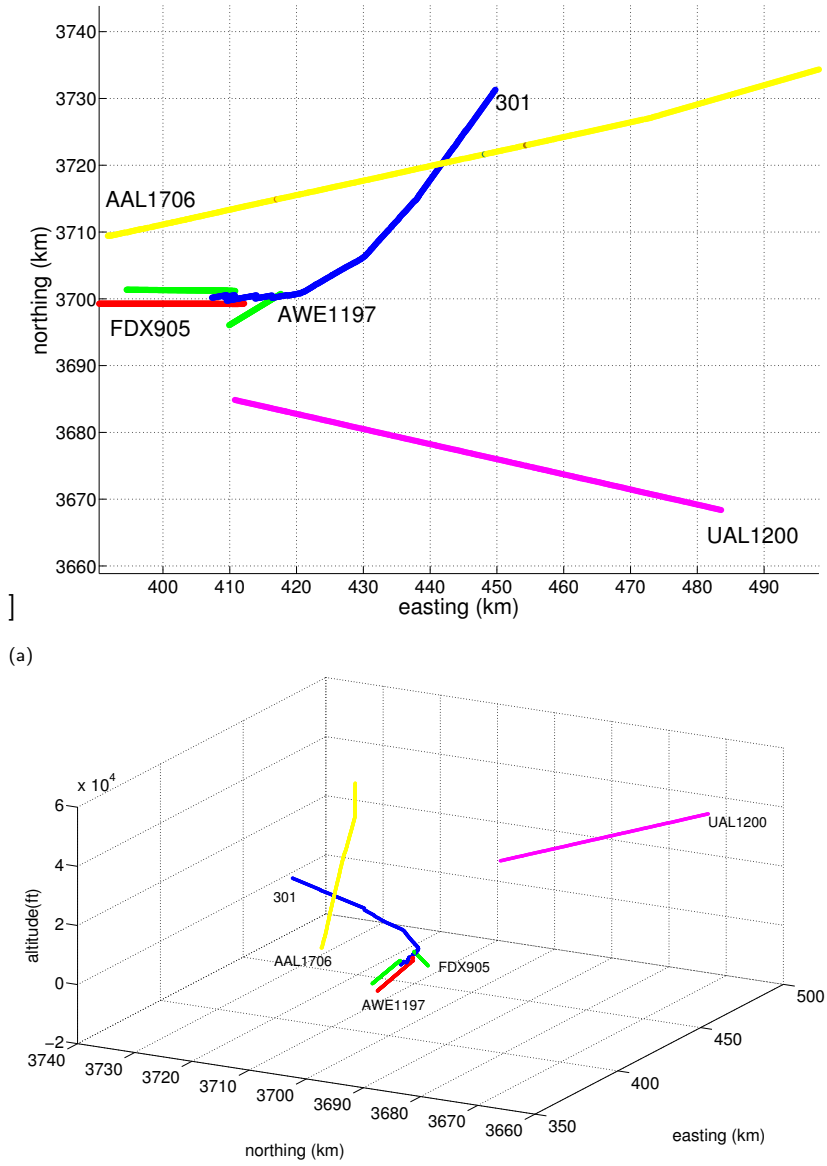


Figure 5.1: Aircraft in the air during a short time interval of data logging, in 2D overhead look (a) and 3D (b).

from ADS-B. Matrix $P_i, F_i, Q_i, S_i, H_i, R_i, K_i$ are the same as defined in standard EKF (Kalman *et al.* (1960)). dt is the time step and in this work we use 1 second, the same as ADS-B updating cycle. When ADS-B dropout happens, we modified the equations above to:

$$x_{i+1} = x_i + v_{g_i} * sin(\theta_i) * dt \tag{5.14}$$

$$y_{i+1} = y_i + v_{g_i} * cos(\theta_i) * dt \tag{5.15}$$

$$\theta_{i+1} = \theta_i \tag{5.16}$$

$$v_{g(i+1)} = v_{gi} \tag{5.17}$$

$$z_{i+1} = z_i + v_{hi} * dt \tag{5.18}$$

$$v_{h(i+1)} = v_{hi} \tag{5.19}$$

$$P_{i+1} = F_i * P_i * F_i^T + Q_i \tag{5.20}$$

Figure 5.1 plots the estimated trajectories of aircraft appearing in the air during a short time interval of data logging, both in 2D and 3D. Latitude and longitude are converted into easting and northing, the geographic Cartesian coordinates for the same point, so that spherical coordinate is not needed for calculation.

collision Estimation

Based on the estimated state of an aircraft, we can calculate its collision point with the UAV. We assume that at a time point t_0 the aircraft is at $\mathbf{x}_{air}$ with velocity $\mathbf{v}_{air}$ and the UAV is at $\mathbf{x}_{uav}$ with velocity $\mathbf{v}_{uav}$. The distance between the two after time interval Δt is $dis = |\mathbf{x}_{air} - \mathbf{x}_{uav} + (\mathbf{v}_{air} - \mathbf{v}_{uav}) * \Delta t|$. The closest distance can be calculated as the following:

$$\begin{aligned} \Delta t^* &= \underset{\Delta t}{\operatorname{argmin}} |\mathbf{x}_{air} - \mathbf{x}_{uav} + (\mathbf{v}_{air} - \mathbf{v}_{uav}) * \Delta t| \\ &= -\frac{(\mathbf{x}_{air} - \mathbf{x}_{uav}) * (\mathbf{v}_{air} - \mathbf{v}_{uav})}{|\mathbf{v}_{air} - \mathbf{v}_{uav}|^2} \end{aligned} \tag{5.21}$$

$$dis_{min} = |\mathbf{x}_{air} - \mathbf{x}_{uav} + (\mathbf{v}_{air} - \mathbf{v}_{uav}) * \Delta t^*| \tag{5.22}$$

$$\mathbf{x}^*_{uav} = \mathbf{x}_{uav} + \mathbf{v}_{uav} \Delta t^* \tag{5.23}$$

If $dis_{min} < L$, where L is the safe separation threshold between the aircraft and the UAV as defined in equations (5.1), the aircraft will enter the UAV's safety volume. This is defined as a collision. However, at this stage we cannot predict that a collision will necessarily happen since $\mathbf{x}_{air}, \mathbf{x}_{uav}, \mathbf{v}_{air}, \mathbf{v}_{uav}$ will still change with time.

We assume that a collision will happen only when both of the following conditions are satisfied:

$$\begin{aligned} dis_{min} &< L \\ dis(\mathbf{x}_{uav}, \mathbf{x}^*_{uav}) &< D \end{aligned} \tag{5.24}$$

In this case, we define $\mathbf{x}^*_{uav}$ in equation (5.23) as the collision point. $dis(\mathbf{x}_{uav}, \mathbf{x}^*_{uav})$ stands for the distance between the UAV's current position and collision position. The two conditions ensure that the UAV must be sufficiently close to the collision point, otherwise a collision cannot be confirmed. The threshold D is set in reference to the length needed for the UAV to execute the optimal avoiding path based on its kinematic constraints. Uncertainty of $\mathbf{x}^*_{uav}$ is determined by that of Δt^*, which can be expressed as:

$$\delta(\Delta t^*) = \sqrt{\sum_{l=1}^{6} (\frac{\partial \Delta t^*}{\partial q_l})^2 * (\delta q_l)^2} \tag{5.25}$$

where q_l is the lth component of the aircraft's state vector $X_{air} = (x_{air}, y_{air}, \theta_{air}, v_{g_{(air)}}, z_{air}, v_{h(air)})$. Therefore,

$$\begin{aligned} \delta(\mathbf{x}^*_{uav}) &= (\delta x^*_{uav}, \delta y^*_{uav}, \delta z^*_{uav}) \\ &= (v_{x(uav)}, v_{y_{(uav)}}, v_{z(uav)}) * \delta(\Delta t^*) \end{aligned} \tag{5.26}$$

5.2.2 The Path for Collision Avoidance

We start by solving the velocity $\mathbf{v}(t)$ in (5.1). We initially assume a fixed t_f, therefore it becomes an Euler-Lagrange Problem Goldstein (1980). It is equivalent to solve the equations

$$\begin{aligned} \frac{\partial L}{\partial v_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{v}_i} &= 0, \quad i = 1, 2, 3 \\ where \quad L(v_i, \dot{v}_i, t) &= \left(\|\mathbf{v}\| + \lambda \| \frac{d\mathbf{v}}{dt} \|^2 \right) \\ \mathbf{v} &= (v_1, v_2, v_3) \end{aligned} \tag{5.27}$$

substitute L into equation 5.27, and combining all three equations, we have

$$\frac{\mathbf{v}}{\|\mathbf{v}\|} + 2\lambda \ddot{\mathbf{v}} = 0 \tag{5.28}$$

therefore,

$$\begin{aligned} \frac{d}{dt}(\mathbf{v} \times \dot{\mathbf{v}}) &= \dot{\mathbf{v}} \times \dot{\mathbf{v}} + \mathbf{v} \times \frac{d}{dt}\dot{\mathbf{v}} \\ &= \mathbf{v} \times \frac{d}{dt}\dot{\mathbf{v}} \\ &= -\mathbf{v} \times \frac{1}{2\lambda} \frac{\mathbf{v}}{\|\mathbf{v}\|} \\ &= 0 \end{aligned} \tag{5.29}$$

It indicates that $\mathbf{v} \times \dot{\mathbf{v}} = const$, that is, the velocity of the UAV always lies on a plane. Therefore, the UAV's motion is restricted on a plane when performing collision avoidance. We can use (5.31) as the UAV's kinematic model on this plane. For simplicity, we further assume a constant $\|\mathbf{v}\|$. Using such model and following Dubins (1957), if we prescribe the initial and terminal position and heading of the UAV in avoidance procedure, the shortest path of the UAV would be a Dubins curve (Dubins (1957)). It is a curve composed of different combination of left turning circle, right turning circle and straight line. The turning rate adopted is either maximum value or zero. The Dubins curve ensure the minimum length Dubins (1957) and we need to observe the experiment results to see if it gives minimum curve smoothness (the $\|\frac{d\mathbf{v}}{dt}\|^2$ term).

dis1	0.5ρ	1.0ρ	1.5ρ	2.0ρ
dis2(km)	0.52	1.13	1.43	1.44

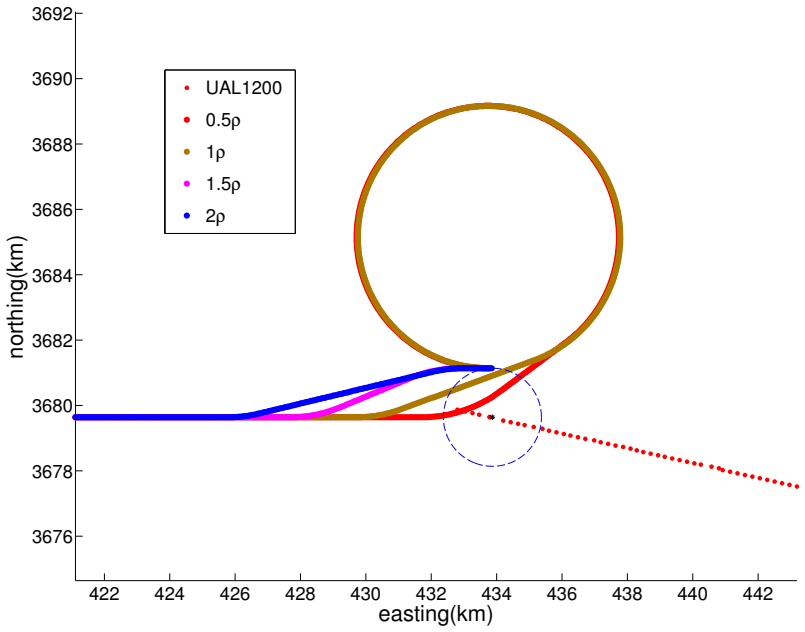
Table 5.1: The Closest Distance Between the UAV and the Aircraft for the Four Avoiding Curves.

Figure 5.2 shows Dubins curves generated with the same initial heading, terminal position and heading but with different initial position [1] . Both initial and terminal headings are the same as the UAV's original heading in route. The terminal position is determined so that the Dubins curve will not pass through the uncertainty circle surrounding the collision point at any time. The radius of the uncertainty circle is determined by the safe separation threshold L in equations 5.1 and the collision point's uncertainty calculated in equation 5.26 as:

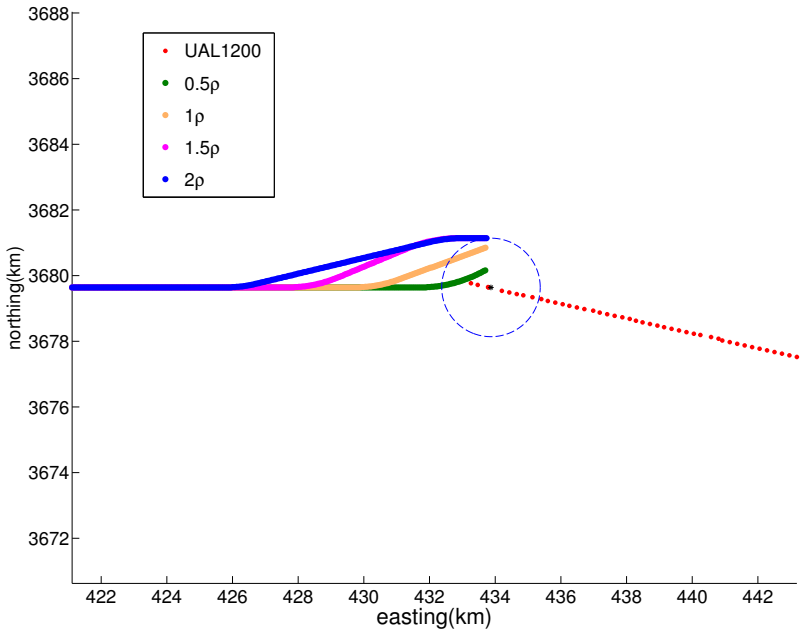
$$r_{uncertain} = \begin{cases} \delta_{max} + L & \text{if } \delta_{max} \leqslant r_u \\ r_u + L & \text{if } \delta_{max} > r_u \end{cases} \tag{5.30}$$

where $\delta_{max} = max(\delta x^*_{uav}, \delta y^*_{uav}, \delta z^*_{uav})$. r_u is a preset threshold to prevent exceedingly large aircraft position uncertainty accumulated during ADS-B dropouts. The Dubins curve is concatenated to a line segment tangent to the uncertainty circle. By combining both of them we get the path the UAV should follow to avoid the obstacle. Figure 5.2 compares Dubins curves with four different start positions. We observe that if it starts sufficiently close to collision point, the UAV will overshoot the terminal point and has to make almost a complete circle to return to it (curve 1ρ,0.5ρ). The UAV does not need to complete the avoiding curve and it can stop when it pass the obstacle. Figure 5.2b shows the avoiding path until

[1] The speed and the turning rate of the UAV are half scaled from Global Hawk's specification in http://www.nasa.gov/centers/dryden/aircraft/GlobalHawk/
performance.html. Dubins curves are calculated based on Shkel and Lumelsky (2001), using the implementation from https://github.com/AndrewWalker/Dubins-curves



(a)



(b)

Figure 5.2: Dubins curves for collision avoidance. The curves start at different distance to the collision point, represented by the black star. The blue dashed circle indicates the uncertainty of the collision point. The start points for the curves are 0.5ρ, 1.0ρ, 1.5ρ, 2.0ρ to the collision point, where ρ is the UAV's turning radius.

the UAV passes the moving obstacle. The metric to evaluate the avoiding efficacy is the closest distance between the UAV and the aircraft. We list the closest distance between the two for the four curves in Table 5.1. We use 0.5km as the safe separation threshold. The table indicates that the UAV and the aircraft are separated more than the threshold in the avoiding procedure for all four curves. For safety, we choose 1ρ to the collision point as the avoidance start position for further testing.

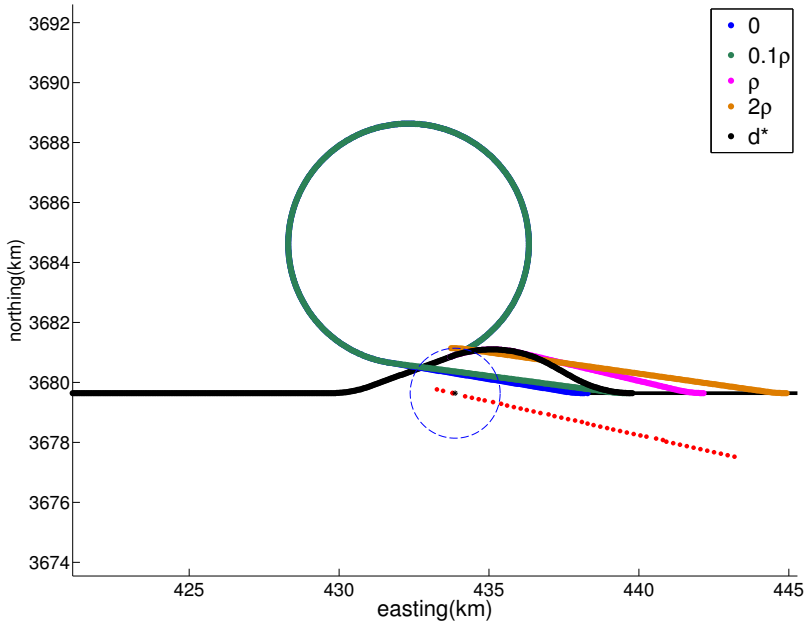


Figure 5.3: The Dubins curves for return with different terminal positions along the route. The terminal points for the curves are 0, 0.1ρ, 0.5ρ, 1.0ρ to a common fixed point, where ρ is the UAV's turning radius.

The Path for Return to Route

Once the UAV finishes the avoidance procedure (flying by the obstacle) it needs to return to the original route to resume its mission. The return path also needs to be the shortest length. That inspires us to use another Dubins curve to achieve this. The initial position and heading are determined at the moment the UAV flies by the aircraft. The terminal heading is the same as that of original route and the terminal position should lie in the route. We plot Dubins curves with four different terminal positions along the route in Figure 5.3. We observe that when the terminal point is close to collision point, the UAV has to follow a complete circle to reach the terminal point due to kinematic constraints (curve $0, 0.1\rho$). But when terminal point is sufficiently far away, the Dubins curve becomes "unfolded" and the curve's length decreases and if the terminal point moves further away, the length will increase. We choose the terminal

point achieving the minimal curve length as the desired terminal point via an incremental search. The Dubins curve calculated using that terminal point is the return path (curve d^* in Figure 5.3). The black curve in Figure 5.3 demonstrates the calculated optimal path of the UAV from collision avoidance to return-to-route. The total avoidance path is the combination of avoiding and return curve. Since both of the curves achieve the shortest length, we automatically have an shortest t_f in equations (5.1) due to constant speed.

5.3 Experiments

To validate the global planner for moving obstacle avoidance, a collision is simulated between the UAV's flight path and an aircraft's path estimated from ADS-B data. We selected six aircraft with less ADS-B dropouts for avoidance tests. ADS-B data of aircraft appearing in the air at the same time as the test aircraft were kept but we did not set the UAV to collide with them. However, the UAV still needs to determine if a collision will happen with each of them. From the results, the UAV did not try to avoid any of them and thus proved that our collision estimation approach did not produce false alarm.

The kinematic model of the UAV is:

$$\begin{aligned} v_x &= V_0 cos(\theta) cos(\gamma) \\ v_y &= V_0 sin(\theta) cos(\gamma) \\ v_z &= V_0 sin(\gamma) \\ \dot{V}_0 &= 0 \\ -c_1 &\leqslant \dot{\theta} \leqslant c_1 \\ -c_2 &\leqslant v_z \leqslant c_2 \\ -c_3 &\leqslant \dot{v}_z \leqslant c_3 \end{aligned} \tag{5.31}$$

where (v_x, v_y, v_z) are velocity components of the UAV. θ is heading angle and γ denotes the climbing angle. Specifically, we set $V_0 = 150knot = 0.077km/s, c_1 = 1.1deg/s, c_2 = 10000ft/min = 0.0508km/s, c_3 = 0.2g$

We tested collision avoidance in two flight modes of the UAV: constant altitude flight ($\gamma = 0$) and climbing ($\gamma = const \;\&\&\; 0 < \gamma \leqslant arcsin(c_3/V_0)$). specified In constant altitude flight, we can ignore the z dimension and visualize the Dubins curve in a 2D. In this case, kinematic models in equations 5.31 and 5.31 are the same. Figure 5.4 demonstrates the Dubins curve for collision avoidance with flight UAL1200. The UAV approaches from 12 directions. We choose the directions incrementally with 30°

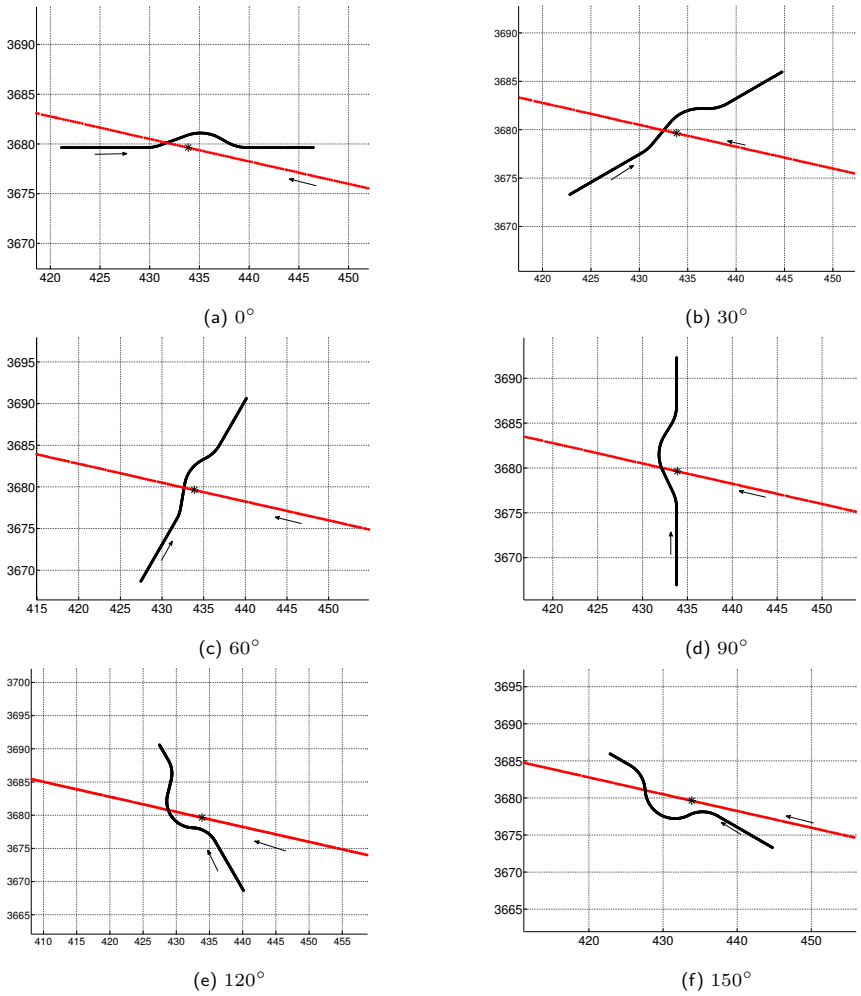


Figure 5.4: Dubins curve (black) for the UAV to avoid collision with flight UAL1200 (red). The black dot on the red line indicates the collision point. In the figures, the UAV and aircraft path intersects because they both travel through the same point in space but in different time. That does not mean the two meet at that point. The sub-captions mean the angles between the UAV's original route and the west (minus x axle). The arrows show their traveling directions. The horizontal axle is easting(km) and vertical is northing(km).

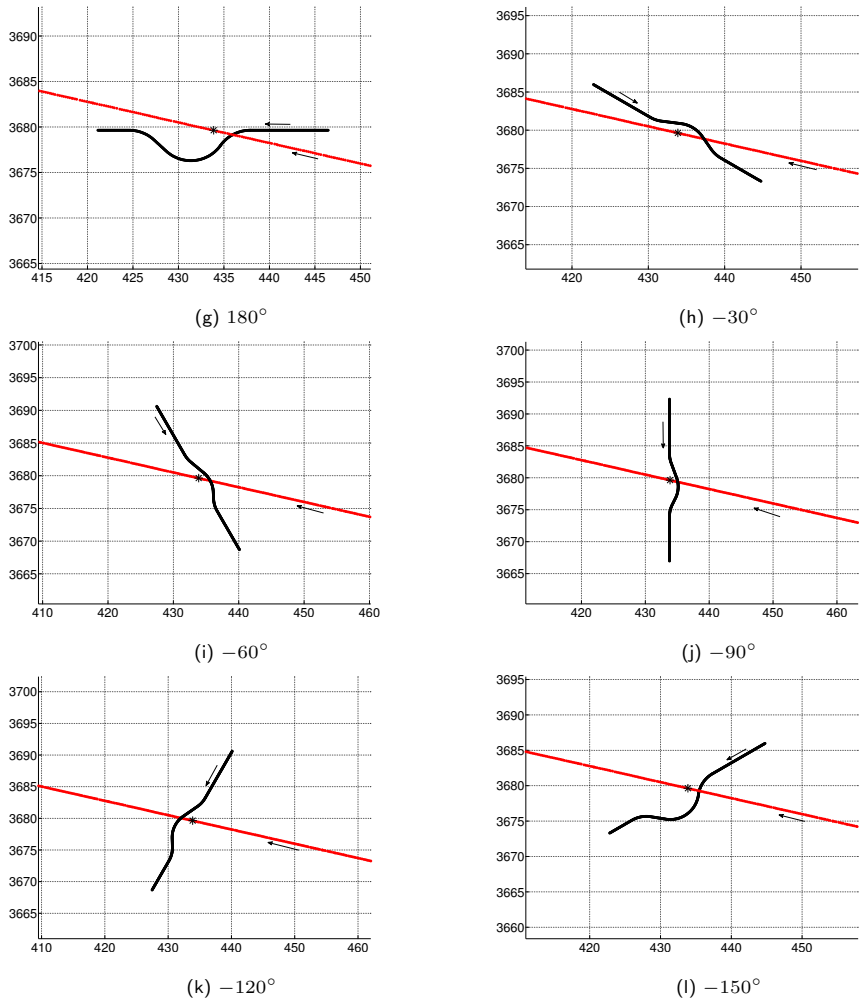


Figure 5.4: Dubins curve (black) for the UAV to avoid collision with flight UAL1200 (red). The black dot on the red line indicates the collision point. In the figures, the UAV and aircraft path intersects because they both travel through the same point in space but in different time. That does not mean the two meet at that point. The sub-captions mean the angles between the UAV's original route and the west (minus x axle). The arrows show their traveling directions. The horizontal axle is easting(km) and vertical is northing(km).

as the step. We measure the efficacy of avoidance with the closest distance between the UAV and the aircraft in the whole meet-and-separate procedure, as listed in Table 5.2.

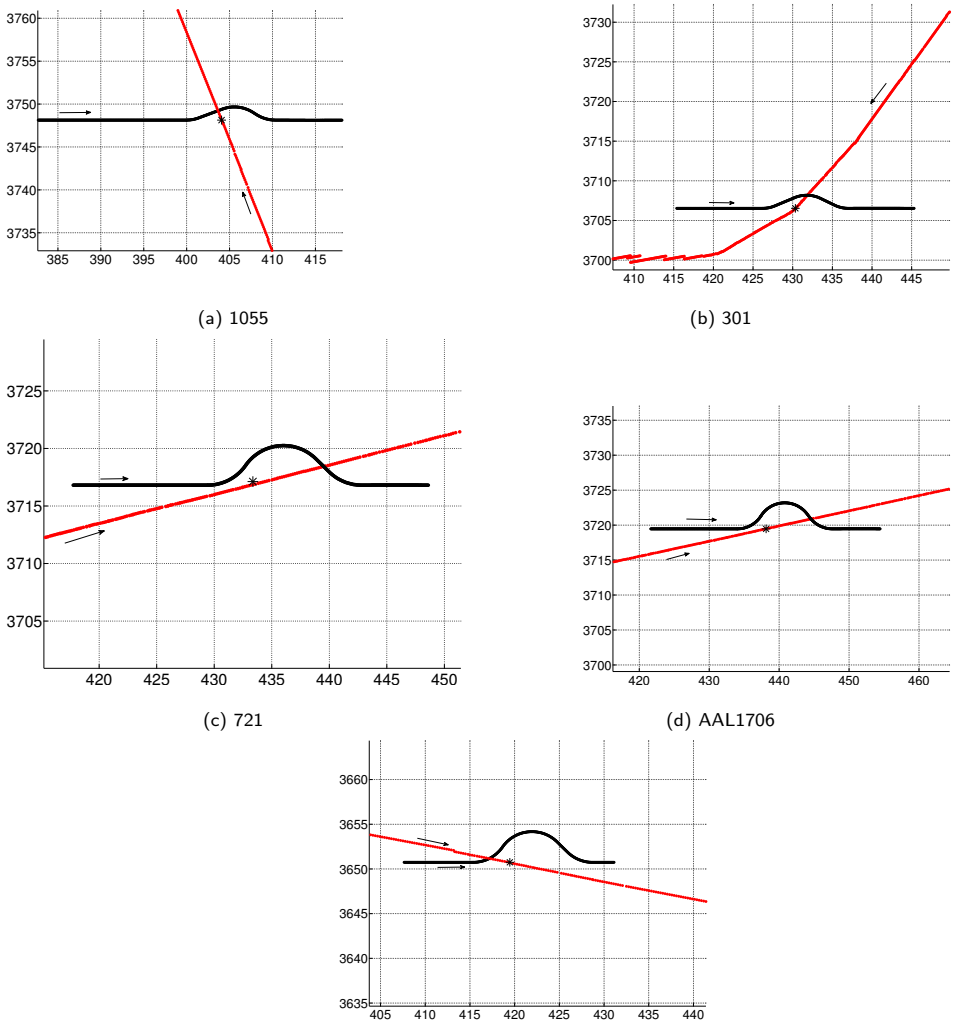
direction	0°	30°	60°	90°	120°	150°
close_d (km)	1.130	0.882	0.424	0.279	1.990	1.95686
direction	180°	-30°	-60°	-90°	-120°	-150°
close_d (km)	1.410	1.290	1.028	0.653	0.39812	0.504415

Table 5.2: The Closest Distance Between the UAV and the Aircraft for the Twelve Approaching Directions.

The closest distances below $L = 0.5$ are highlighted with red in the table. They corresponds to Figure 5.4c,5.4d,5.4k, where the directions in which the UAV deviates from its original route are almost parallel to the aircraft's traveling direction. In another word, the aircraft is "chasing" the UAV when the UAV tries to avoid it. Such drawback can be compensated by considering velocity of the aircraft when estimating the uncertainty radius of the collision point in equation (5.30). It would be part of future work. It can also be compensated with a local path planner which is able to perform a second level avoidance. We next investigate the optimality of the generated path in terms of path length and smoothness, the two terms of cost function in equations (5.1). We investigate by comparing the Dubins curve with another simple avoidance strategy, where the UAV is always commanded to head tangent to the uncertainty circle of collision point, which is updated at ADS-B transmission frequency. Both methods use Dubins curve to return to route. We presented the path length and total smoothness of the path from deviation start to return to the route for both methods in Table 5.3. We only listed directions in which the tangent method also achieves successful separation (the closest distance $> L$). We observe that in most cases the Dubins curves do cost shorter path length and less smoothness.

We also tested the global path planner in collision with other obstacles (other aircraft). We presented results when the UAV approaches the aircraft from an identical direction (0°). Figure 5.5 demonstrates the Dubins curve avoidance path to avoid different aircraft. Table 5.4 shows the the closest distance between the UAV and the aircraft. They are all above L, thus prove the efficacy of Dubins avoidance curve.

In the climbing mode, the pitch angle γ is determined by the preset collision point on the aircraft's estimated path. The generated Dubins curve lies on the plane expanded by the roll and pitch axis of the UAV just the moment the UAV starts the avoidance path. A transformation is made between kinematic models in equations 5.31 and 5.31. We tested collision avoidance when climbing, varying θ in equation (5.31) of the original route. The generated Dubins curves to avoid flight UAL1200 are



(a) 1055

(b) 301

(c) 721

(d) AAL1706

(e) VRD336

Figure 5.5: Dubins curve (black) for the UAV to avoid collision with different flights (red). The black dot on the red line indicates the collision point. The UAV approaches from the west. The arrows show their traveling directions.

	Dubins		Tangent	
direct	length	smoothness	length	smoothness
0°	10.659	0.100	12.676	0.157
30°	10.823	0.102	14.462	0.150
60°	10.889	0.103	13.310	0.159
150°	16.191	0.221	14.0587	0.181
180°	15.384	0.195	12.330	0.139
-30°	11.293	0.097	12.445	0.144
-60°	10.371	0.096	13.828	0.167
-90°	10.256	0.094	14.289	0.178
-120°	12.215	0.114	15.787	0.208
-150°	15.038	0.185	15.614	0.186

Table 5.3: Comparison of Length and Smoothness of the Avoidance Path Between Dubins Curve and Tangent Avoiding Approach.

flight	1055	301	721	AAL1706	VRD336
close_d (km)	0.565	1.195	1.579	1.755	1.399

Table 5.4: The Closest Distance Between the UAV and the Aircraft for the Other Five Aircraft During Collision Avoidance with a Constant Altitude.

demonstrated in Figure 5.6. Table 5.5 shows the minimal distance between the UAV and the aircraft during the avoidance procedure. Similar to Table 5.2, the UAV fails to make a separation above $L = 0.5$ when the UAV's deviation from the route has a projected component parallel to the aircraft's approaching direction.

direction	0°	30°	60°	90°	120°	150°
close_d (km)	1.250	0.912	0.360	0.327	1.842	1.925
direction	180°	-30°	-60°	-90°	-120°	-150°
close_d (km)	1.365	1.187	1.089	0.806	0.508	0.454

Table 5.5: The Closest Distance Between the UAV and the Flight UAL1200 for the Twelve Approaching Directions.

Collision avoidance with other five aircraft are shown in Figure 5.7, where the UAV heads towards the east while climbing. Table 5.6 represents the UAV's minimal distance to the aircraft through the avoidance procedure. None of them falls below L, which proves the efficacy of Dubins avoidance curve when the UAV is climbing.

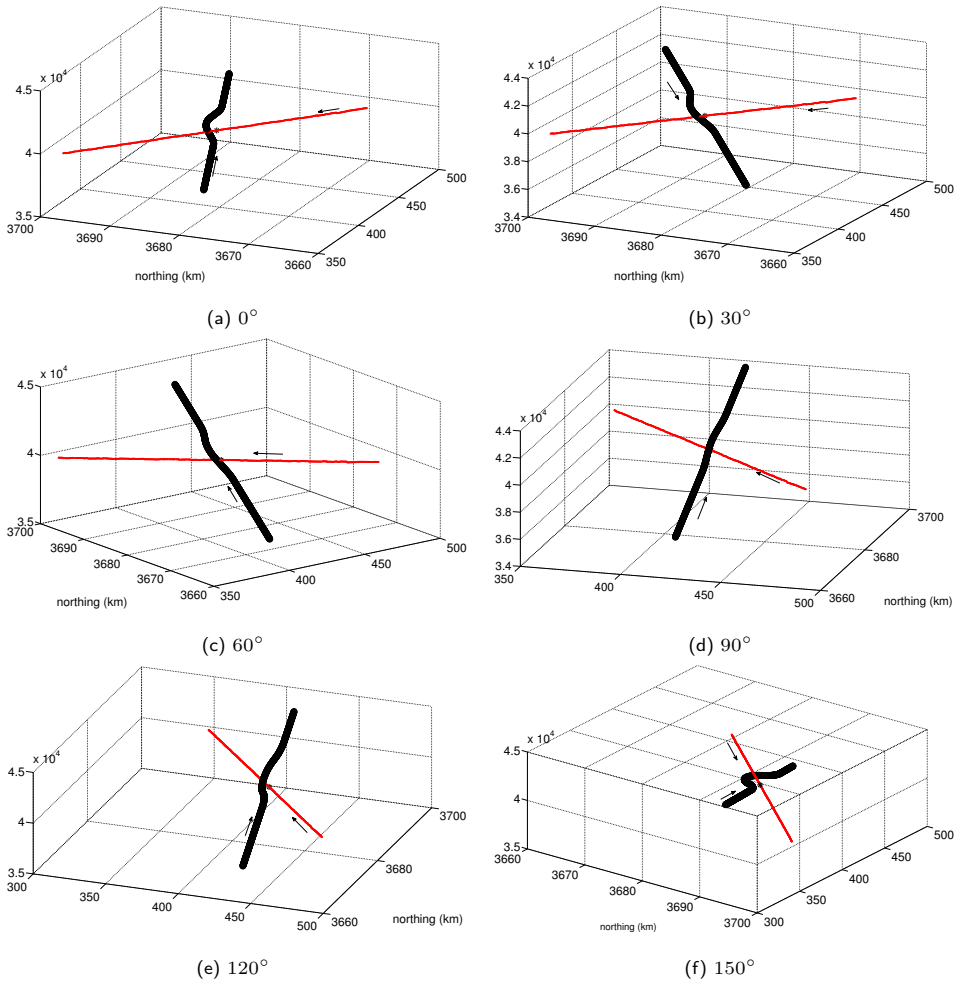


Figure 5.6: Dubins curve (black) for the UAV to avoid collision with flight UAL1200 (red curve) when it is climbing. The black dot on the red curve indicates the collision point. The sub-captions mean the angles between the UAV's original route's heading and the west (minus x axle). The arrows show their traveling directions. The horizontal axles are northing(km) and easting (km). The vertical axle is altitude(ft).

flight	1055	301	721	AAL1706	VRD336
close_d (km)	0.508	1.211	1.521	1.762	1.397

Table 5.6: The Closest Distance Between the UAV and the Aircraft for the Other Five Aircraft During Collision Avoidance in Climb.

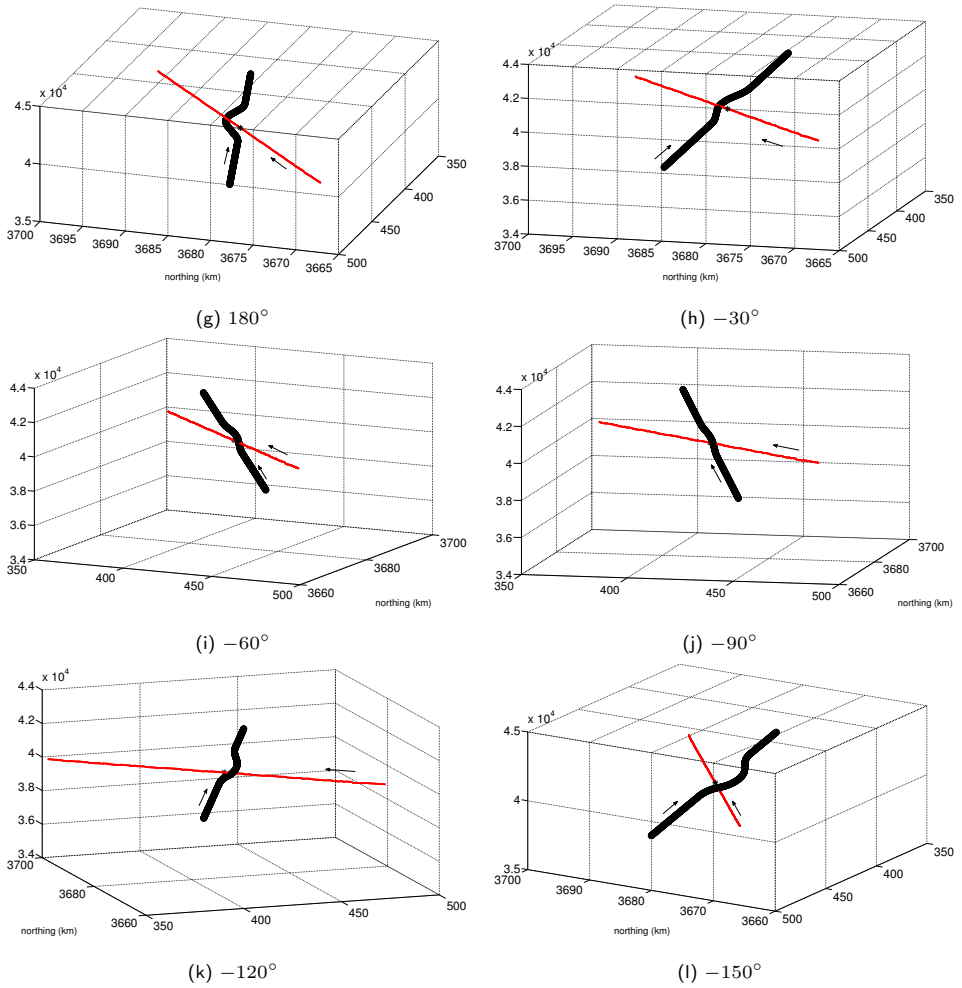
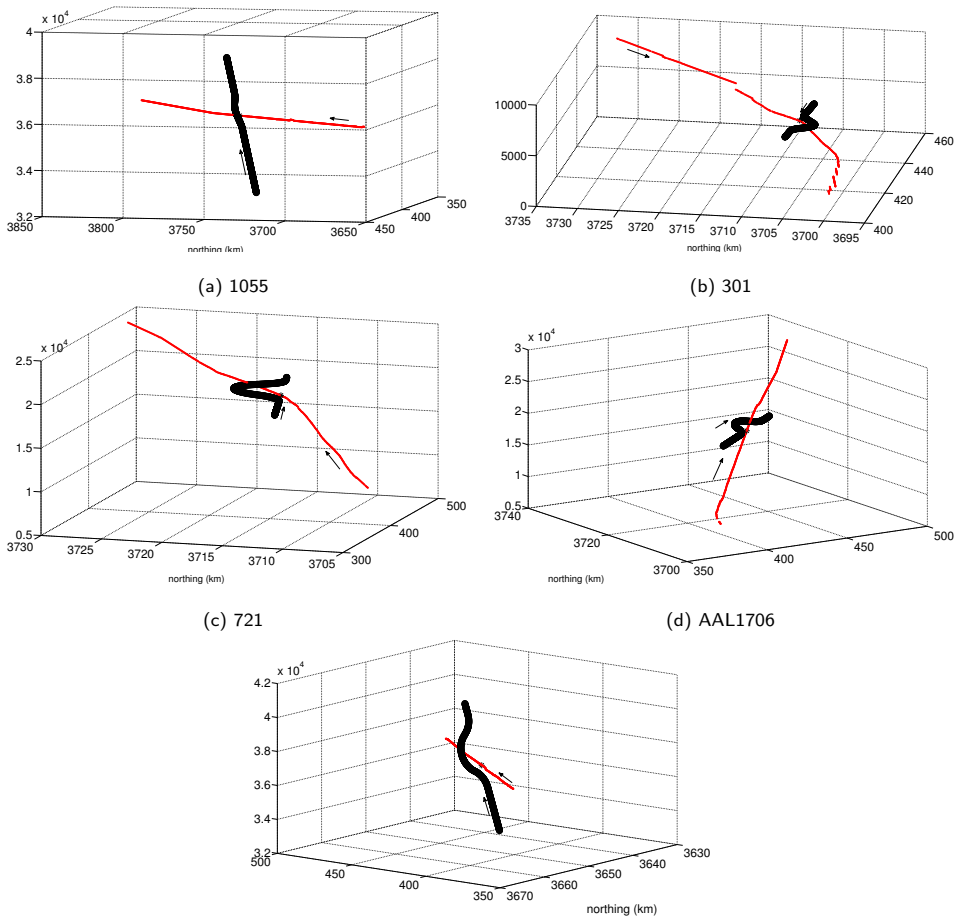


Figure 5.6: Dubins curve (black) for the UAV to avoid collision with flight UAL1200 (red curve) when it is climbing. The black dot on the red curve indicates the collision point. The sub-captions mean the angles between the UAV's original route's heading and the west (minus x axle). The arrows show their traveling directions. The horizontal axles are northing(km) and easting (km). The vertical axle is altitude(ft).



(a) 1055

(b) 301

(c) 721

(d) AAL1706

(e) VRD336

Figure 5.7: Dubins curve (black) for the UAV to avoid collision with different flights (red) when they are climbing. The black dot on the red line indicates the collision point. The UAV heads towards the east while climbing. The arrows show their traveling directions.

Chapter 6

CLOSED-LOOP RRT FOR MOVING OBSTACLE AVOIDANCE

6.1 Overview

Motion planning involves getting a robot to automatically move while avoiding collisions with obstacles. It seeks to find a solution to the problem of "Go from the start to the goal while respecting all of the robot's constraints." The classical motion planning problem known as the piano mover's problem is found to be PSPACE-hard (Reif (1979)).

Two main schools of thought have been developed for motion planning (LaValle (2006)): 1) *combinatorial planning,* which constructs structures in the configuration space (C-space) that discretely and completely capture all information needed to perform planning. It is also named the exact method. 2) *sampling-based planning,* which uses collision detection algorithms to probe and incrementally search the C-space for a solution. It does not completely characterize all of the obstacle-free C-space's structure. In a number of practical problems, characterizing obstacle free C-space is difficult. Therefore, Sampling-based methods are preferred in such cases.

There are two main types of sampling-based methods: graph based and tree based planners. Graph based methods utilize random sampling in the state space to build a roadmap of the free state space. Graph search methods can be used to create paths from the roadmap. It is a multi-query approach because it is able to find connections between multiple initial-goal query pairs. Instead of building a roadmap up front, tree based methods incrementally construct the graph. Each type of tree based methods has a vertex selection method, which determines where to expand next from among vertices in the graph. After that, a local planning method constructs an edge from the selected vertex, thereby extending the tree. Tree-based methods are suitable for single-query planning. They are also able to deal with differential constraints by encoding control information for each edge of the tree. In our application, the UAV has differential constraints and therefore a tree-based planner is adopted.

Specifically, Closed-Loop Rapidly Exploring Random Tree (CL-RRT) (Kuwata *et al.* (2008)) is selected from tree-based planners. Compared to traditional RRT than samples in the robot's configuration space, CL-RRT samples in the space of inputs to the UAV's closed-loop system. In section 6.2.2, the advantages of CL-RRT over traditional RRT will be discussed.

6.2 Algorithm Description

In the last chapter, the Dubins curve based path planner is developed. However, only simplified model of the UAV is used. Instead, CL-RRT is able to accommodate the actual UAV's model.

6.2.1 Problem Formulation

This section presents a generalized formulation of the path planning problem for UAV moving obstacle avoidance. It extends the problem defined in equation 5.1. The path is represented by a function

$$f(x, y, z) = 0 \tag{6.1}$$

It must go through the given start and goal. Therefore $f(x_s, y_s, z_s) = 0$ and $f(x_g, y_g, z_g) = 0$, where (x_s, y_s, z_s) is the start and (x_g, y_g, z_g) the goal.

The obstacle should not enter the UAV's safety zone:

$$\text{for } i = 1, 2, ..N_o, \forall t \in [0, t_f], \;\; \mathbf{x}_{O_i}(t) \notin O(\mathbf{x}(t)) \tag{6.2}$$

where $O(\mathbf{x}(t))$ stands for the safety region of the UAV at t. It can be a sphere or a cylinder. $\mathbf{x}_{O_i}(t)$ is the estimated position of aircraft O_i at t. It is extrapolated from O_i's current position and velocity:

$$\mathbf{x}_{O_i}(t) = \mathbf{x}_{O_i}(t_0) + \mathbf{v}_{O_i}(t_0) * (t - t_0) \tag{6.3}$$

It is noted that the prediction can be done in more elaborated way using a Kalman filter (Watanabe *et al.* (2006)) but it is required that the dynamic characteristics is known at least partially in advance.

The UAV must also satisfy other space constraints such as the height limit and geo-fences:

$$S(\mathbf{x}(t)) < 0 \tag{6.4}$$

The UAV's model is given by

$$\dot{\mathbf{y}(t)} = \mathbf{f}(\mathbf{y}(t), \mathbf{u}(t)), \;\; \mathbf{u}(t) \in U. \tag{6.5}$$

where $\mathbf{y} \in \mathbb{R}^{n_y}$ is the state of the system. $\mathbf{u} \in \mathbb{R}^{n_u}$ is the control input to it. U is the limit of control. The minimum path length requirement is fulfilled as

$$f^* \;=\; \underset{f}{\operatorname{argmin}} \int_f d\mathbf{s} \tag{6.6}$$

In summary, the problem can be defined as: Given the initial state $\mathbf{y}(0) = \mathbf{y}_0$ and the control limit U, generate an optimal path $f^*(x, y, z) = 0$ defined by (6.6) while satisfying (**??**) to (6.5).

6.2.2 Motion Prediction Using the Closed-Loop System

CL-RRT extends the RRT by making use of a low-level controller. It samples an input to the closed-loop system consisting of the vehicle model and the controller Kuwata *et al.* (2008).

Fig. 6.1 (from Kuwata *et al.* (2009)) shows the closed-loop system for motion prediction. The low-level controller takes a reference command $\mathbf{r} \in \mathbb{R}^{n_r}$. The reference command typically has a much lower dimension than vehicle states (i.e.,$n_r \ll n_x$). For example, in our application, the reference command is a 3D waypoint to guide the UAV (three parameters). The UAV's model included six states

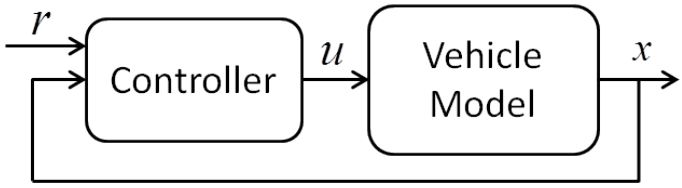


Figure 6.1: Closed-loop motion prediction. Given an input command $\mathbf{r}$, the controller generates vehicle commands $\mathbf{u}$. An updated prediction of the vehicle's state $\mathbf{x}$ is obtained with $\mathbf{u}$ and the vehicle model.

(nine parameters), which are the 3D coordinates, speed, yaw, and pitch. Given a waypoint command and current vehicle state $\mathbf{x}(t_i)$, CL-RRT runs a forward simulation using the controller and the vehicle model to predict the vehicle state $\mathbf{x}(t_{i+1})$. Each predicted state is checked against the constraints. The prediction repeats until the waypoint is reached by the simulated trajectory. Thus we obtain a predicted trajectory $\mathcal{X}(t)$.

The close-loop approach has several advantages over the basic version of RRT (LaValle and Kuffner (2001)). First, the closed-loop prediction results in smaller prediction errors. This property is desirable for checking predicted trajectory's collision with moving obstacles, where we need to estimate the distance between predicted position of the UAV and obstacles at each time instance. Second, the forward simulation is able to handle any nonlinear controller and vehicle model, and the predicted trajectory $\mathcal{X}(t)$ satisfies the vehicle model (6.5) by construction. Finally, a single input to the closed-loop system can create a full trajectory. This requires fewer samples to build a tree and therefore improves the efficiency of sampling-based planning.

6.2.3 Tree Expansion

The tree expansion for CL-RRT is summarized in algorithm 1. In each loop of adding a new node, a sample is first obtained (line 2. Sample strategies in different applications are described in chapter 8.

Algorithm 1 TreeExpand()

1: **for** Δt **do**
2: Generate a sample $\mathbf{z}_s$.
3: Sort the nodes in the tree by heuristics in ascending order .
4: **for** each node q in the tree,in the sorted order **do**
5: Simulate a trajectory $\mathcal{X}(t), t \in [t_1, t_2]$ from q to $\mathbf{z}_s$ using the closed-loop system and check the trajectory against constraints in section 6.2.1.
6: **if** Satisfy the constraints **then**
7: Add the end of $\mathcal{X}(t)$ to the tree with q as their parent. Break.
8: **end if**
9: **end for**
10: **for** each newly added node q_n **do**
11: Calculate the cost of q_n by adding length of the trajectory from its parent and to the cost of its parent.
12: Simulate a trajectory from q_n to the goal using the closed-loop system and check it against constraints.
13: **if** Satisfy the constraints **then**
14: Mark q_n as goal reachable.
15: Calculate cost-to-go of q_n as the length of the trajectory to the goal.
16: **end if**
17: **end for**
18: **end for**

The nodes in the tree are sorted ascendingly by heuristics (line 3). Heuristics are approximate cost from each node to the sample node. For example, the length of a dubins curve can be used as the heuristics for fixed-wing vehicles. Starting from the node q with the smallest heuristics, a trajectory is simulated using the closed-loop system in section 6.2.2 from q to the sample node $\mathbf{z}_s$. The trajectory is checked against the constraints stated in section 6.2.1. If the constraints are satisfied, the node q is added to the tree (line 7). For a newly added node q_n, its cost is calculated as the sum of the trajectory length from its parent q to q_n and the cost of q. Then an attempt is made to reach from q_n to the goal: a trajectory from q_n to the goal is simulated using the closed-loop system and checked against constraints (line 12). If it satisfies the constraints, q_n will be marked as reachable to the goal and its cost-to-go is the length of the simulated trajectory. The tree expansion process is illustrated in Figure 6.2. To generate a path, the goal- reachable node q^* with smallest sum of cost and cost-to-go is selected. The node sequence connecting the start to the goal via q^* is the generated path.

6.2.4 *Execution Loop*

In the complete loop of algorithm execution, a repetitive path replanning strategy is adopted to compensate the inaccuracy of the UAV's closed-loop system and the prediction of obstacles' motion. It is summarized in algorithm 2. In each loop, the algorithm first predicts a collision by simulating the UAV's trajectory up the the time horizon Δt_m. Δt_m is determined by the time needed for the UAV to make a turning maneuver to avoid collision. If the collision location is close to the start, a reactive avoidance action such as climbing up will be taken. The closeness is determined by using equation (16) in Lin and Saripalli (2014). Otherwise, algorithm 1 is used to generate a collision avoidance path. After a path is generated, it is not immediately sent to the UAV. Instead, it is checked against the constraints with updated UAV and obstacles' states. If the constraints are still satisfied, the path is sent to the UAV to execute. Otherwise, TreeExpand() is repeated to obtain a new path. The loop repeats until the goal is reached.

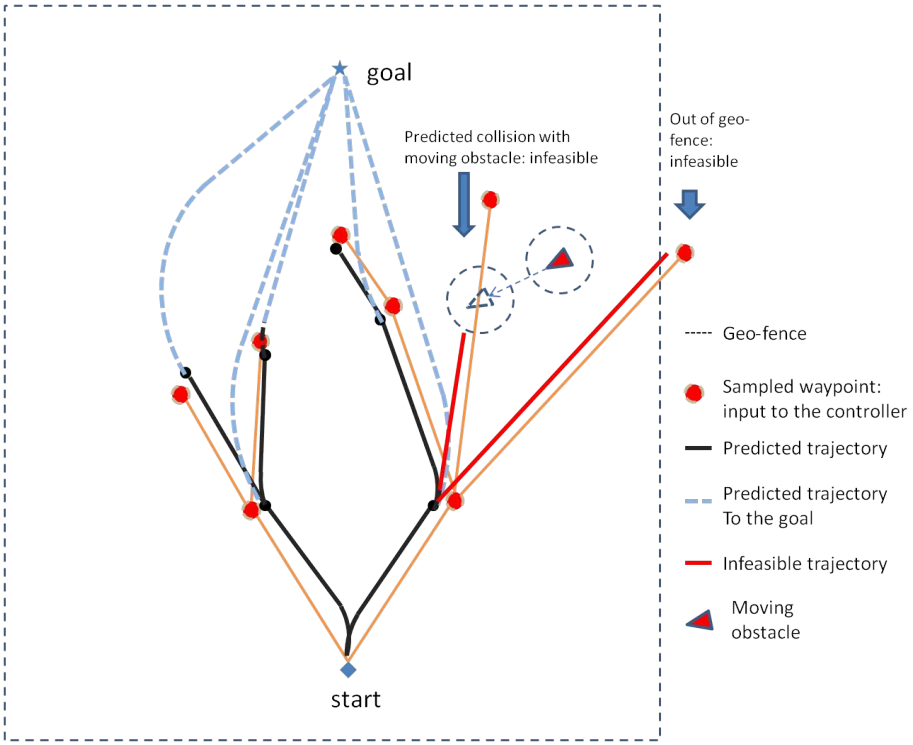


Figure 6.2: Illustration of tree expansion for CL-RRT. UAV trajectories are generated using the closed-loop system. The trajectories are checked against constraints. The black trajectories are feasible. The red ones are infeasible because they will either induce collision with the moving obstacle or reach out of the geo-fence. The blue dashed are the trajectories connecting each node to the goal.

Algorithm 2 Execution Loop()

```
1: repeat
2:     Predict the collision up to a time horizon Δt_m
3:     if A collision is predicted then
4:         if Collision location is close to the start location then
5:             take reactive avoidance action.
6:         else
7:             TreeExpand()
8:             if No path is feasible then
9:                 Go to line 7
10:            end if
11:            Update the UAV and the obstacles' states and simulate a new UAV trajectory starting from the updated
               position to the goal and check against the constraints.
12:            if Constraints violated then
13:                Go to line 7
14:            else
15:                Send the path to UAV.
16:            end if
17:        end if
18:    end if
19: until Finish the waypoint sequence
```

Chapter 7

MODIFICATION AND EXTENSION TO CLOSED-LOOP RRT

7.1 Overview

In this chapter we try to improve the Closed-Loop RRT based method in the previous chapter. First, methods to speed up the planning are proposed. Following them, an approach to avoid obstacles not moving along a straight line is developed.

7.2 Generating More Candidate Paths

7.2.1 A Greedy Version of Closed-Loop RRT

From the experiment results of Closed-Loop RRT based methods (Figure 8.6 and 8.10), it is observed that a generated path contains only one node between the start and the goal. That inspires us to modify the tree expansion algorithm to the version in algorithm 3: The algorithm repeats a cycle of adding a

Algorithm 3 TreeExpand for the Greedy Version of Closed-loop RRT

```
for Δt do
    Generate a sample waypoint wp_s = (x_s, y_s, z_s) that satisfies constraints.
    Predict a trajectory from the start to wp_s using the closed-loop system and check against the constraints in each simulation.
    step
    if Satisfy the constraints then
        Predict a trajectory from wp_s to the goal using closed-loop system and check against the constraints .
        if Satisfy the constraints then
            Add wp_s to the tree as a child of the start node.
        end if
    end if
end for
Select the sample waypoint that results in the shortest total trajectory from the tree.
```

node to the tree in a given amount of time. In a single cycle, a sample waypoint $\mathbf{wp}_s$ is generated. A trajectory connecting the start and $\mathbf{wp}_s$ is simulated by prediction with the closed-loop system. It is checked against the constraints including obstacle avoidance and geo-fence. If it satisfies the constraints, a second trajectory is simulated between $\mathbf{wp}_s$ and the goal and is checked against the constraints. If the constraints are satisfied, the sample waypoint $\mathbf{wp}_s$ is added to the tree with the start node as its parent. Such cycles are repeated for the preset time interval Δt. After that, the sample waypoint that provides the shortest trajectory from the start to the goal is selected as the final generated waypoint from the tree.

The start of the tree, the generated waypoint, and the goal together compose the collision avoidance path.

Compared to algorithm 1, the modified algorithm generates a sample and immediately check the possibility of building a collision-free trajectory that connects the start and the goal through the sample waypoint. It is a greedy approach and avoid the labor to build a complete tree. This will work because we consider avoiding a small number of aircraft. That means obstacles are sparse. According to Choset (2005), such tree expansion approach suffers the risk of getting stuck in a local minima. From our observations in the experiments, the generated path did not get stuck in a local minima. This is also because of the sparsity of obstacles in the environment.

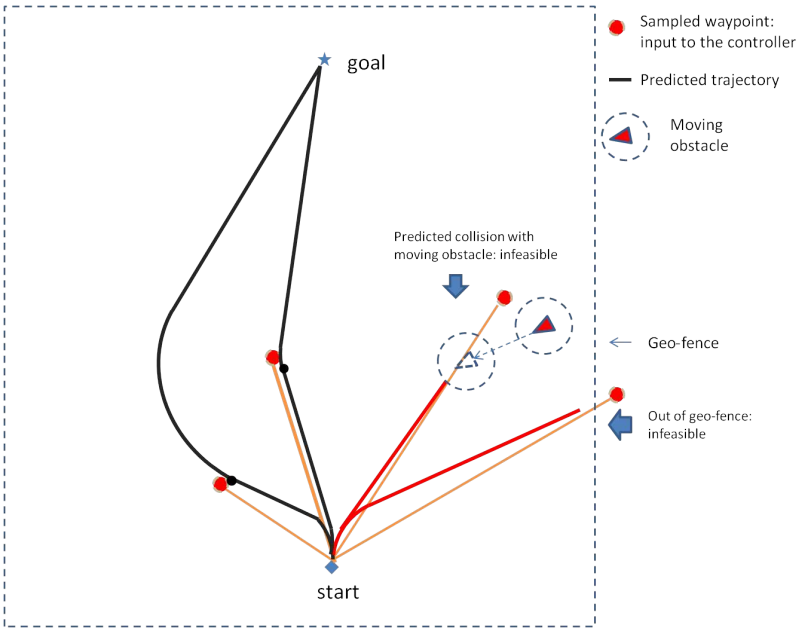


Figure 7.1: Illustration of tree expansion for a greedy version of Closed-Loop RRT. UAV trajectories are generated using the closed-loop prediction. The generated trajectories are then evaluated for feasibility. The black trajectories are feasible. The red trajectories are infeasible because they will either induce collision with moving obstacles or reach out of the geo-fence.

Fig 7.1 shows an example of a tree built by algorithm 3. The red (grey) dots represent sample waypoints. They are used as the reference inputs to the controller to simulate the UAV's trajectories. The black (dark) curves are the feasible predicted trajectories. They start from the start node, go through a sample waypoint, and arrive at the goal. The red trajectories are infeasible. They will induce collision with moving obstacles or reach out of the geo-fence.

Algorithm 4 Path Generation Using Intermediate Points

```
for Δt do
    Generate a sample waypoint wp_s = (x_s, y_s, z_s) that satisfies constraints.
    Predict a trajectory from the start to wp_s using the closed-loop system and check against the constraints in each simulation.
    step
    if Satisfy the constraints then
        Obtain N points along the trajectory from the start to wp_s evenly.
        for i = N to 1 do
            Predict a trajectory from the ith point to the goal using closed-loop system and check stepwise against the constraints.
            if Satisfy the constraints then
                Add the ith point to candidate avoidance waypoints.
            else
                Break
            end if
        end for
    end if
end for
Select the waypoint that results in the shortest total trajectory.
```

7.2.2 Generating Trajectories from Intermediate Points

To obtain a further larger number of candidate trajectories, we made a next level improve to the algorithm in last sub-section. If the trajectory from the start to a sample waypoint satisfy all the constraints, N points will be obtained evenly along the trajectory. Starting from the point closest to $\mathbf{wp}_s$ (the Nth point), another trajectory is simulated using the closed-loop system to connect the point to the goal while checking against constraints stepwise. The point will be added to candidate waypoints for avoidance if the trajectory results in no collision. The for loop breaks when a such trajectory leads to a collision. The updated algorithm is summarized in algorithm 4. Here, the concept of a tree data structure is abandoned in the algorithm because only a vector is needed to store candidate waypoints for collision avoidance.

Figure 7.2 illustrates the path generation procedure. Simulated trajectories going through intermediate points (dark blue dots) are marked by dashed blue curves. Considering trajectories from intermediate points is a major improvement to the algorithm in the last sub-section. This way the space between a sampled waypoint and the straight line connecting the start and the goal will be explored. As a result, a larger number of candidate paths for collision avoidance can be generated with the same amount of waypoint samples. To compare the number of candidate path generated in the same amount of time by closed-loop RRT, greedy closed-loop RRT, and the intermediate points method, we run them with the same start, goal, and obstacle for the same amount of time. The closed-loop system in 8.2.2 is used in all three planners. The start is $(33.4409, -111.996, 1436.7)$ and goal is $(33.441, -112.029, 1436.7)$. The

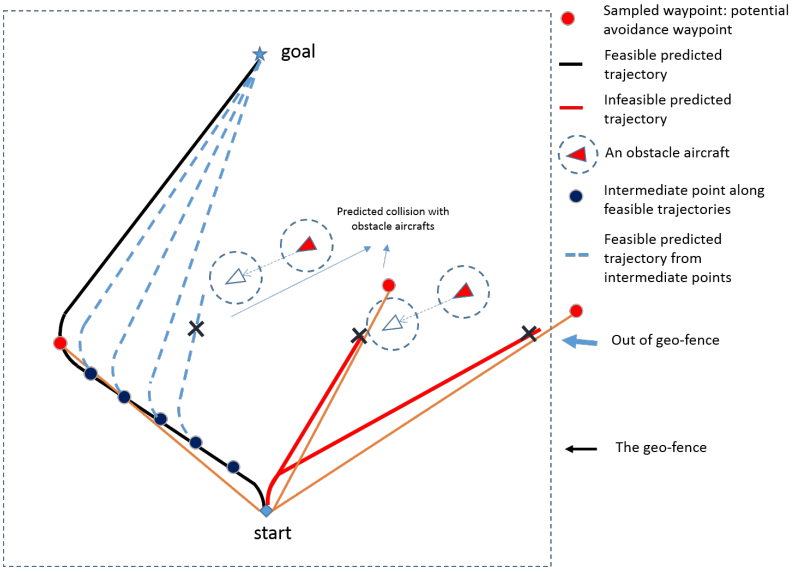


Figure 7.2: Illustration of the path generation using intermediate points. UAV trajectories are generated using closed-loop prediction and waypoint sampling. The generated trajectories are then evaluated for feasibility. The black trajectories are feasible. The red trajectories are infeasible because they will induce collision with obstacle aircraft or reach out of the geo-fence. The blue dashed line are feasible trajectories connecting intermediate waypoints and the goal.

obstacle starts from $(33.441, -112.01, 1439.7)$ and moves towards the start at 15.43 m/s. The initial speed of the UAV is 14.75 m/s.

time(s)	0.1	0.5	1.0
CL-RRT	16	82	161
greedy CL-RRT	33	165	330
intermediate	72	396	759

Table 7.1: The number of candidate paths generated by the Closed-Loop RRT based methods(CL-RRT), the greedy version of Closed-Loop RRT methods (CL-RRT), and the intermediate points method (intermediate) for the same start, goal, obstacle, initial state of the UAV, and the closed-loop system.

Table 7.1 lists the number of candidate paths generated by the the Closed-Loop RRT, the greedy version of Closed-Loop RRT, and the method using intermediate waypoints. It is observed that the number of candidate trajectories from the Closed-Loop RRT based method is increased by two times by the greedy Closed-Loop RRT. It is increased by four times by including trajectories from intermediate points.

7.3 Planning Using the Reachable Set

In chapter 6 and previous sections, linear motion assumption is used to predict the position of a moving obstacle. However, a moving obstacle may not always move along a straight line. Therefore in this section, reachable sets are used to represent the motion uncertainty of moving obstacles. Collision check for the UAV is performed against reachable sets induced by moving obstacles.

7.3.1 Calculation of the Reachable Set

The reachable set defines the region that a moving obstacle can reach at each time instance. All motions of the obstacle permitted by kinematic constraints are considered when the reachable set is calculated. The $x-y$ and z components of the reachable set are calculated separately. The $x-y$ component RS_{xy} is a bounded area and the z component is a 1D interval $[h_{lower}, h_{upper}]$. A point (x, y, z) falls in the obstacle's reachable set if and only if

$$(x, y) \in RS_{xy} \tag{7.1}$$

$$\text{and } h_{lower} \leqslant z \leqslant h_{high} \tag{7.2}$$

Constraints 7.1 and 7.2 are used for collision prediction and checking in path planning. The path planning algorithm is described in the next section.

To calculate the boundary of RS_{xy}, we assume that a moving obstacle moves according to the Dubin's car model:

$$v = const \tag{7.3}$$

$$\dot{\phi} \leqslant \omega_{max} \tag{7.4}$$

Where v is the obstacle's horizontal speed and ϕ is its heading. ω_{max} is the obstacle's maximal turning rate. It corresponds to the minimal turning radius $\rho = \frac{v}{\omega_{max}}$. We further assume that there exists an upper limit ϕ_{max} for the obstacle's turning angle w.r.t its initial heading.

The boundary of RS_{xy} is calculated by initially setting the heading of the moving obstacle to 0 and the position to $(0, 0)$ then apply the rotation and translate transform. The boundary consists of $S_{1,2,3,4}(\theta, t)$ and S_5,as in Fig.7.3.

θ is the direction of a normal unit vector at each point. For normal unit vector $\hat{n} = (n_x, n_y)$ at (x, y), $\theta = atan2(n_y, n_x)$. Given the minimal turning radius ρ, speed v, required safety radius r, and a time instance t, the equations of $S_{1,2,3,4}(\theta, t)$ are derived from Wu and How (2012). $S_{1,2}(\theta, t)$ are

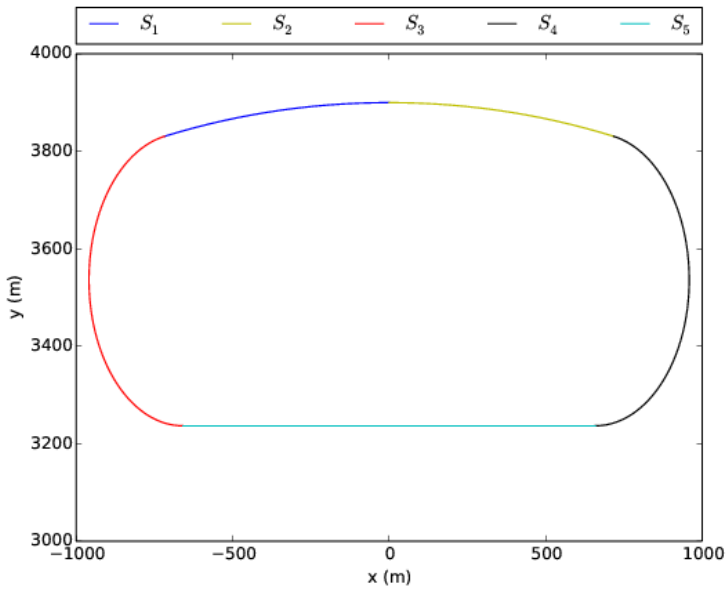


Figure 7.3: The boundary of x-y component of the reachable set (RS_{xy}) defined by Equations 7.5, 7.6, 7.9, 7.10. Here $\rho = 2291.83m$, $t = 30s$, $v = 30m/s$, $r = 300m$

.

parametrically represented as

$$S_1(\theta, t) = \begin{bmatrix} -\rho(1 - cos\theta) + (vt + \rho\theta + r)sin\theta \\ -\rho sin\theta + (vt + \rho\theta + r)cos\theta \end{bmatrix} \tag{7.5}$$

$$S_2(\theta, t) = \begin{bmatrix} \rho(1 - cos\theta) + (vt - \rho\theta + r)sin\theta \\ \rho sin\theta + (vt - \rho\theta + r)cos\theta \end{bmatrix} \tag{7.6}$$

with respective domains

$$S_1 : \ max(-\phi_{max}, -\pi) \leqslant \theta \leqslant 0 \tag{7.7}$$

$$S_2 : \ 0 \leqslant \theta \leqslant min(\phi_{max}, \pi) \tag{7.8}$$

$S_{3,4}(\theta, t)$ are represented as

$$S_3(\theta, t) = S_1(-\phi_{max}, t) + \begin{bmatrix} rsin\theta \\ rcos\theta \end{bmatrix} \tag{7.9}$$

$$S_4(\theta, t) = S_2(\phi_{max}, t) + \begin{bmatrix} rsin\theta \\ rcos\theta \end{bmatrix} \tag{7.10}$$

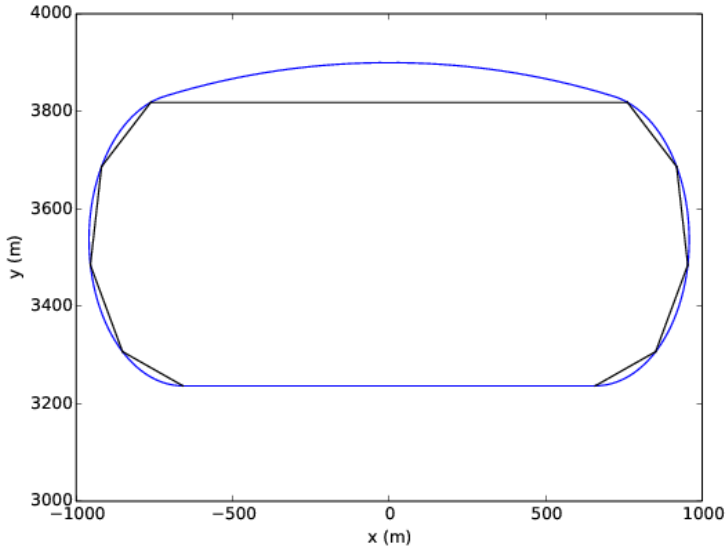


Figure 7.4: The polygon (black) of 10 vertex to approximate the boundary (blue). The same boundary as in Fig. 7.3 is used.

with respective domains

$$S_3 : \ -\pi \leqslant \theta \leqslant -\phi_{max} \tag{7.11}$$

$$S_4 : \ \phi_{max} \leqslant \theta \leqslant \pi \tag{7.12}$$

If $\phi_{max} > \pi$, $S_{3,4}$ are no longer part of RS_{xy}'s boundary. $S_5(t)$ is the horizontal line segment connecting the two lowest ends of $S_{1,2,3,4}$. They are either $S_3(-\pi, t)$ and $S_4(\pi, t)$,or $S_1(-\pi, t)$ and $S_2(\pi, t)$, depending on which pair of curves is defined for $\theta = \pm\pi$ at time t.

Transcendental equations need to be solved if equations 7.5,7.6,7.9,7.10 are used to check if a 2D point falls in RS_{xy}. Therefore, we attempt to approximate the boundary with a polygon. Then the point-in-polygon (PIP) problem Sutherland *et al.* (1974) can be used to check if a point is inside RS_{xy}. The vertex of the polygon are $S_{1,2,3,4}(\theta_i, t)$, where $\theta_i = i\frac{2\pi}{N}, \ i = 0, 1, ..., N-1$. Fig. 7.4 shows the polygon for $N = 10$. The ratios between the area of the polygon and the full RS_{xy} for different N values are compared in Fig. 7.5. It is observed the ratio is above 0.95 when $N = 35$. Therefore, a polygon of 35 vertex is used to approximate the boundary when checking if a point falls in RS_{xy}. The vertex need to be rotated and translated based on the obstacle's heading and position for an actual moving obstacle.

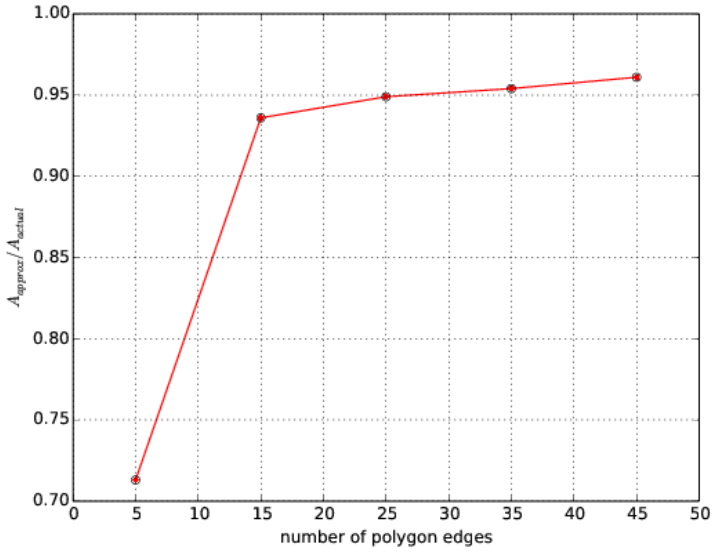


Figure 7.5: The ratios between the polygon and RS_{xy}'s areas for different polygon vertex number.

In the z direction, the lower and upper height limits are:

$$h_{lower} = z_0 + (v_h - \Delta v_h)t - h_r \tag{7.13}$$

$$h_{high} = z_0 + (v_h + \Delta v_h)t + h_r \tag{7.14}$$

where v_h is the obstacle's vertical speed and Δv_h is its possible change. h_r is the vertical safety threshold.

7.3.2 Path Generation

The path generation is similar to algorithm 3. The only difference is that the constraint in equation 6.2 is replaced by equations 7.1 and 7.2. That means the UAV cannot enter the obstacle's reachable set. Fig.7.6 illustrates the procedure to create intermediate waypoints. Infeasible trajectories (red) either fall into the reachable sets or reach out of the geo-fence. Feasible trajectories (black) connect the start and the goal through an intermediate waypoint. The comparison of path planning with and without reachable sets will be performed in section 8.3.

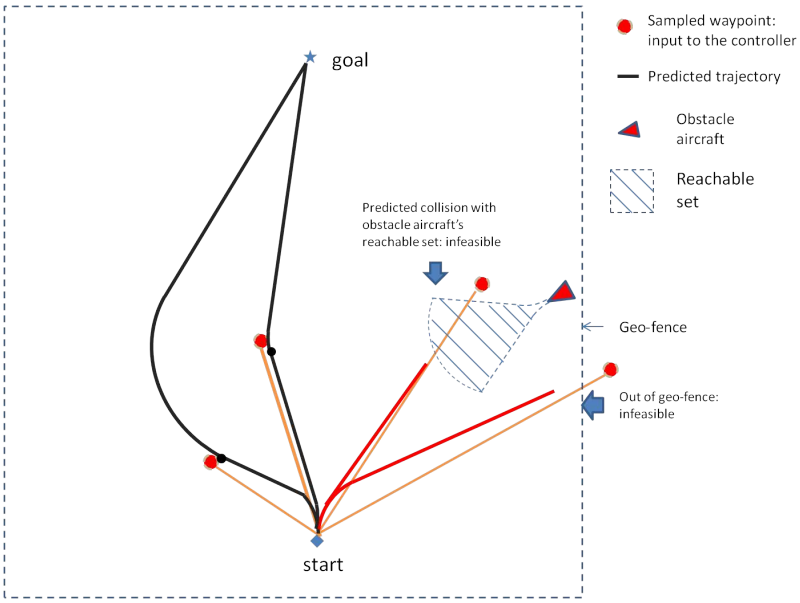


Figure 7.6: Illustration of creating candidate intermediate waypoints including reachable sets. UAV trajectories are generated using closed-loop prediction. The generated trajectories are then evaluated for feasibility. The black trajectories are feasible. The red trajectories are infeasible because they will enter the obstacle's reachable set or reach out of the geo-fence

Chapter 8

EXPERIMENTAL VALIDATION OF CLOSED-LOOP RRT AND ITS EXTENSIONS

8.1 Experimental Validation of Closed-Loop RRT

The system setup of this section is described in 3.4.1. This section describe the sampling strategy, the closed-loop dynamic system of the vehicle, the experiments, and the results.

8.1.1 Sampling Strategy

An UAV's state is defined as:

$$\mathbf{y} = (t, x, y, z, \psi, v_x, v_y, v_z) \tag{8.1}$$

It includes the UAV's position, velocity, yaw (ψ). A node in the CL-RRT tree contains a UAV state. A sample contains the position and the yaw. It is the input to the closed-loop system for generating a whole trajectory. To obtain a sample (x_s, y_s, z_s, ψ_s), (x_s, y_s) is first sampled by

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + r \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix} \tag{8.2}$$

$$\text{with} \begin{cases} r = \sigma_r n_r + r_0 \\ \theta \sim U(\theta_0 - \pi/2, \theta_0 + \pi/2) \end{cases} \tag{8.3}$$

where n_r is a random variable with standard Gaussian distribution, r_0 is the x-y distance between the root and the goal, and $\sigma_r = 0.5 r_0$ is the standard deviation. θ_0 is the yaw of the root node, θ is sampled uniformly in an angle of π spanned at root and centered in θ_0. z_s is calculated from (x_s, y_s).

Referring to figure 8.1, $\mathbf{e}_z$ is the unit vector along z, O is the root and B is the goal. $\overrightarrow{OA} = \overrightarrow{OB} \times \mathbf{e}_z$. The normal vector of plane AOB is

$$\mathbf{n}_p = \overrightarrow{OA} \times \overrightarrow{OB} / \| \overrightarrow{OA} \times \overrightarrow{OB} \| \tag{8.4}$$

We prescribe the sample point $P_s = (x_s, y_s, z_s)$ to lie in plane AOB. Therefore, $\overrightarrow{OP_s} \cdot \mathbf{n}_p = 0$. From it, we have

$$z_s = z_0 - [(x_s - x_0) n_p^x + (y_s - y_0) n_p^y] / n_p^z \tag{8.5}$$

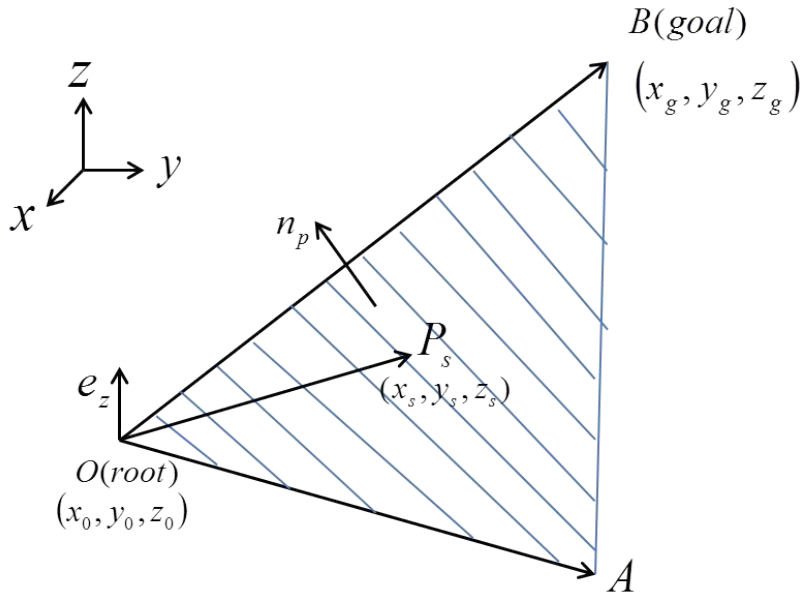


Figure 8.1: Demonstration of How Z Coordinate of a Sample is Calculated.

$n_p^z = 0$ corresponds to the trivial case when the root and the goal only differ in z. This case does not have practical significance. Therefore $n_p^z = 0$ will never happen. We set

$$\psi_s = \psi_0. \tag{8.6}$$

Equations 8.9,8.5 and 8.6 combined generate the sample.

8.1.2 The Closed-Loop System for Trajectory Prediction

This section describe how to generate a trajectory connecting two configurations of the UAV: (x_1, y_1, z_1, ψ_1) and (x_2, y_2, z_2, ψ_2) [1] . The trajectory generation includes two steps:

- Generate a 3D Dubins curve between two configurations.
- Follow the dubins curve to generate the trajectory.

3D Dubins Curve Generation

1 Generate a 2D Dubins curve C from (x_n, y_n, ψ_n) to (x_s, y_s, ψ_s) as in Shkel and Lumelsky (2001). A Dubins curve consists of a combination of three segment, each of which is ether a straight line or a circle arc.

[1] The full quadrotor model includes pitch and roll in addition. But position and yaw are sufficient for trajectory generation purpose.

2 Extend the Dubins curve to 3D: for each point (x, y) in C, $z = z_n + l(x, y) * tan(\alpha)$, where $l(x, y)$ is the length from (x_n, y_n) to (x, y) along C and $tan(\alpha) = \frac{z_s - z_n}{l(x_s, y_s)}$. After the extension, a 3D Dubins curve C' consisting of a straight line and helices is obtained.

Following the Dubins Curve

The procedure to follow the Dubins curve is described in algorithm 5. The input to the algorithm is the

Algorithm 5 Following the Dubins Curve

1: **function** DUBINSFOLLOW($\mathbf{x}$,C')
2: Estimate the closet segment to $\mathbf{x}$. Say it is jth segment of C', denoted as C'_j
3: **for** $i = j$ to 3 **do**
4: **while** the end of C'_j is not reached **do**
5:

$$\mathbf{u}'(t) = g(\mathbf{x}(t), C') \tag{8.7}$$

$$\dot{\mathbf{y}(t)} = \mathbf{f}(\mathbf{y}(t), \mathbf{u}'(t)) \tag{8.8}$$

6: **end while**
7: **end for**
8: **end function**

UAV's position $\mathbf{x}$ and a Dubins curve C'. First, the algorithm estimates the closest segment of C' to $\mathbf{x}$. Since the UAV is initially at the starting location of the Dubins curve, the first segment is the closest one. Based on the UAV's position and the segment's mathematics equation (for example, the equation of a straight line or a helix), velocity command can be calculated using the vector-field path following method (Owen *et al.* (2013)). This is represented by equation 8.7. It corresponds to the controller of the closed-loop dynamics system as in Figure 6.1. Equation 8.8 corresponds to the vehicle's kinematic model in Figure 6.1. The model is illustrated in Figure 8.2.

Figure 8.2a describes the model for x and the same for y and z. As in the figure, velocity v_x and velocity command u_x are transformed into the body reference frame (TB). The error between them is used as the input to a P-controller. The output of the controller is thresholded by the AR.Drone's acceleration of saturation a_{max} to obtain the desired acceleration $\dot{v_x}$. The velocity output of the model $\dot{x}$ is calculated by integrating $\dot{v_x}$ and transforming back to the world reference frame (Inv(TB)). As in figure 8.2b, the desired yaw speed v_ψ is obtained with another P-controller. The input is the error between $atan(u_y, u_x)$ and the current yaw ψ. v_ψ thresholded by the saturation value $\dot{\psi}_{max}$ gives the

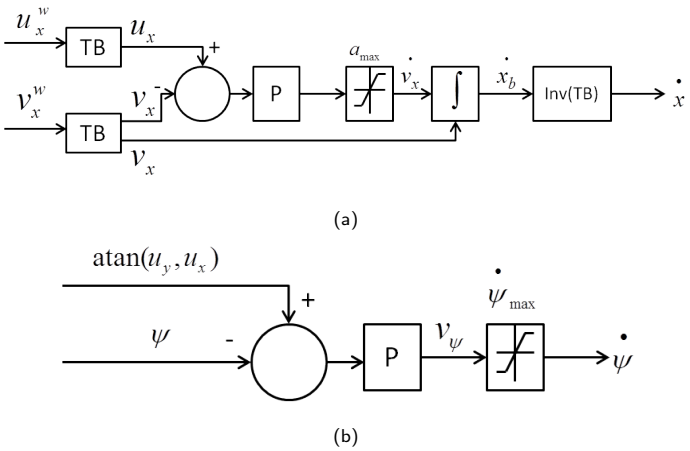


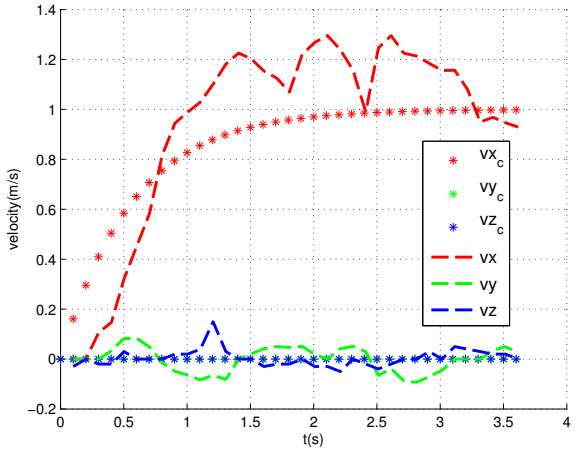
Figure 8.2: Approximated kinematic model of the AR.Drone. (a) describes the model for x and the same for y and z. As in the figure, velocity v_x and velocity command u_x are transformed into the body reference frame (TB). The error between them is used as the input to a P-controller. The output of the controller is thresholded by the AR.Drone's acceleration of saturation a_{max} to obtain the desired acceleration $\dot{v_x}$. The velocity output of the model $\dot{x}$ is calculated by integrating $\dot{v_x}$ and transforming back to the world reference frame (Inv(TB)). As in (b), the desired yaw speed v_Ψ is obtained with another P-controller. The input is the error between $atan(u_y, u_x)$ and the current yaw Ψ. v_Ψ thresholded by the saturation value $\dot{\Psi}_{max}$ gives the yaw rate $\dot{\Psi}$.

yaw rate $\dot{\psi}$. In the model, the yaw of the UAV is controlled to align with that of the velocity command. The AR.Drone was controlled in the same way in flight experiments. This way the AR.Drone always align its heading with the direction of its velocity. It flew like a fixed-wing aircraft and therefore can be considered as a proxy of a fixed-wing UAV. To compare the model with AR.Drone's actual behavior, both a real AR.Drone and the model were commanded to fly under the command $\mathbf{u} = (1, 0, 0)$ for 3.5 seconds and their states v.s. time are plotted in Figure 8.3. It is observed in Figure 8.3a that the velocity v_x of the model and a real UAV has accordance in evolving trend and the maximal error is around $0.2m/s$. A similar pattern is observed for x position in figure 8.3c. Yaw of the model and a real UAV almost overlap in figure 8.3b.

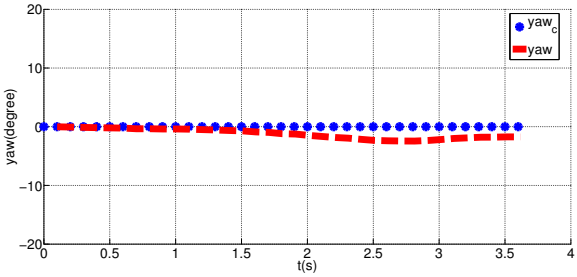
Figure 8.4 shows the simulated trajectory (black dots) using algorithm 5 along the Dubins curve (blue) connecting $z_n = (0, 0, 0.5, 0)$ and $z_s = (4, 2, 2, 0)$.

8.1.3 The Flying Experiments

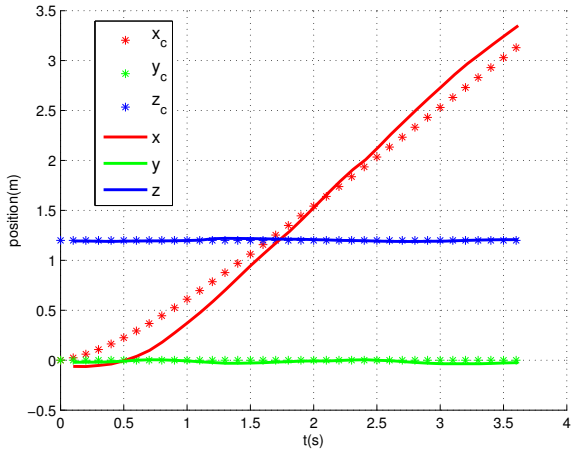
Three situations were designed for experiments: 1) avoid static obstacles in office condition; 2) avoid virtual moving obstacles; 3) avoid actual moving obstacles. 2) and 3) were carried out in a parking structure. The software architecture is detailed in 3.4.1. In all three situations, the origin of the coordinate system was set to the position of take-off location. The direction of x was set to the UAV's yaw direction



(a)



(b)



(c)

Figure 8.3: Comparison of the behavior of the kinematic model and a real AR.Drone. (a) compares the velocity, (b) compares the yaw, and (c) compares the position. Variables with the subscript "c" mean data from the model. Variables without subscripts come from a real AR.Drone.

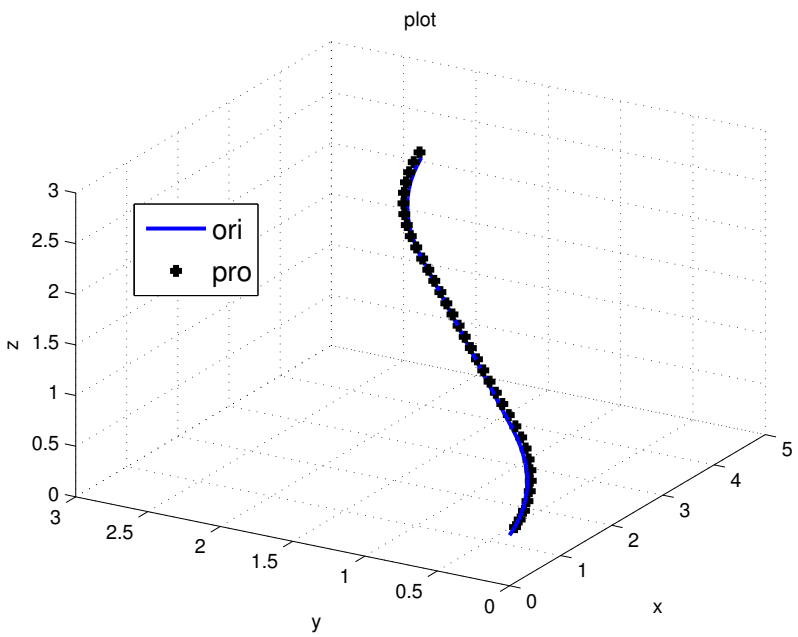
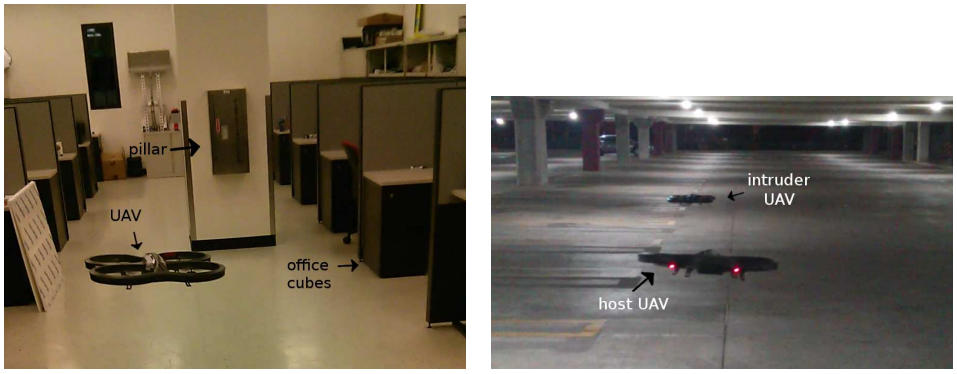


Figure 8.4: The simulated trajectory using the closed-loop system between configurations $z_1 = (0, 0, 0.5, 0)$ and $z_2 = (4, 2, 2, 0)$. The blue line is the Dubins curve and black dots are the trajectory.



(a) (b)

Figure 8.5: Two experiment situations: (a) avoid a static obstacle in an office; (b) avoid another AR.Drone in a parking structure.

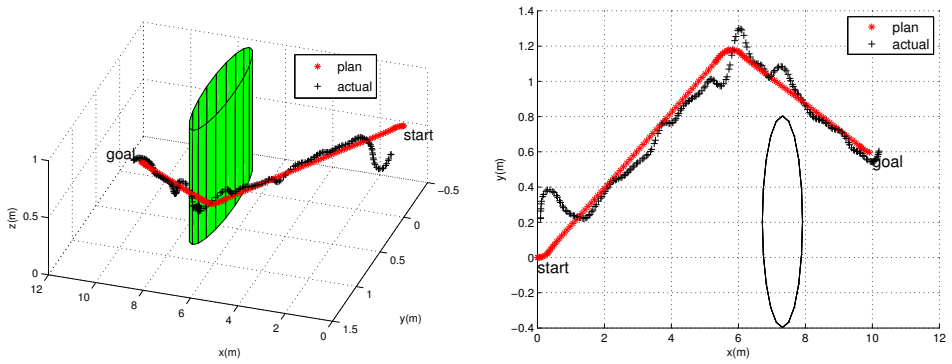
the moment it finished take-off. This defined the global reference frame. Meter was used as the length unit. Δt in algorithm 2 was set to 1s. In all experiments, instead of a full waypoint sequence, only a single start and goal pair was considered. It was used as the first step flight experiments to validate the path planning algorithm.

Flight Experiments in Office

The office had a pillar in the center as the obstacle and table cubes around as the geofence. The UAV need avoid all of them. The size and locations of the pillar and cubes were known to the path planner. The start was $(0, 0, 0.8)$ and the goal was $(10, 0.6, 0.5)$. The pillar located at $(7.31, 0.2)$ and its safety radius was set to 0.6 meter according to the pillar's size. The four corners of the geofence were set to $(-1.5, 0)$, $(1.2, 0)$, $(-1.5, 11.58)$ and $(1.2, 11.58)$ based on the office's area.

flight#	1	2	3	4	5	6	avg
min_dis(m)	0.83	0.74	0.94	0.92	0.95	0.83	0.87

Table 8.1: The Closest Distance Between the UAV and the Pillar in Six Flights in the Office.

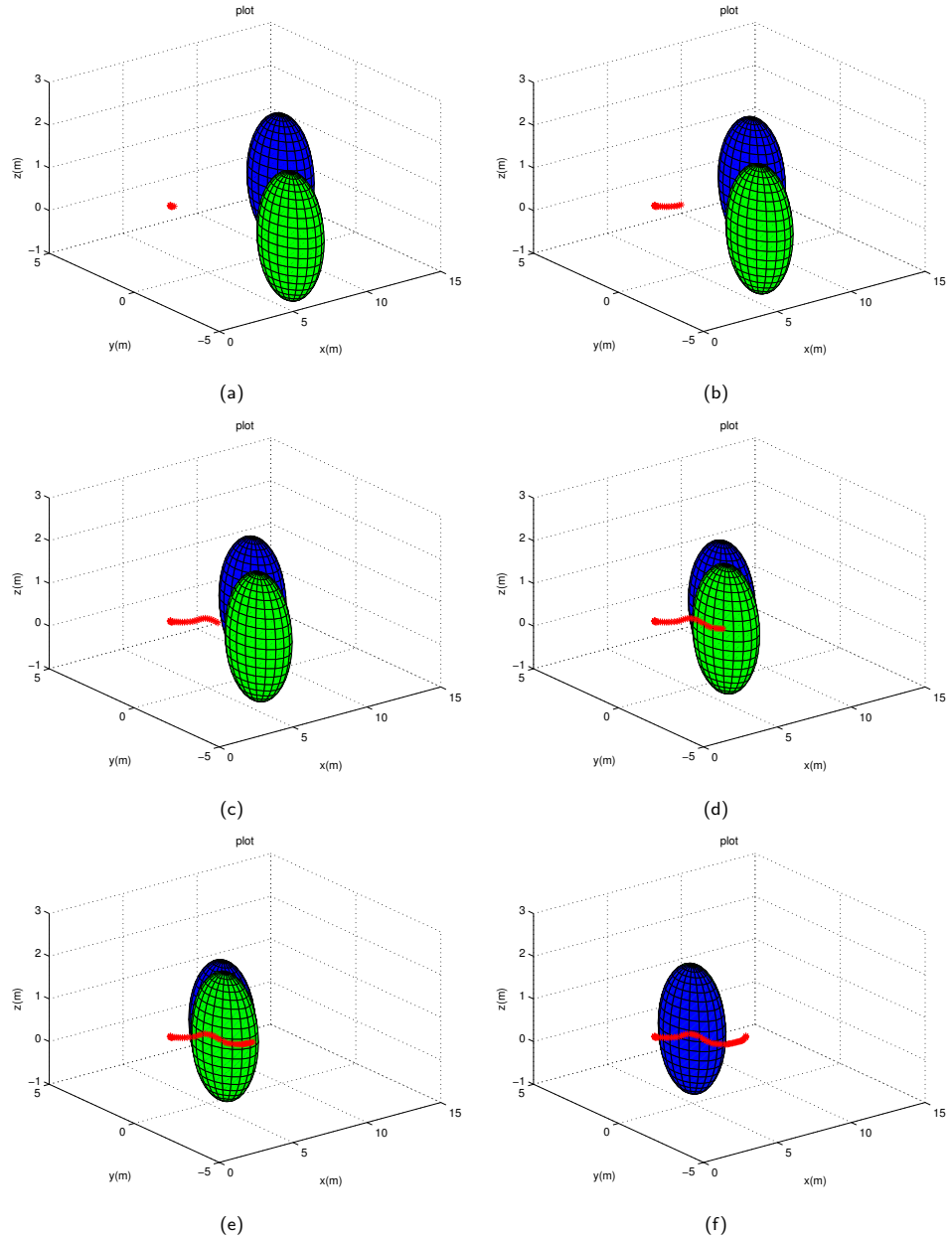


(a) (b)

Figure 8.6: The logged trajectory of one flight in the office. The red is the planned path and the black is the executed trajectory. (a) is from a 3D perspective and (b) from a top view. The green cylinder represents the pillar and its radius is 0.6m.

6 successful flights and 5 failing ones were performed. Failure was caused by the inaccuracy of position system. The logged data showed that the UAV's trajectory were clear from the pillar and cubes but actually the UAV collided with them. The magnetometer for yaw measurement was noisy and made the odometry drift from the real position. We didn't make efforts to improve the position system because localization was not the focus of this paper and a less accurate position system suffice to demonstrate the planning algorithm's efficacy. The minimum distance between the UAV and the pillar in the 6 successful flights are displayed in table 8.1. They are all above the 0.6m safety threshold, showing successful

avoidance. Logged trajectory of one flight of them is plotted in figure 8.6. It is observed in the figure that the UAV followed the path even though there were drifts. The difference of the UAV's start from the path's start was due to the drift when taking off.



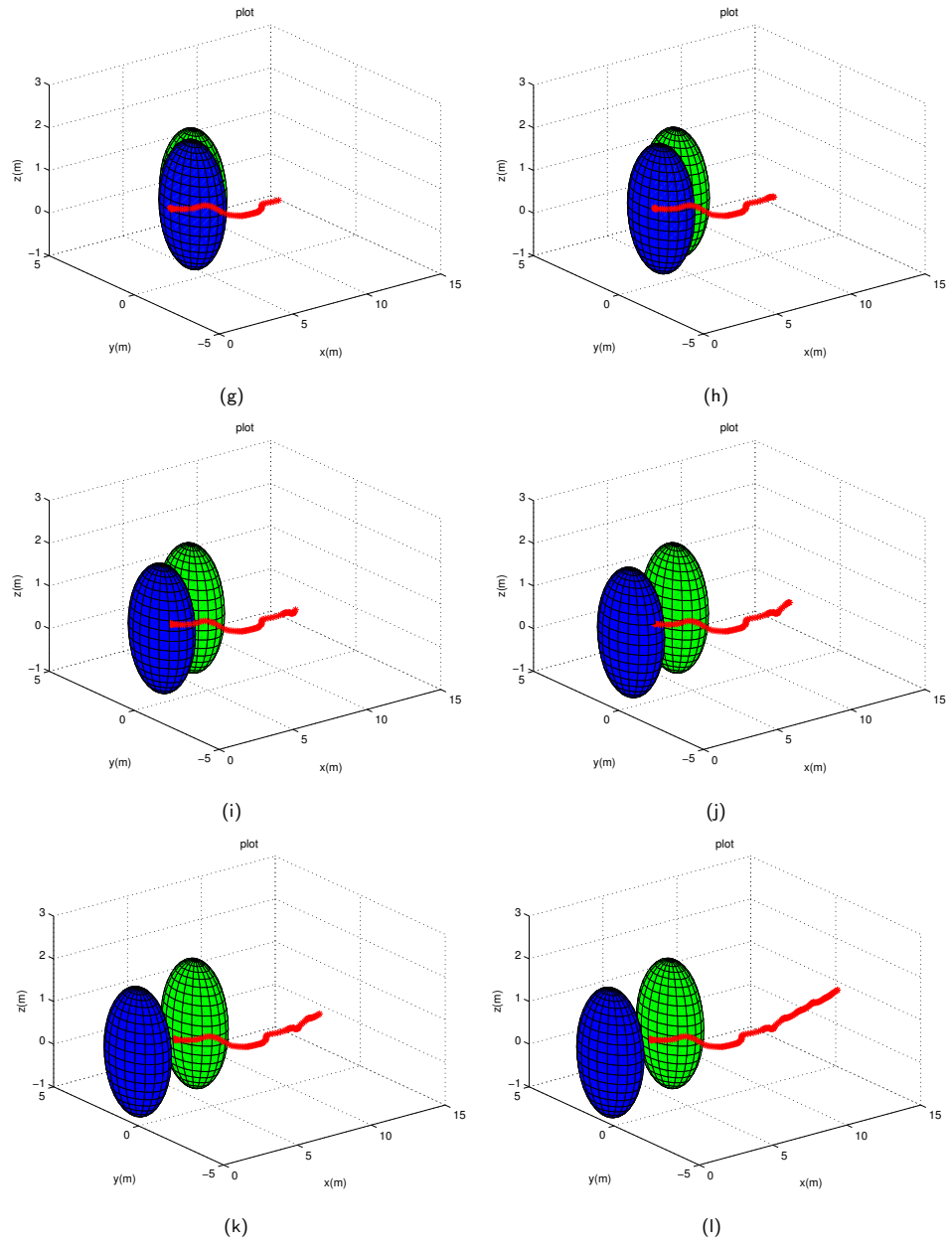


Figure 8.7: The logged trajectory of the AR.Drone (red) at different time instants to avoid two virtual obstacles. The green obstacle flew perpendicular to the original route of the AR.Drone and the blue obstacle flew opposite to the original route. Both obstacles have a safety radius of 1.5m.

Avoiding Virtual Moving Obstacles

Virtual Obstacles were logged trajectories of a quadrotor flying in simulation. *hector_quadrotor* [2] ROS package was used to create them. For example, a virtual obstacle flying towards the UAV can be created by flying the quadrotor in simulation from $(0, 0, 0.75)$ to $(10, 0, 0.75)$ by velocity command $v = (-1, 0, 0)$. The command was constant but the velocity of the obstacle was not. In all experiments, the UAV took off from $(0, 0, 0)$ and tried to fly to $(15, 0, 0.8)$. The length unit is meter.

Figure 8.7 shows the trajectory of the AR.Drone (red) avoiding two virtual obstacles (blue and green) at different time instants. Their radii stand for safety radii. The blue obstacle flew towards the AR.Drone from $(10, 0, 0.75)$ to $(0, 0, 0.75)$ under the command $v = (-1, 0, 0)$ and the green one from $(5, -5, 0.75)$ to $(5, 1, 0.75)$ under $v = (0, 1, 0)$. In the whole procedure, the UAV stayed outside the two spheres, indicating successful avoidance. More flight experiments were carried out with different virtual obstacles and combinations of them. The speed of the UAV was set to 1m/s. The closest distances between the AR.Drone and the obstacles were showed in table 8.2. In the table, vkx10 means that the obstacle was created to move from $(10, 0, 0.75)$ to $(0, 0, 0.75)$ under command $v = (-k, 0, 0)$. It flew to the AR.Drone head-to-head. v1y-k means that the obstacle started from $(k, -k, 0.75)$ and stopped at $(k, 1, 0.75)$, representing an obstacle approaching from the lateral direction. In row 1 to 4, the UAV avoided obstacles flying towards the UAV at different speeds. It is observed that the UAV was able to avoid obstacles of speeds up to 3m/s under the given take-off and goal position. Row 5 shows the results to avoid only a perpendicularly moving obstacle. The UAV was able to stay well clear of it during the whole trajectory in all 3 flights. In the next 3 rows, the UAV tried to avoid two obstacles. One moved head-to-head and another perpendicularly. The latter started from different locations. For each combination, the first row shows the smallest distance between the UAV and the first obstacle. The second row shows that between the UAV and the second. For all 3 combinations, the UAV maintained separations to two obstacles larger than safety threshold 1.5m in every flight. Therefore, the algorithm was validated in two-obstacle situations.

The efficacy of replanning was demonstrated in the procedure to avoid the obstacle v1y-7 as in figure 8.8. The obstacle's speed was close to zero when the UAV took off. The path generated by the planner was a straight line to the goal (figure 8.8a). Later the obstacle speeded up and the straight line path would result in collision. Therefore, a new path was created (figure 8.8b) with the UAV's current

[2] http://wiki.ros.org/hector_quadrotor

row#	virtual obs	1	2	3	avg
1	v1x10	2.97	2.78	1.89	2.55
2	v2x10	2.12	1.69	1.81	1.87
3	v3x10	1.97	1.86	2.21	2.01
4	v4x10	1.34	1.39	1.44	1.39
5	v1y-7	3.71	3.74	3.52	3.66
6	v1x10&v1y-5	1.95	2.62	2.89	2.49
		1.71	2.42	2.04	2.06
7	v1x10&v1y-6	2.71	1.80	2.65	2.39
		1.54	1.61	1.72	1.62
8	v1x10&v1y-7	2.97	2.76	2.47	2.73
		1.63	1.68	1.58	1.63

Table 8.2: The minimum distances (m) between the UAV and virtual obstacles in multiple experiments. vkx10 means that the obstacle was created to move from $(10, 0, 0.75)$ to $(0, 0, 0.75)$ under command $v = (-k, 0, 0)$. It flew to the AR.Drone head-to-head. v1y-k means that the obstacle started from $(k, -k, 0.75)$ and stopped at $(k, 1, 0.75)$, representing an obstacle approaching from the side. The unit is meter. Rows 1-5 are situations involving only one obstacle and rows 6-8 involve two obstacles. There flights were conducted for each type of obstacles. The average minimum distances are also presented.

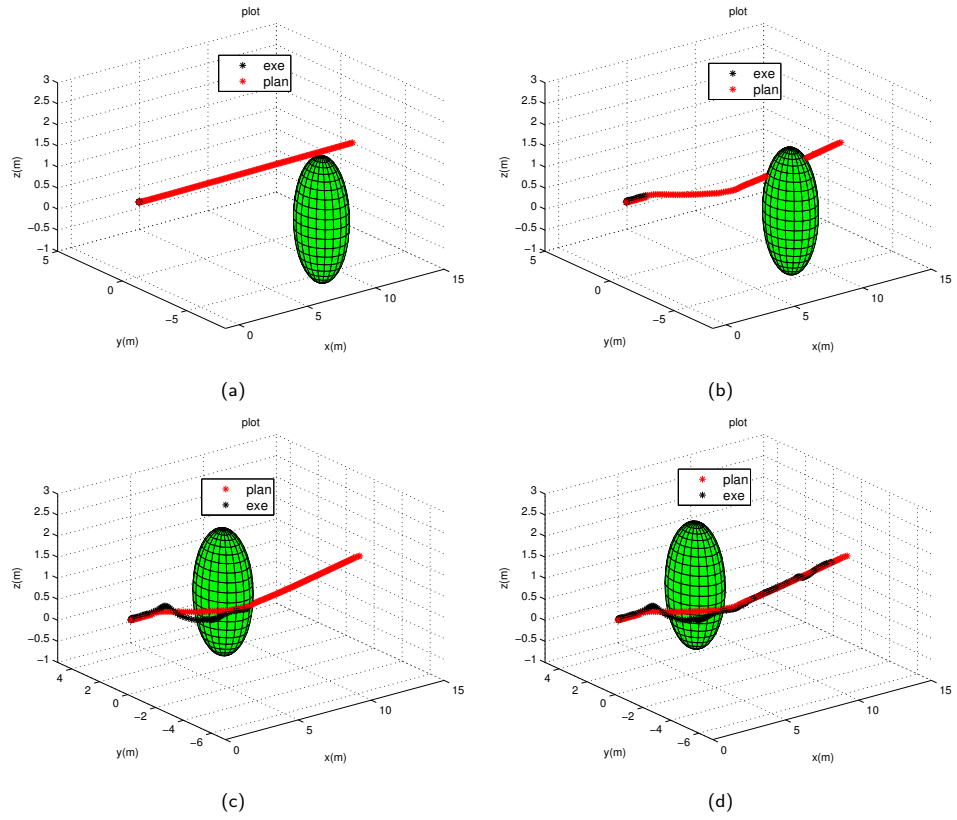
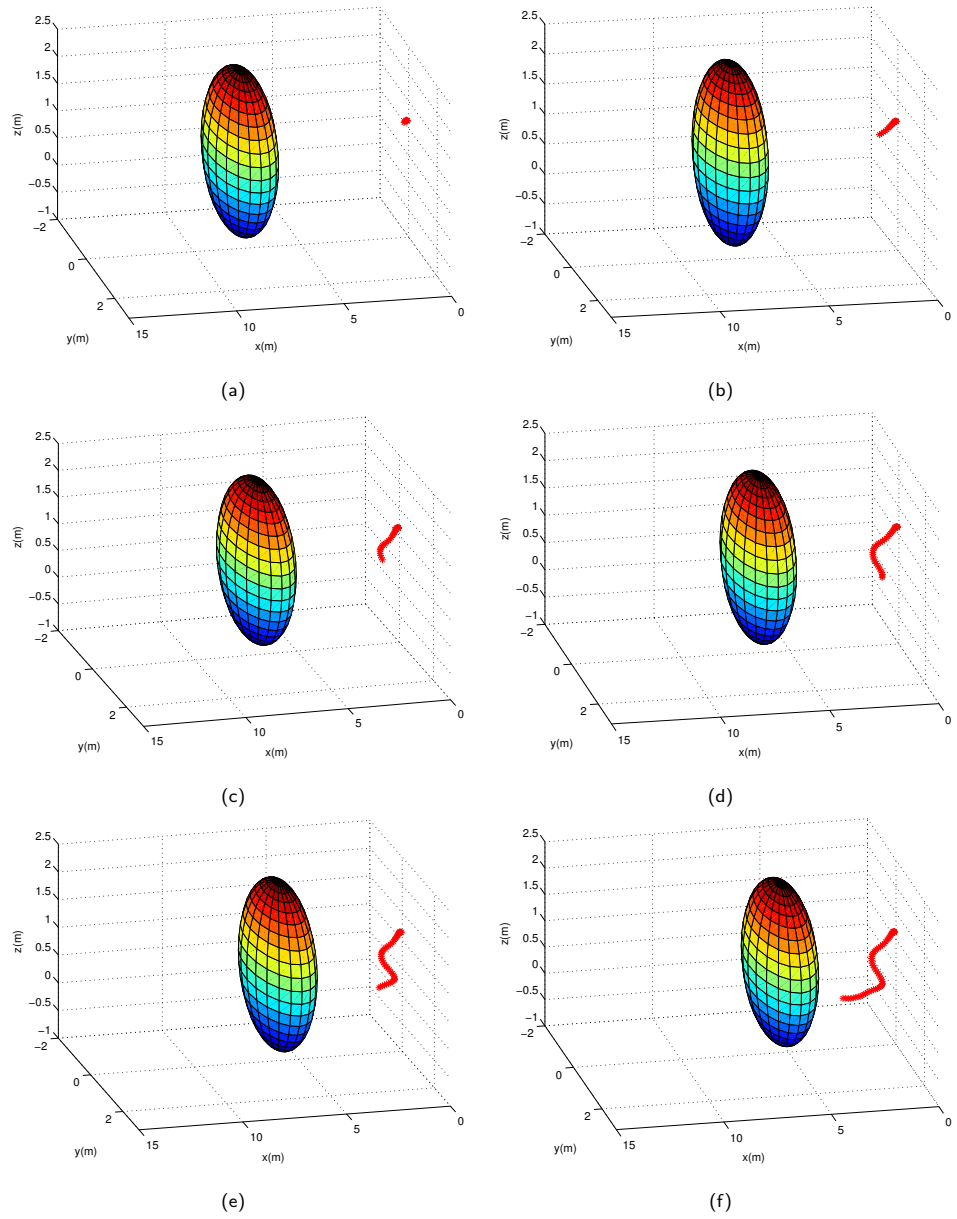


Figure 8.8: A replan example: in (a) a straight path (red) would not cause collision due to the small initial speed of the obstacle (green sphere of 1.5m radius). The obstacle however speeded up in (b) and the planner abandoned the previous straight path and created a new path to avoid the obstacle. The black dots mean the executed trajectory when following the path.

position as the start. The UAV was able to avoid the obstacle by following the new path until reaching the goal (figure 8.8c and 8.8d).



Avoiding an Actual Moving Obstacle

Another AR.Drone was used as the moving obstacle (figure 8.5b). It was controlled by a separate laptop. Its states were transmitted to the laptop through Wifi. They were next transmitted to the laptop controlling the host AR.Drone through Ethernet. This way the planner knew the states of the moving

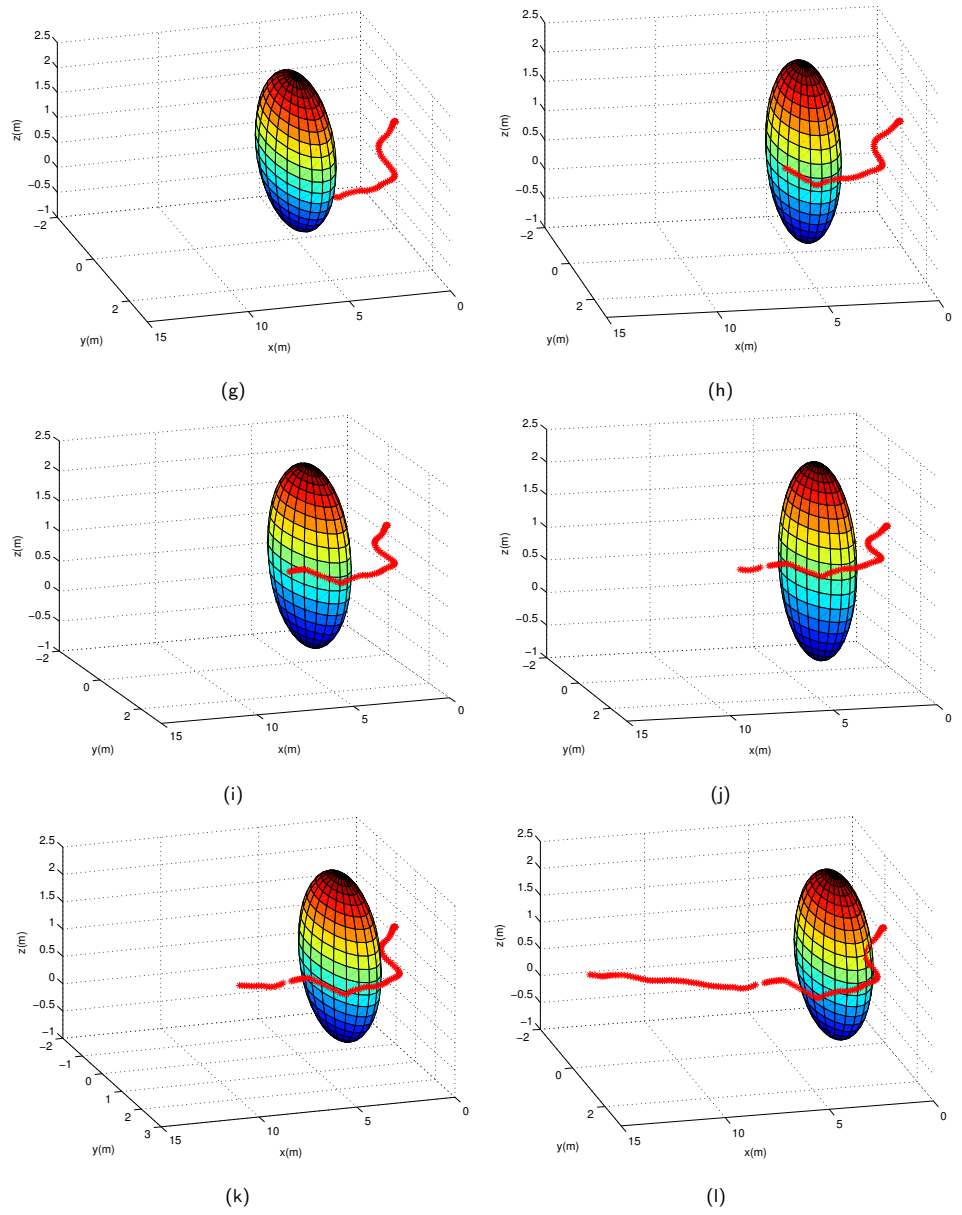
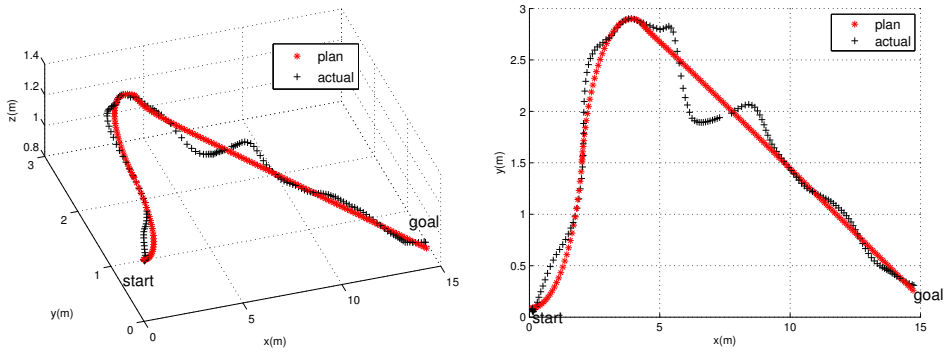


Figure 8.9: The logged trajectory of the UAV (the red dot curve) at different time instants to avoid a real moving obstacle. The sphere represents the logged obstacle's position at each instant. Its radius represents the obstacle's safety radius and is 1.5m. In all the figures, we observe that the UAV (left end of the red curve) always stays outside the sphere, indicating the success of avoidance.



(a) (b)

Figure 8.10: The logged trajectory (black) and planned path (red) of the flight in figure 8.9.

obstacle. The obstacle AR.Drone flew towards the host UAV's taking-off position by following a straight line. The host AR.Drone needed to avoid it and reach the goal. In the experiment, both AR.Drones first took off and started to hover. When both of them were in stable hover, the obstacle AR.Drone started to fly towards the host one and the host one started the execution loop in algorithm 2.

Figure 8.9 demonstrated a full avoidance procedure. Each figure shows a different time instant. The red is the logged trajectory of the UAV and the sphere is the obstacle. The end of the trajectory stands for the position of the UAV at the same moment as the obstacle. The obstacle started at $(7.753, -0.145, 0.735)$ and stopped at $(2.43, -0.04, 0.71)$. The UAV started from $(0.11, 0.08, 1.11)$ and stopped at $(14.74, 0.3, 0.85)$. The obstacle was commanded to fly at 0.5m/s and the UAV at 1m/s. In the figures, the UAV never enter the sphere whose radius is the safety distance (1.5m in our case). This proves that UAV successfully avoided the obstacle. The full trajectory for the same flight is plotted with the planned path in figure 8.10.

flight#	1	2	3	4	5	6	7	8	9	10	avg
v:1.0	2.58	1.74	2.65	2.29	2.28	2.11	3.45	2.33	2.41	1.63	2.35
v:0.5	2.55	1.99	2.22	1.98	1.58	1.44	1.90	1.99	2.37	3.83	2.19

Table 8.3: The closest distance (in meter) between the UAV and the moving obstacle. The speed of the obstacle is 0.5m/s and that of the UAV is 0.5m/s or 1.0m/s. We carried out 10 flights for each speed. Only the minimum distance at 6th flight of 0.5m/s is slightly below the 1.5m safe separation threshold. All the rest flights successfully prove the efficacy of collision avoidance. The last column is the average minimum distance.

Multiple flights were conducted with the obstacle's speed at 0.5m/s and the UAV's speed at 0.5m/s or 1.0m/s. In all of them, the obstacle AR.Drone took off at $(7.5, 0, 0)$ and fly along a straight line until reaching $(-2, 0, 0.75)$. Table 8.3 lists the smallest distance between the obstacle and the UAV in all flights. From the table, the smallest distance of only one flight was slightly below the safety threshold

(1.5m). This proves the efficacy of the algorithm to avoid a real moving obstacle. Flight videos are available at `http://www.youtube.com/watch?v=ggS5-x0rjNM`. Efficacy of the algorithm in avoiding virtual and actual moving obstacles were examined using logged position data based on optical flow. The logged position had similar drifts as in office condition although no physical collision occurred. In the absence of more accurate position systems such as motion capturing system, optical flow was used for positioning both the UAV and obstacles. The experiments were made as proxy to obstacle avoidance with positioning system of higher accuracy such as GPS.

Insights From the Experiments

It is found in the virtual obstacles experiment that the algorithm was unable to deal with obstacles faster than a certain speed given a fixed initial distance between the UAV and the obstacle. A local planner to generate reactive avoidance motion was needed. Implementing and integrating the local planner is one of future works. From the experiment in the office, we observed that drifts of the odometry caused collision with obstacles. This implied that the algorithm will fail with low accuracy of position systems. In the next section, local avoidance will be integrated and GPS will be used as the position system.

8.2 Experiment Validation of Greedy Closed-Loop RRT

8.2.1 Sampling Strategy

A sample waypoint is obtained by

$$\begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_g \end{bmatrix} + \begin{bmatrix} rcos\theta \\ rsin\theta \\ 0 \end{bmatrix} \tag{8.9}$$

$$\text{with} \begin{cases} r = \sigma_r n_r + r_0 \\ \theta \sim U(\theta_0 - \pi/2, \theta_0 + \pi/2) \end{cases} \tag{8.10}$$

where (x_0, y_0, z_0) is the start and (x_g, y_g, z_g) is the goal. n_r is a random variable with standard Gaussian distribution, r_0 is the x-y distance between the start and the goal, and $\sigma_r = 0.5r_0$ is the standard deviation. r_0 is thresholded by r_m to prevent a large deviation from the original route. θ_0 is the heading of the vehicle when it reaches the start node, θ is sampled uniformly in an angle of π spanned at the start and centered in θ_0. z_s is set to z_g.

8.2.2 The Closed-Loop System

This section describes the implementation of the controller and the UAV's model for the closed-loop prediction in Fig 6.1. The controller used in prediction was based on the actual controller used by the autopilot of the UAV. In our application, *APM:Plane* [3] was used. Its controller includes *L1-controller* Park *et al.* (2004) for navigation and *Total Energy Control System*(TECS) Shevchenko (2013) for controlling the speed and the height. The reference command for the L1-controller was the latitude and the longitude of the target waypoint and the outputs are roll and yaw commands for the UAV. The reference commands for TECS are target height and speed and the outputs are pitch and throttle commands for the UAV.

The UAV's dynamic model receives roll, yaw, pitch, and throttle commands and updates the UAV's state. They can be described in the following equations.

$$\dot{x} = Vcos(\beta)cos(\alpha) \tag{8.11}$$

$$\dot{y} = Vcos(\beta)sin(\alpha) \tag{8.12}$$

$$\dot{z} = Vsin(\beta) \tag{8.13}$$

$$\dot{V} = K_1(T_c/M - gsin(\beta)) \tag{8.14}$$

$$\dot{\beta} = K_2(\beta_c - \beta) \tag{8.15}$$

$$\dot{\alpha} = K_3(\alpha_c - \alpha) + \frac{g}{V}tan(\gamma_c) \tag{8.16}$$

where $(\gamma_c, \alpha_c, \beta_c, T_c)$ are roll, yaw, pitch and throttle commands. The UAV's state is

$$\mathbf{X} = (t, x, y, z, V, \beta, \alpha) \tag{8.17}$$

where (x, y, z) is the 3D position, V is the speed, β is the yaw and α is the pitch. K_1, K_2, K_3 are control gains. Universal Transverse Mercator coordinate system (UTM) is used as the geographical coordinate for (x, y). NEU convention is used for (x, y, z). We compare a predicted trajectory by the closed-loop system in Fig 6.1 with an actual flight trajectory of SITL. The UAV flew from waypoint $(33.42, -111.91, 669)$ to $(33.42, -111.94, 615)$. The initial heading of the UAV is 74° from the north. Figure 8.23 shows the comparison of the two trajectories. It is observed that the trajectories almost overlap in x. They don't overlap in y and z but show accordance in the changing trend. The differences in y and z are are less than 30 meters. In fig 8.11d, it is seen that the distance between the two trajectories in $x - y$ plane is less than 50 meters and at most of the time is less than 30 meters. The differences are small

[3] http://plane.ardupilot.com/

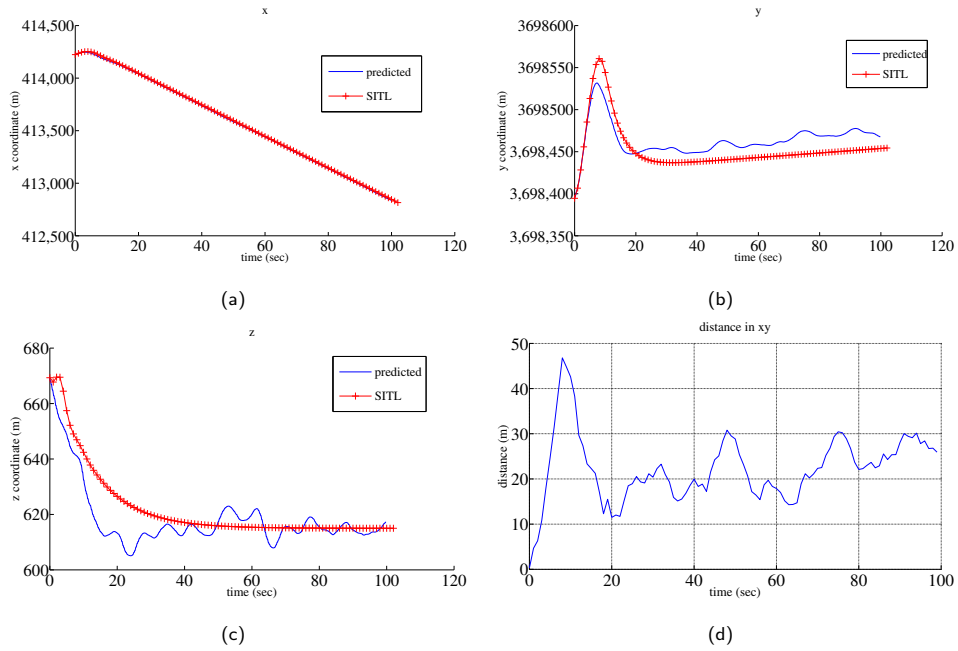


Figure 8.11: Comparison of predicted and actual flight trajectory of the UAV. (a)-(c) compare their difference in x, y and z respectively. (d) plots their distance in the $x-y$ plane at each time instance.

compared to the required separation from the UAV to other aircraft, in our case, 300 meters. There are no perfect overlaps between the predicted and actual trajectories because approximation and simplification are applied in controllers and UAV's model in the closed-loop system for prediction. In prediction, we are unable to simulate some sensors' readings which are used in the actual controller and UAV model. However, the implemented closed-loop prediction is sufficient for path planning.

8.2.3 Software-in-the-Loop Experiments

Software In the Loop (SITL) simulation was used as the first step test before real flight tests. In SITL, the key low level hardware drivers (such as gyros, accelerometers and GPS) run in the same way that they would run in a real flight. The architecture of SITL setup is detailed in 3.4.2.

In Software In the Loop simulation, the waypoints sequence 1→2→3→4 (as in Fig 3.14) was assigned to the UAV. The coordinates of the waypoints were:1(33.422,-111.909,615),2(33.422,-111.934,615), 3(33.407,-111.940,615),4(33.408,-111.909,615). The UAV used in SITL was commanded to fly at 30 m/s. The average speeds for the six logged ADS-B aircraft were 62, 250, 255, 125, 75, 201 m/s respectively. They didn't fly with constant speed. In SITL, logged ADS-B data were played back at 1HZ (the frequency of ADS-B communication) and the path planner received the data and used them in path

planning. ADS-B data logging is described in 3.4.2. The time horizon for collision detection was 30 seconds. Different offsets were added to the logged data to induce collision with the UAV at different time instances. The UAV was initially flown in SITL along the waypoint sequence without introduction of obstacles. A full trajectory $\mathcal{X}'(t)$ from the first to the last waypoint was obtained. To make an aircraft encounter the UAV at time t, we add an offset to the logged data:

$$\mathbf{x}_{off} = \mathbf{x}'(t) - \mathbf{x}_{ac}(t) \tag{8.18}$$

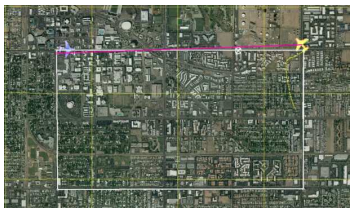
where $\mathbf{x}'(t) \in \mathcal{X}'(t)$. $\mathbf{x}_{ac}(t)$ is the position of the aircraft at t according to the logged ADS-B data. In the experiment, offsets were created by assigning $t = 40, 60, 100, 120, 140,\ 160, 180, 200, 220, 240, 260$ seconds.

Fig 8.12 shows three typical situations to avoid a single aircraft, where the UAV has head-on, lateral and tail encounters with the aircraft. Using the first column, head-on encounter, as an example to explain the figures: the UAV plans to fly from waypoint 1 to 2 by following the straight path between them. The collision detection algorithm detects a collision with the coming aircraft (blue icon in the figures). The white cross indicates the predicted collision location. The path planner generates a path to avoid the aircraft. The path (marked with yellow) starts from waypoint 1 and ends at waypoint 2. It goes through a generated intermediate waypoint W. In Figure 8.12g, 8.12j, 8.12m, and 8.12p , the UAV maintains a separation to the aircraft by following the path. Columns 2 and 3 show the case of lateral and tail encounters.

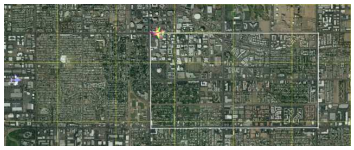
Fig 8.14 shows the situation of simultaneously avoiding two aircraft. From the figure, we observe that the path planning algorithm is applicable to two aircraft avoidance. Collisions of the two aircraft with the UAV happen closely in the original flight route. Otherwise, it is a problem of avoiding two single aircraft in sequence. Some SITL tests videos can be found at `http://www.public.asu.edu/~ylin122/ADSB.mp4`.

To further test the algorithm, we created different encounters of aircraft and the UAV by randomly selecting the offsets to the logged ADS-B data. The tests include one, two, and three aircraft. The UAV may come across multiple aircraft simultaneously or in sequence. We performed 20 tests involving one, two and three aircraft respectively. Real and simulated ADS-B data were used. Each test may use different aircraft. The same aircraft may encounter the UAV at different time instance in different tests.

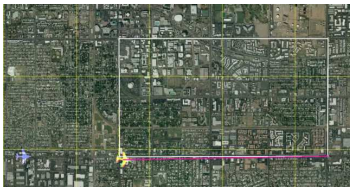
In collision avoidance, the UAV should keep the other aircraft from its safety volume. The safety volume used in the experiments is a cylinder centered at the UAV's position with bottom circle radius of



(a) head encounter:0



(b) lateral encounter:0

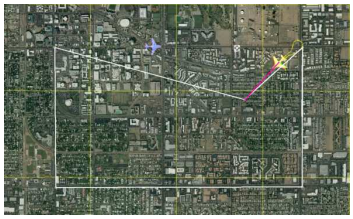


(c) tail encounter:0

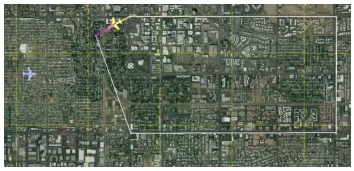
(d) head encounter:1

(e) lateral encounter:1

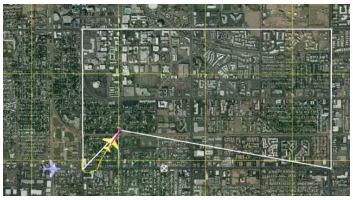
(f) tail encounter:1



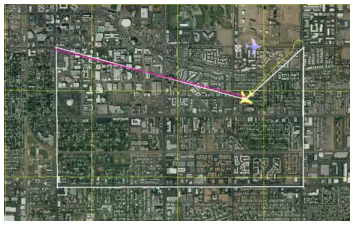
(g) head encounter:2



(h) lateral encounter:2

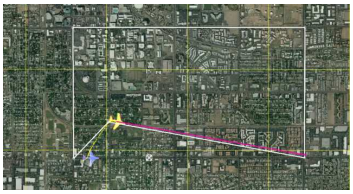


(i) tail encounter:2



(j) head encounter:3

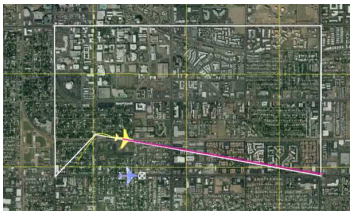
(k) lateral encounter:3



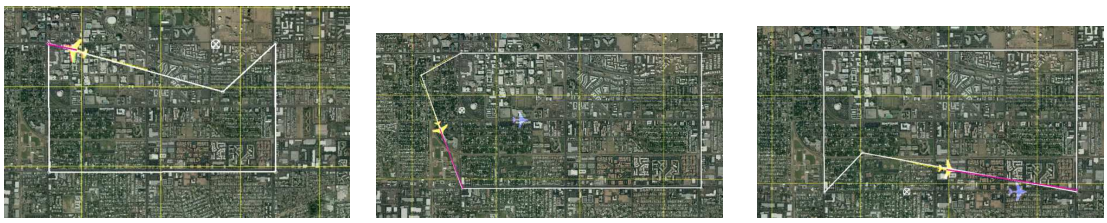
(l) tail encounter:3

(m) head encounter:4

(n) lateral encounter:4



(o) tail encounter:4



(p) head encounter:5 (q) lateral encounter:5 (r) tail encounter:5

Figure 8.12: Path generation to avoid a single aircraft for greedy closed-loop RRT. The green icon is the UAV and the blue is the obstacle aircraft. The white cross indicates the predicted collision location. Three columns show the situation of head-on, lateral and tail encounters with the aircraft. The second figure of each column shows the generated path in yellow. From the third to the last figure of each column, the UAV follows the generated paths and avoids the aircraft.

300 meters and height 100 meters. The values are based on the rule of NASA UAS Operation challenge [4] . This means that an aircraft should maintain horizontal distance larger than 300m or vertical distance larger than 50m. In Fig 8.15, we plot the horizontal and vertical distances between the UAV and the aircraft when the UAV is closest to each aircraft. The rectangle marks the safety volume. It is observed from the figure that the collision avoidance algorithm fails in only five encounters. In some of the failures, the horizontal distance between the UAV and an aircraft is only slightly lower than 300m threshold.

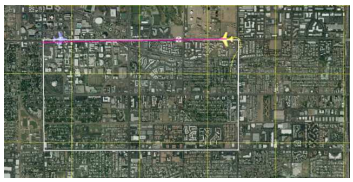
8.3 Experiment Validation of Planning Using Reachable Sets

Software-In-the-Loop experiments of the same setup were used. We first show in Fig.8.16 the path generation and collision avoidance procedure in three representative situations, namely head-on, lateral and double-aircraft encounters. The second column is used to explain the avoidance procedure.

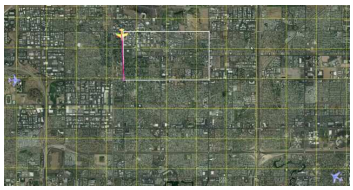
By default, the UAV needs to fly through waypoints sequence 1→2→3→4. A collision (the white cross) was predicted in Fig 8.16b and the avoidance path (red path) was generated in Fig 8.16e using Algorithm 3. After that, the UAV should fly to the intermediate waypoint W before to waypoint 3. The UAV follows the new waypoint sequence 1→2→W→3→4 and avoids a collision with the obstacle aircraft in Fig 8.16h, 8.16k, 8.16n, and 8.16q. In all experiments, the time interval for path generation Δt was set to 1 second.

We next compare the performance of the reachable set and linear motion assumption used in equation 6.3 to predict the obstacle aircraft's motion. An offset was added to one aircraft trajectory so that it will encounter the UAV at the point marked by a black circle in Fig.8.17. 20 SITL experiments were performed. The linear motion assumption was used in 10 of them and the reachable set is used in the

[4] http://www.uasaoc.org/MEDIA/2014_UAS_AOC_Rules_Final.pdf



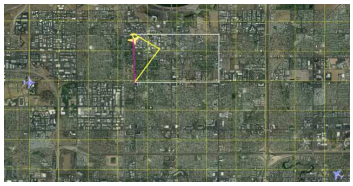
(a) case A:0



(b) case B:0

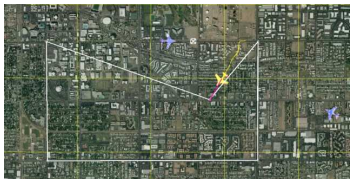
(c) case C:0

(d) case A:1



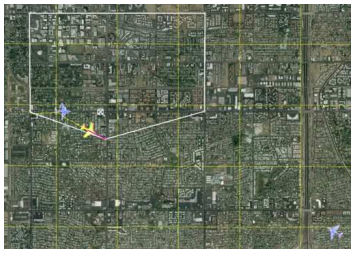
(e) case B:1

(f) case C:1

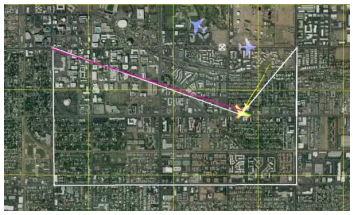


(g) case A:2

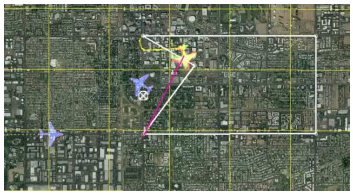
(h) case B:2



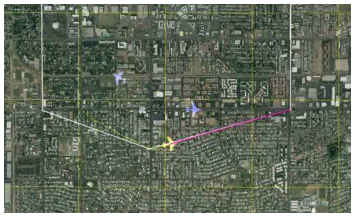
(i) case C:2



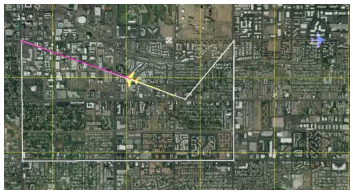
(j) case A:3



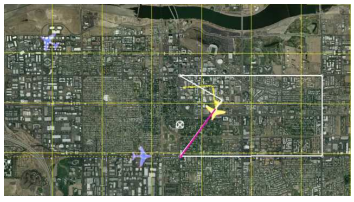
(k) case B:3



(l) case C:3

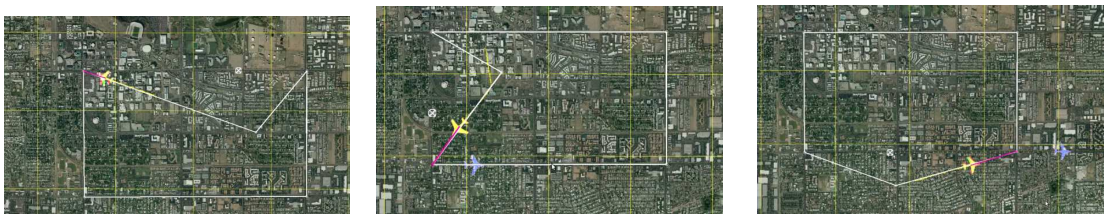


(m) case A:4



(n) case B:4

(o) case C:4



(p) case A:5 (q) case B:5 (r) case C:5

Figure 8.14: Path generation to simultaneously avoid two aircraft for greedy closed-loop RRT. The green icon is the UAV and the blue is the obstacle aircraft. The white cross indicates the predicted collision location. In each column, encounters happen in different sections of the original flight plan. Each column shows a different situation. Collisions of the two aircraft with the UAV happen closely in the UAV's original route in each case.

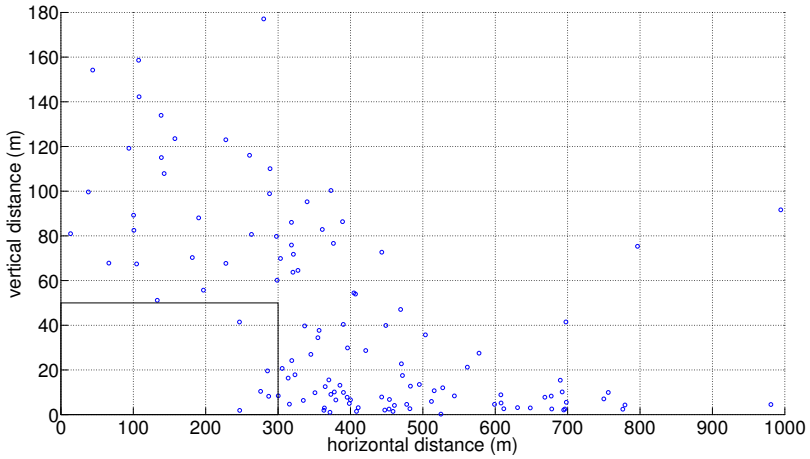
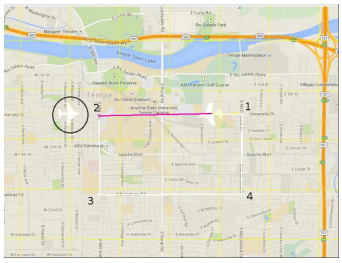


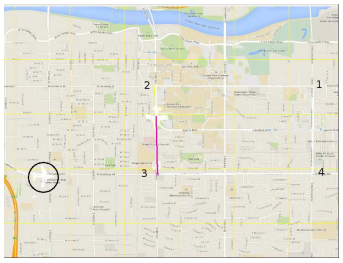
Figure 8.15: Scatter plot of horizontal and vertical distances when the UAV and aircraft were closest in all 120 encounters. Successful collision avoidance requires that a scatter point should not fall into the rectangle with 300m as its width and 50m as its height. It is observed only 5 encounters in total fall into the rectangle.

rest. The horizontal and vertical distances between the UAV and the aircraft when they were closest to each other are plotted in Fig.8.18. The blue circles stand for the results using the reachable set and red crosses the linear motion assumption. The rectangle indicates the safety zone around the UAV. It corresponds to a cylinder whose height is 100 meters and bottom circle radius is 300 meters. It is observed that the obstacle do not enter the safety zone in all 10 runs with the reachable set. However, the linear motion assumption gives 6 failures. Such failures are predictable because the obstacle aircraft is not moving along a straight line when it encounters the UAV.

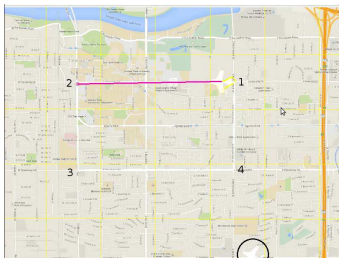
To further test the algorithm, offsets were added to the same ADS-B trajectory (UA787) so that encounters happen in different time instance after the UAV takes off. The time instances selected were $t = 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260,$
$280, 300s$. For each offset, the experiments were run 10 times. In Fig.8.19, we plot the horizontal and



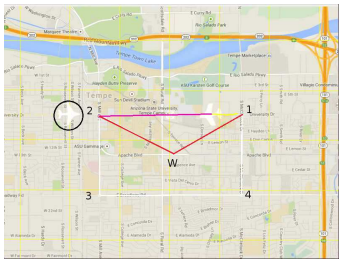
(a) head encounter:0



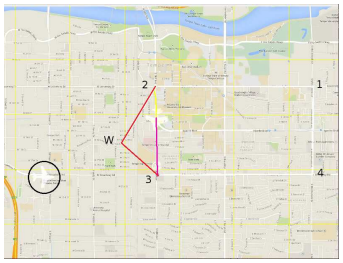
(b) lateral encounter:0



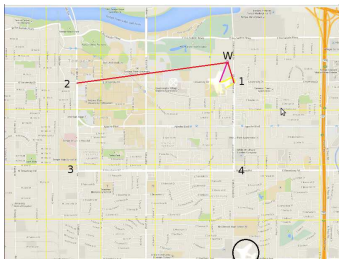
(c) double aircraft:0



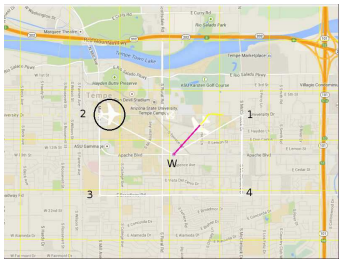
(d) head encounter:1



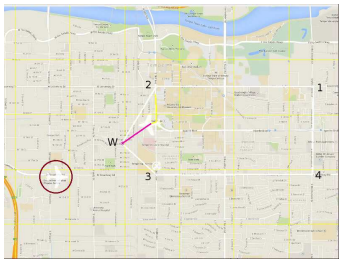
(e) lateral encounter:1



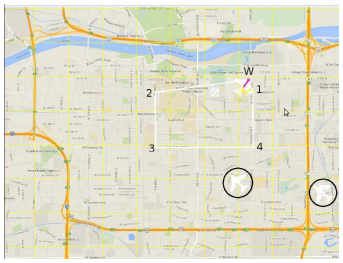
(f) double aircraft:1



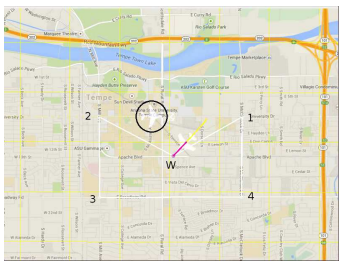
(g) head encounter:2



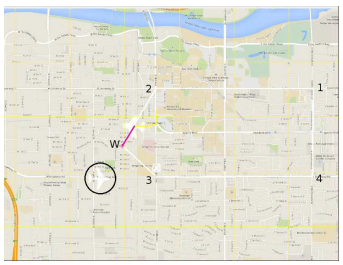
(h) lateral encounter:2



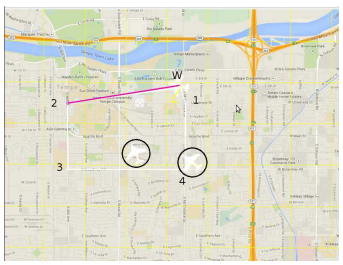
(i) double aircraft:2



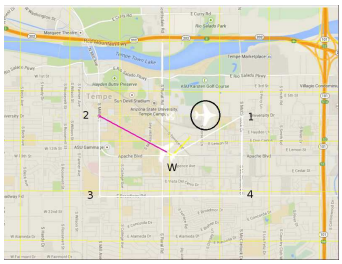
(j) head encounter:3



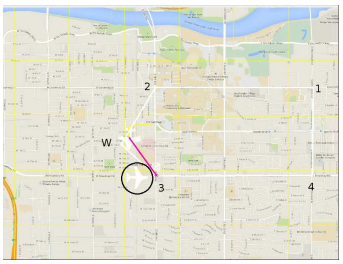
(k) lateral encounter:3



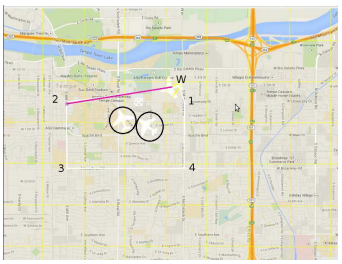
(l) double aircraft:3



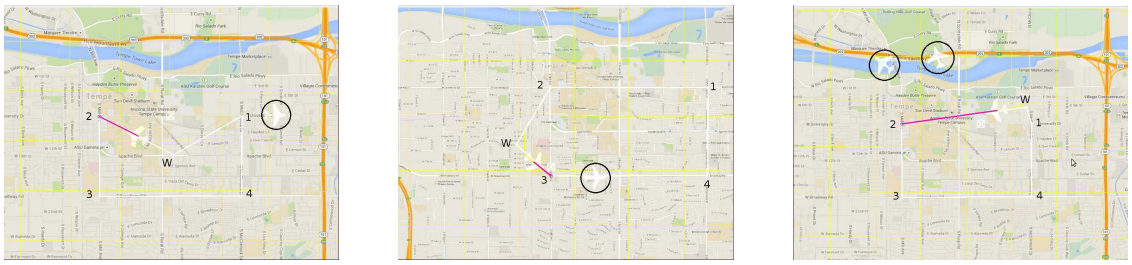
(m) head encounter:4



(n) lateral encounter:4



(o) double aircraft:4



(p) head encounter:5 (q) lateral encounter:5 (r) double aircraft:5

Figure 8.16: Path generation to avoid one or two obstacle aircraft using reachable sets. The circled aircraft icons are obstacle aircraft. The white cross indicates the predicted collision location. The first two columns show the situation of head-on and lateral encounters with a single obstacle aircraft. The third one shows the encounter with double aircraft. The second figure of each column shows the generated path in red. From the third to the last figure of each column, the UAV follows the generated paths and avoids the obstacle aircraft. By default, the UAV needs to fly through waypoints sequence 1→2→3→4. The path planner generates an intermediate waypoint W. The UAV follows the new waypoint sequence (1→2→W→3→4 in the second column for example) and avoids a collision with obstacle aircraft.

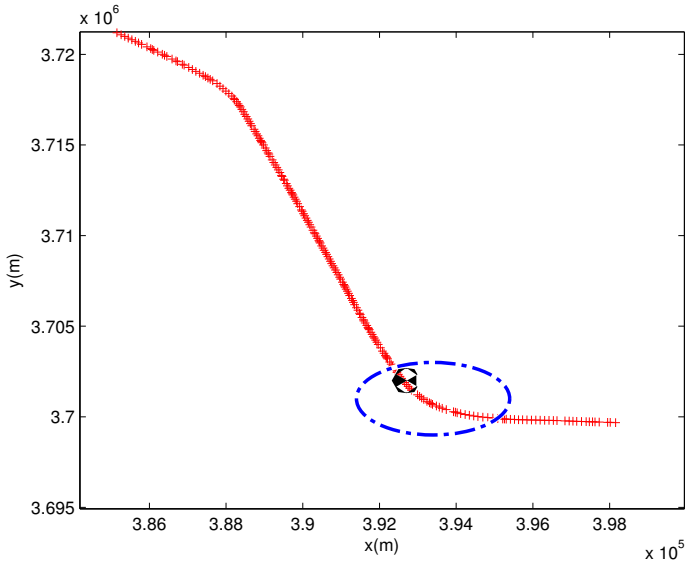


Figure 8.17: Part of a logged ADS-B trajectory of an actual aircraft (UA787). The blue dashed ellipse indicates the section that the aircraft makes a right turn. The black circle indicates the set encounter point for the experiment.

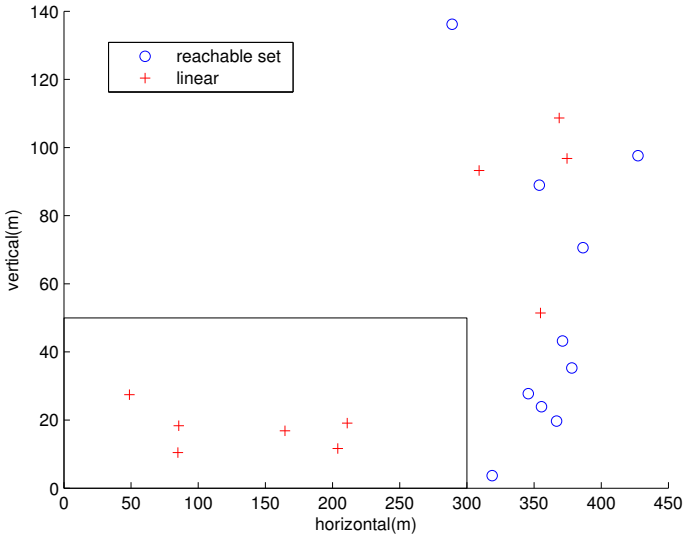


Figure 8.18: The horizontal and vertical distances between the UAV and the obstacle in Fig.8.17 when they were closet to one another. The encounter is set as in Fig.8.17. The blue circles stands for collision checking using the reachable set. The red cross stands for the situation using linear motion assumption. The rectangle is the safety zone.

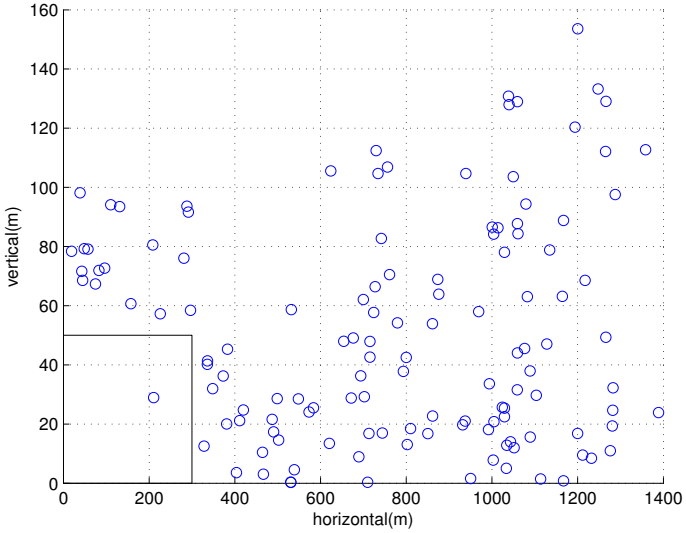


Figure 8.19: The horizontal and vertical distances between the UAV and the obstacle UA787 when they were closet to one another. Encounters happen at different time instances after the UAV takes off. The rectangle shows the safety zone.

vertical distances between the UAV and the obstacle aircraft when they were closest to each other. It is seen that the obstacle entered the UAV's safety zone in only one encounter. This results in 99.23% success rate.

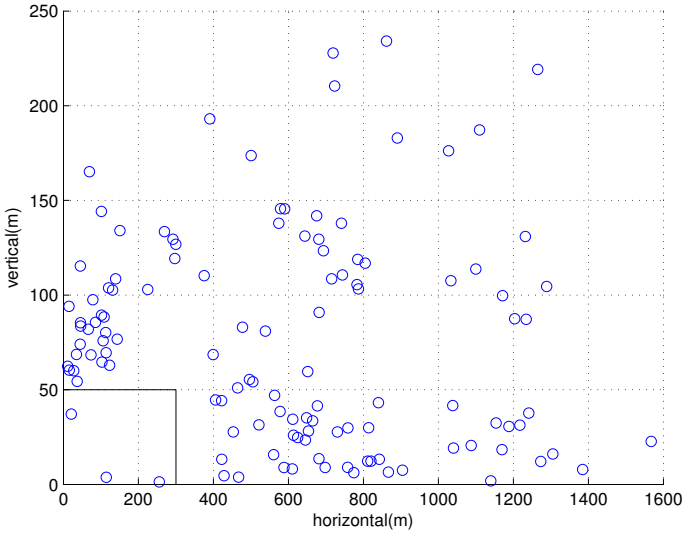


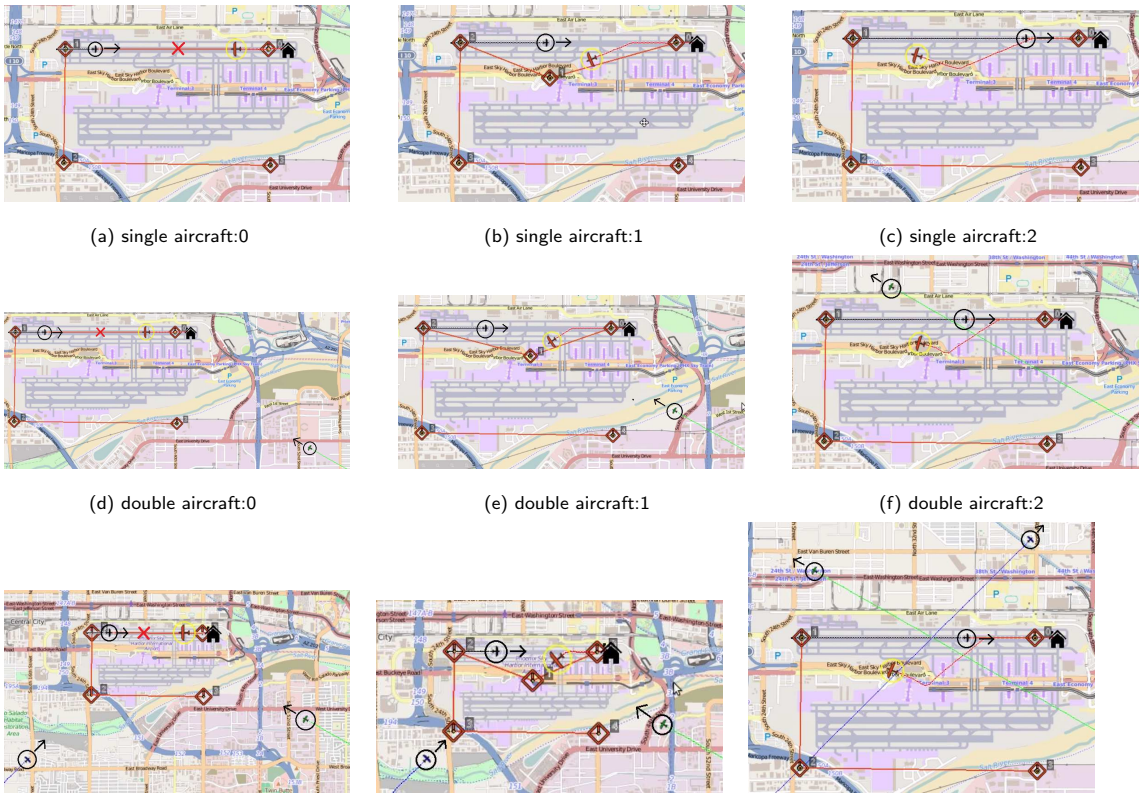
Figure 8.20: Scatter plot of horizontal and vertical distances between the UAV and obstacle aircraft when they were closest (using reachable sets). The offsets and combinations of aircraft were randomly selected. 20 tests involving one, two, or three aircraft were performed. 120 encounters happen in total.

As a more extensive test, same tests involving random obstacle aircraft as in section 8.2.3 were performed. Fig.8.20 shows the horizontal and vertical distances between the UAV and aircraft when they were closest in all 120 encounters. We observe that only in 3 encounters the avoidance fails. This gives us a 97.5% success rate. It improves the 95% success rate using the linear motion assumption in Fig. 8.15.

8.4 Experiment Validation of the Method Using Intermediate Points

8.4.1 Hardware-in-the-Loop Experiment

The path guiding controller and the UAV's model in the closed-loop system were the same as in section 8.2.2. Similar to previous Software-In-the-Loop experiments, logged ADS-B data of commercial aircraft's trajectories were used as the obstacle aircraft and will be played back in the simulation. The only difference is that the speeds of the obstacle aircraft were scaled according to the speed of the fixed-wing UAV used in HIL. Their representative speeds were 31, 125, 127.5, 62.5, 37.5, 100.5 m/s respectively. The representative speed of the UAV was 15 m/s. The UAV needs to fly through a given waypoint sequence 1→2→3→4 as in Figure 8.21a. The coordinates of the waypoints were:1(33.440,-111.996,1436.7),



(a) single aircraft:0 (b) single aircraft:1 (c) single aircraft:2

(d) double aircraft:0 (e) double aircraft:1 (f) double aircraft:2

(g) three aircraft:0 (h) three aircraft:1 (i) three aircraft:2

Figure 8.21: Path generation to avoid obstacle aircraft (using intermediate points). The red icon with yellow circle is the UAV and the black circles mark the location of obstacle aircraft. The black arrows show their moving directions. The three rows present the procedure to avoid one to there aircraft respectively. In each row, the UAV first generate a avoidance waypoint (waypoint 1 in the figures of the second column) when collision is predicted. Then it flew towards the avoidance waypoint. The UAV heads towards the original goal waypoint after reaching the avoidance waypoint (the figures of the third column).

2(33.441,-112.029,1436.7), 3(33.425,-112.029,1436.7), 4(33.425,-111.995,1436.7). The immediate next waypoint was defined as the goal in path planning. Figure 8.21 shows the path generation procedure from the map of HIL software to avoid single, double, and three obstacle aircraft. Taking the single aircraft case (the first row) as an example, in Figure 8.21a the UAV flew to the goal waypoint 1 according to the original plan and detects a collision (indicated by a red cross) with obstacle aircraft. In Figure 8.21b the avoidance waypoint (waypoint 1) was generated and the UAV flew to the waypoint and avoid collision with the obstacle aircraft. In Figure 8.21c the UAV passes the obstacle aircraft and the avoidance waypoint was removed from the waypoint sequence. The UAV flew to the original goal. Figures in the second and the third rows show the situation involving more than one obstacle aircraft.

To throughly test the algorithm, we performed HIL tests involving different obstacle aircraft with different encounter time. The aircraft and their encounter time with the UAV were randomly selected.

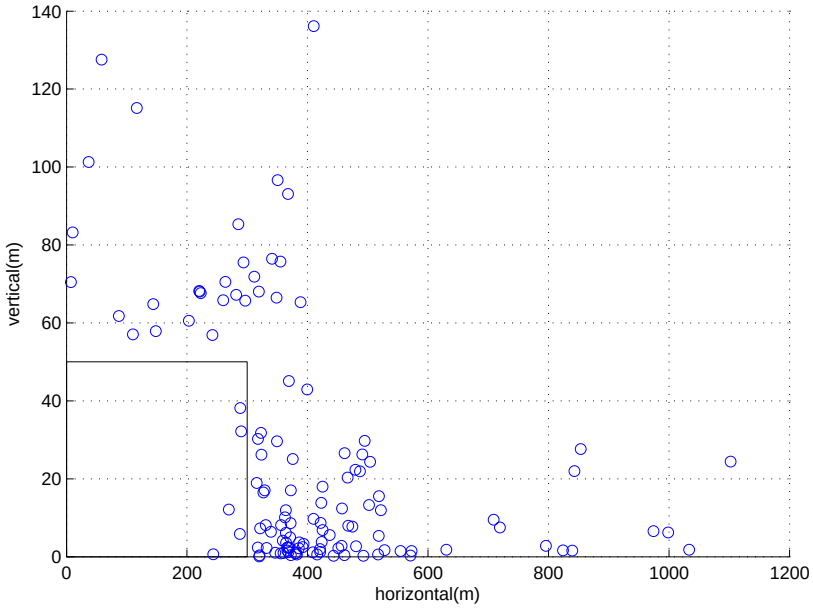


Figure 8.22: Scatter plot of horizontal and vertical distances when the UAV and aircraft were closest in all 120 encounters. Successful collision avoidance requires that a scatter point should not fall into the rectangle with 300m as its width and 50m as its height. It is observed only 5 encounters in total fall into the rectangle.

10 tests were performed with 1 to 3 aircraft respectively. Figure 8.22 shows both the horizontal and vertical distance between the UAV and each aircraft when they were closest to each other. The rectangle indicates the safety zone around the UAV. It is a cylinder centered at the UAV's position with bottom circle radius of 300 meters and height 100 meters. From the figure, we observe that only in 5 encounters the obstacle aircraft fall into the safety zone.

8.4.2 Outdoor Real Flight Experiment

We first describe the closed-loop system of the vehicle used in planning. Following that, we present the outdoor flight experiment results.

The Vehicle's Closed-Loop System

A waypoint command to the AR.Drone contains (x_g, y_g, z_g) position and speed v_c. With the position and speed as the input, the waypoint guiding controller outputs the yaw command, acceleration in $x - y$

plane and vertical speed. It is described as

$$\phi_c = atan2(y_g - y, x_g - x) \tag{8.19}$$

$$a = k_1(v_c - v) \tag{8.20}$$

$$v_z = k_2(z_g - z) \tag{8.21}$$

Where (x, y, z) is the vehicle's current position and v the current speed. k_1 and k_2 are different gains. With the heading ϕ_c, acceleration a, and vertical speed v_z as the input, the vehicle's model is

$$\dot{\phi} = k_3(\phi_c - \phi) \tag{8.22}$$

$$\dot{v} = a \tag{8.23}$$

$$\dot{x} = vcos\phi \tag{8.24}$$

$$\dot{y} = vsin\phi \tag{8.25}$$

$$\dot{z} = v_z \tag{8.26}$$

To validate the closed-loop system model, an AR.Drone was commanded to fly from (33.409969,-111.893976,1.54) to (33.410139,-111.893931,1.5) with the speed of 2 m/s. Figure 8.23 compares the trajectory data acquired from the real flight and simulated by the closed-loop system. Figure 8.23a, 8.23b, and 8.23c show the comparison in x,y,z components. Figure 8.23d plots the distance in $x-y$ component. It is observed from the figures that the $x - y$ error distance does not exceed 2m and z error stays less than 0.1 meter. Such modeling error is acceptable by the path planner and therefore the closed-loop system model can be used in planning.

Flight Tests

In the flight tests, the host UAV takes off from (33.409960,-111.893948) and it needs fly to the goal (33.4102431,-111.89398441). The obstacle UAV takes off from (33.410187,-111.893931) and flew to where the host UAV takes off. They both fly at speed 1 m/s and height 1.5 m. There will be head-on encounter between the host and obstacle UAV.

Figure 8.24 shows the trajectories of both host and obstacle UAVs in one of experiment flights. UTM coordinate was used in the figure. UAVs' positions were subtracted by the initial position of the host UAV. Each triangle has the same heading as the UAV at that moment. Triangles marked with the same number stand for the position and heading of two UAVs at the same time instance. The obstacle UAV flew from the north to the south and the host UAV flew in the opposite direction. The host UAV flew to

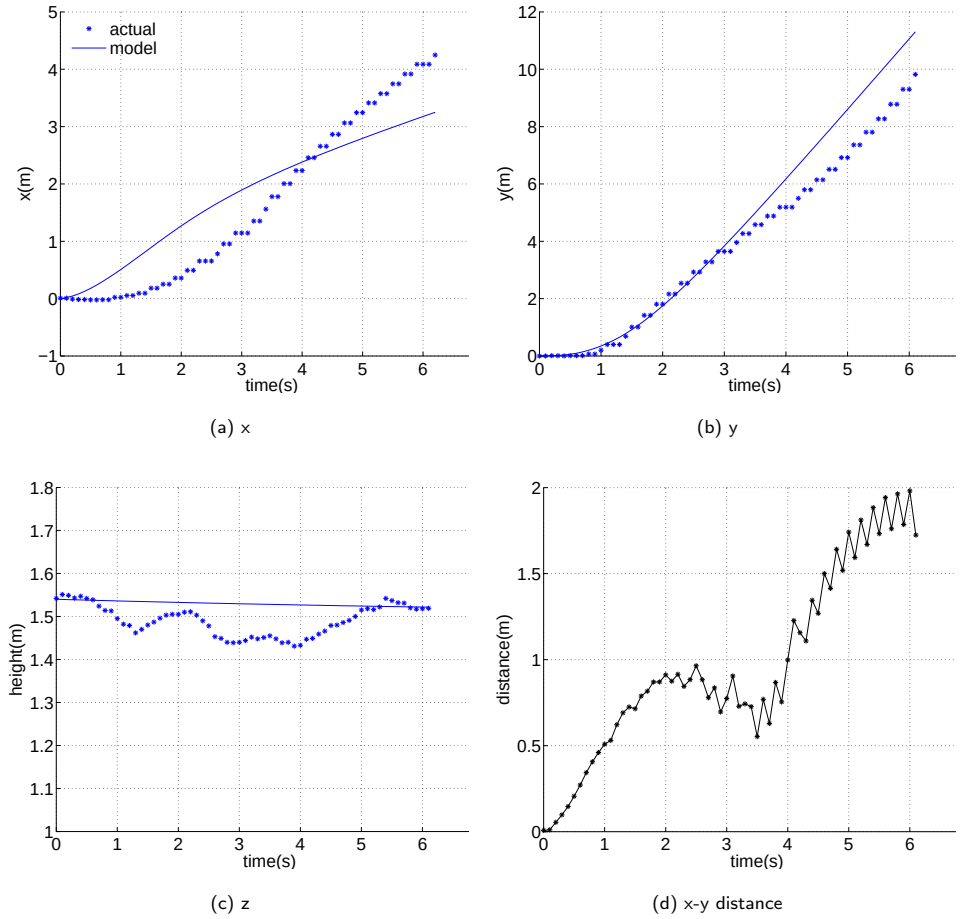


Figure 8.23: Comparison of trajectory predicted by the closed-loop system model (blue line) and actual flight trajectory (blue dots) of an AR.Drone flying outdoor. (a)-(c) compare their difference in x, y and z respectively. (d) plots their distance in the $x-y$ plane at each time instance. x and y were subtracted by the vehicle's initial x and y value.

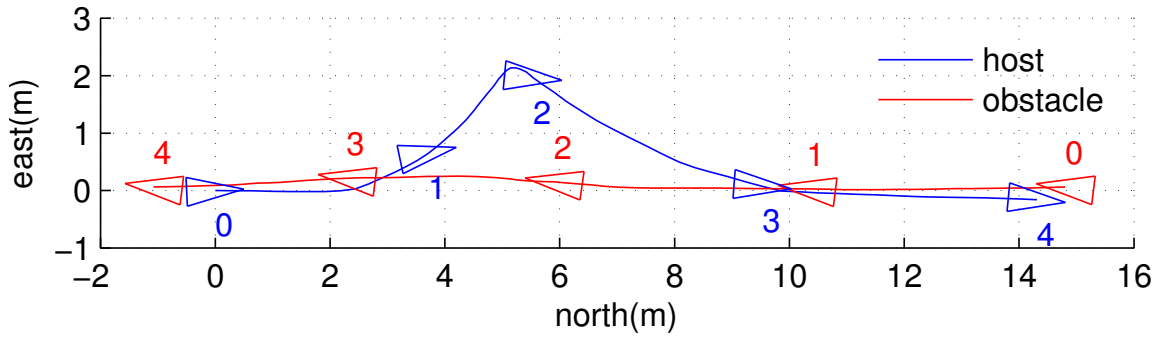


Figure 8.24: Trajectories of the host (blue) and obstacle (red) UAV in one of outdoor real flight experiments. The triangles stands for the UAV. The number marked the position of two UAVs at the same time instance.

the east to avoid the collision with the obstacle UAV. After it passes the obstacle UAV, it continues to its original goal.

(a) (b) (c)

Figure 8.25: Snapshots of one of outdoor flight experiments. East was on the right and north was in the direction going away from the camera. Initially, the host UAV flew to the north and the obstacle UAV to the south (Figure 8.25a). Then the host UAV flew to the east to avoid the obstacle UAV (Figure 8.25b. After passing the obstacle UAV, the host UAV returned to the original route and headed to the goal (Figure 8.25c).

Figure 8.25 are the snapshots of the flight experiment of Figure 8.24. The camera was facing to the north and the east was on the right. Initially, the host UAV flew to its goal to the north and the obstacle UAV to the south (Figure 8.25a). Upon the prediction of a collision, he host flew to the generated avoidance waypoint in the east to avoid collision with the obstacle UAV (Figure 8.25b). The host UAV returned to its original route and headed to its goal after passing the obstacle UAV (Figure 8.25c). Table 8.4 records the horizontal and vertical distance between the host and obstacle UAVs when they

flight	1	2	3	4	5	6	7	8	9	10
h_dis	2.365	2.14	4.431	6.119	3.523	3.296	3.43	3.25	7.757	3.83
v_dis	0.03	0.45	0.01	0.117	0.032	0.05	0.001	0.013	0.015	0.033

Table 8.4: The Horizontal and Vertical Distance Between the Host and Obstacle UAV When They Were Closest to Each Other from Ten Flight Tests.

were closest in each flight experiment. 10 flights in total were performed. The safety volume is a cylinder whose height of 1m and bottom circle radius 2 m. Therefore, the obstacle incurs the UAV's safety volume in none of the flight in table 8.4. They indicates successful collision avoidance.

Chapter 9

CONCLUSIONS

9.1 Summary

In this thesis we have developed and experimentally validated algorithms that allow an Unmanned Aerial Vehicle to autonomously avoid moving obstacles. The main contributions of this thesis are:

- An computer-vision based reactive avoidance method as the reactive motion planner to avoid a moving obstacle close to the UAV. The method uses the potential field to create avoidance actions.
- A global path planner based on Dubins curve to generate an optimal path for a fixed-wing UAV to avoid collisions.
- A Closed-Loop RRT based method that is able to generate collision avoidance paths in real time for a UAV. The path satisfies the UAV's model by construction.
- Extending the Closed-Loop RRT based method so that it can generate more candidate paths in the same amount of time and deal with obstacles' motion uncertainty.

The initial reactive method is a fast motion planner but it does not consider the vehicle's model and path optimality. The Dubins curve method creates path that is optimal in terms of length but it uses a simplified vehicle's model. The Closed-Loop RRT based method is able to deal with very complex non-linear model of the vehicle and create optimal paths. It can be improved to generate more candidate paths by using a greedy version of Closed-Loop RRT and exploiting intermediate points. By using a reachable set to represent a moving obstacle's motion uncertainty in collision check, the method can be improved to avoid obstacles that moves not in a straight line.

9.2 Future Directions

In the thesis, the path planning methods are at best validated by real flights using small quadrotors in a small area (less than 50m$\times$50m). The scale of the space where collision avoidance happens will be much larger than this. Therefore, it is still an interesting problem to investigate if the methods can be applied to real flights in a larger space in the air by real flight experiments.

This thesis focus on motion planning for moving obstacle avoidance. Detection and tracking moving obstacles using a sensor onboard on an UAV is still an open problem. The state-of-the-art works (Fu *et al.* (2014)) can only detect and tracking them in a camera image. To obtain 3D information of the obstacle using onboard sensors is challenging. MacLachlan (2005) developed a method to track moving objects using laser for ground vehicle. This could be applied to moving objects tracking for UAV as well. The integration of moving obstacle detection and tracking techniques and motion planning algorithms to avoid them is the final target of sense-and-avoid research. Only after achieving this, we can say that we build a sense-and-avoid system for UAVs.

REFERENCES

Andert, F., F. Adolf, L. Goormann and J. Dittrich, "Mapping and path planning in complex environments: An obstacle avoidance approach for an unmanned helicopter", in "Robotics and Automation (ICRA), 2011 IEEE International Conference on", pp. 745–750 (2011).

Aoude, G. S., B. D. Luders, J. M. Joseph, N. Roy and J. P. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns", Autonomous Robots **35**, 1, 51–76 (2013).

Bachrach, A., R. He and N. Roy, "Autonomous flight in unstructured and unknown indoor environments", in "in Proceedings of EMAV", (2009).

Beard, R. W. and T. W. McLain, "Implementing dubins airplane paths on fixed-wing uavs", Contributed Chapter to the Springer Handbook for Unmanned Aerial Vehicles (2013).

Bekris, K., B. Chen, A. Ladd, E. Plaku and L. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives", in "Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on", vol. 1, pp. 656–661 vol.1 (2003).

Biswas, J. and M. Veloso, "Depth camera based indoor mobile robot localization and navigation", in "Robotics and Automation (ICRA), 2012 IEEE International Conference on", pp. 1697–1702 (IEEE, 2012).

Borenstein, J. and Y. Koren, "Real-time obstacle avoidance for fast mobile robots", Systems, Man and Cybernetics, IEEE Transactions on **19**, 5, 1179–1187 (1989).

Borenstein, J. and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots", Robotics and Automation, IEEE Transactions on **7**, 3, 278–288 (1991).

Bristeau, P.-J., F. Callou, D. Vissière, N. Petit *et al.*, "The navigation and control technology inside the ar.drone micro uav", in "18th IFAC World Congress", pp. 1477–1484 (2011).

Carnie, R., R. Walker and P. Corke, "Image processing algorithms for uav "sense and avoid"", in "Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on", pp. 2848–2853 (2006).

Chakravarthy, A. and D. Ghose, "Obstacle avoidance in a dynamic environment: a collision cone approach", Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **28**, 5, 562–574 (1998).

Chang, F., C.-J. Chen and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique", computer vision and image understanding **93**, 2, 206–220 (2004).

Choset, H. M., *Principles of robot motion: theory, algorithms, and implementation* (MIT press, 2005).

Contarino, M. and L. Scire Consultants, "All weather sense and avoid system for uass", Report to the Office of Naval Research for R3 Engineering (2009).

Damas, B. and J. Santos-Victor, "Avoiding moving obstacles: the forbidden velocity map", in "Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on", pp. 4393–4398 (2009).

Dey, D., C. Geyer, S. Singh and M. Digioia, "Passive, long-range detection of aircraft: Towards a field deployable sense and avoid system", in "Field and Service Robotics", pp. 113–123 (Springer, 2010).

Dubins, L. E., "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents", American Journal of mathematics **79**, 3, 497–516 (1957).

Ebdon, M. D. and J. Regan, "Sense-and-avoid requirement for remotely operated aircraft (roa)", Air Combat Command, Langley Air Force Base, VA (2004).

FAA, "Ads-b broadcast services", http://www.faa.gov/nextgen/ads-b/broadcastservices/ (2010).

FAA, "Integration of civil unmanned aircraft systems (uas) in the national airspace system(nas) roadmap", http://www.faa.gov/about/initiatives/uas/media/UAS_Roadmap_2013.pdf (2013).

Fiorini, P. and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles", The International Journal of Robotics Research **17**, 7, 760–772 (1998).

Fox, D., W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance", IEEE Robotics & Automation Magazine **4**, 1, 23–33 (1997).

Fraichard, T. and H. Asama, "Inevitable collision states. a step towards safer robots?", in "Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on", vol. 1, pp. 388–393 vol.1 (2003).

Fu, C., A. Carrio, M. Olivares-Mendez, R. Suarez-Fernandez and P. Campoy, "Robust real-time vision-based aircraft tracking from unmanned aerial vehicles", in "Robotics and Automation (ICRA), 2014 IEEE International Conference on", (2014).

Fulgenzi, C., C. Tay, A. Spalanzani and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes", in "Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on", pp. 1056–1062 (IEEE, 2008a).

Fulgenzi, C., C. Tay, A. Spalanzani and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes", in "Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on", pp. 1056–1062 (2008b).

Gandhi, T., M.-T. Yang, R. Kasturi, O. Camps, L. Coraor and J. McCandless, "Detection of obstacles in the flight path of an aircraft", Aerospace and Electronic Systems, IEEE Transactions on **39**, 1, 176–191 (2003).

Geyer, M. S. and E. N. Johnson, "3d obstacle avoidance in adversarial environments for unmanned aerial vehicles", in "AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, CO", (2006).

Goldstein, H., "Classical mechanics", Reading: Addison-Wesley (1980).

Goodrich, M. A., "Potential fields tutorial", Class Notes (2002).

Gottschalk, S., M. C. Lin and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection", in "Proceedings of the 23rd annual conference on Computer graphics and interactive techniques", pp. 171–180 (ACM, 1996).

Green, W., P. Oh and G. Barrows, "Flying insect inspired vision for autonomous aerial robot maneuvers in near-earth environments", in "Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on", vol. 3, pp. 2347–2352 Vol.3 (2004).

Green, W. E. and P. Y. Oh, "Optic-flow-based collision avoidance", Robotics & Automation Magazine, IEEE **15**, 1, 96–103 (2008).

Green, W. E., P. Y. Oh, K. Sevcik and G. Barrows, "Autonomous landing for indoor flying robots using optic flow", in "in ASME International Mechanical Engineering Congress and Exposition", pp. 1347–1352 (2003).

Heng, L., L. Meier, P. Tanskanen, F. Fraundorfer and M. Pollefeys, "Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing", in "Robotics and automation (ICRA), 2011 IEEE international conference on", pp. 2472–2477 (IEEE, 2011).

Henry, P., M. Krainin, E. Herbst, X. Ren and D. Fox, "Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments", in "In the 12th International Symposium on Experimental Robotics (ISER", (Citeseer, 2010).

Hrabar, S., "3d path planning and stereo-based obstacle avoidance for rotorcraft uavs", in "Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on", pp. 807–814 (2008).

Hrabar, S., "Reactive obstacle avoidance for rotorcraft uavs", in "Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on", pp. 4967–4974 (IEEE, 2011).

Hrabar, S., "An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance", Journal of Field Robotics **29**, 2, 215–239 (2012).

Hrabar, S. and G. Sukhatme, "Vision-based navigation through urban canyons", Journal of Field Robotics **26**, 5, 431–452 (2009).

Hsu, D., *Randomized single-query motion planning in expansive spaces*, Ph.D. thesis, Stanford University (2000).

Hsu, D., R. Kindel, J.-C. Latombe and S. Rock, "Randomized kinodynamic motion planning with moving obstacles", The International Journal of Robotics Research **21**, 3, 233–255 (2002).

Hsu, D., J.-C. Latombe and R. Motwani, "Path planning in expansive configuration spaces", in "Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on", vol. 3, pp. 2719–2726 vol.3 (1997).

Jung, B. and G. S. Sukhatme, "Detecting moving objects using a single camera on a mobile robot in an outdoor environment", in "International Conference on Intelligent Autonomous Systems", pp. 980–987 (2004).

Kalman, R. E. *et al.*, "A new approach to linear filtering and prediction problems", Journal of basic Engineering **82**, 1, 35–45 (1960).

Karaman, S., M. R. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime motion planning using the rrt*", in "Robotics and Automation (ICRA), 2011 IEEE International Conference on", pp. 1478–1483 (IEEE, 2011).

Kavraki, L., P. Svestka, J.-C. Latombe and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", Robotics and Automation, IEEE Transactions on **12**, 4, 566–580 (1996).

Ko, N. Y. and R. Simmons, "The lane-curvature method for local obstacle avoidance", in "Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on", vol. 3, pp. 1615–1621 vol.3 (1998).

Kobilarov, M., "Cross-entropy motion planning", The International Journal of Robotics Research **31**, 7, 855–871 (2012).

Koenig, S. and M. Likhachev, "Incremental a*.", in "NIPS", pp. 1539–1546 (2001).

Koenig, S. and M. Likhachev, "D*lite", in "Eighteenth National Conference on Artificial Intelligence", pp. 476–483 (American Association for Artificial Intelligence, Menlo Park, CA, USA, 2002), URL `http://dl.acm.org/citation.cfm?id=777092.777167`.

Kuwata, Y., S. Karaman, J. Teo, E. Frazzoli, J. How and G. Fiore, "Real-time motion planning with applications to autonomous urban driving", Control Systems Technology, IEEE Transactions on **17**, 5, 1105–1118 (2009).

Kuwata, Y., J. Teo, S. Karaman, G. Fiore, E. Frazzoli and J. P. How, "Motion planning in complex environments using closed-loop prediction", in "Proc. AIAA Guidance, Navigation, and Control Conf. and Exhibit", (2008).

Lai, C.-K., M. Lone, P. Thomas, J. Whidborne and A. Cooke, "On-board trajectory generation for collision avoidance in unmanned aerial vehicles", in "Aerospace Conference, 2011 IEEE", pp. 1–14 (IEEE, 2011).

LaValle, S. M., *Planning algorithms* (Cambridge university press, 2006).

LaValle, S. M. and J. J. Kuffner, "Randomized kinodynamic planning", The International Journal of Robotics Research **20**, 5, 378–400 (2001).

Lieb, D., A. Lookingbill, D. Stavens and S. Thrun, "Tracking multiple moving objects from an autonomous helicopter", (2004).

Likhachev, M., D. Ferguson, G. Gordon, A. Stentz and S. Thrun, "Anytime search in dynamic graphs", Artif. Intell. **172**, 14, 1613–1643, URL `http://dx.doi.org/10.1016/j.artint.2007.11.009` (2008).

Lin, Y. and S. Saripalli, "Path planning using 3d dubins curve for unmanned aerial vehicles", in "Unmanned Aircraft Systems (ICUAS), 2014 International Conference on", pp. 296–304 (IEEE, 2014).

Lippiello, V. and F. Ruggiero, "3d monocular robotic ball catching with an iterative trajectory estimation refinement", in "Robotics and Automation (ICRA), 2012 IEEE International Conference on", pp. 3950–3955 (IEEE, 2012).

Liu, W. and I. Hwang, "Probabilistic aircraft midair conflict resolution using stochastic optimal control", Intelligent Transportation Systems, IEEE Transactions on **15**, 1, 37–46 (2014).

Lucas, B. D., T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision.", in "IJCAI", vol. 81, pp. 674–679 (1981).

Luders, B. D., G. S. Aoude, J. M. Joseph, N. Roy and J. P. How, "Probabilistically safe avoidance of dynamic obstacles with uncertain motion patterns", (2011).

MacLachlan, R., "Tracking moving objects from a moving vehicle using a laser scanner", Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-07 (2005).

McCandless, J. W., "Detection of aircraft in video sequences using a predictive optical flow algorithm", Optical Engineering **38**, 3, 523–530 (1999).

Mejias, L. and I. F. M. P. Campoy, "Omnidirectional bearing-only see-and-avoid for small aerial robots", in "Automation, Robotics and Applications (ICARA), 2011 5th International Conference on", pp. 23–28 (IEEE, 2011).

Melega, M., S. Lazarus and A. Savvaris, "Autonomous collision avoidance based on aircraft performances estimation", in "Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th", pp. 5B4–1 (IEEE, 2011).

Merz, T. and F. Kendoul, "Beyond visual range obstacle avoidance and infrastructure inspection by an autonomous helicopter", in "Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on", pp. 4953–4960 (2011).

Metni, N. and T. Hamel, "A uav for bridge inspection: Visual servoing control law with orientation limits", Automation in construction **17**, 1, 3–10 (2007).

Minguez, J. and L. Montano, "Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios", Robotics and Automation, IEEE Transactions on **20**, 1, 45–59 (2004).

Missiuro, P. and N. Roy, "Adapting probabilistic roadmaps to handle uncertain maps", in "Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on", pp. 1261–1267 (2006).

Owen, E. and L. Montano, "A robocentric motion planner for dynamic environments using the velocity space", in "Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on", pp. 4368–4374 (2006).

Owen, M., R. W. Beard and T. W. McLain, "Implementing dubins airplane paths on fixed-wing uavs*", Handbook of Unmanned Aerial Vehicles pp. 1677–1701 (2013).

Park, S., J. Deyst and J. P. How, "A new nonlinear guidance logic for trajectory tracking", in "AIAA guidance, navigation, and control conference and exhibit", pp. 1–16 (2004).

Patel, R. B., P. J. Goulart and V. Serghides, "Real-time trajectory generation for aircraft avoidance maneuvers", in "AIAA Guidance, Navigation and Control Conference, no. AIAA", vol. 5623 (2009).

Pettersson, P. O. and P. Doherty, "Probabilistic roadmap based path planning for an autonomous unmanned helicopter", Journal of Intelligent and Fuzzy Systems **17**, 4, 395–405 (2006).

Pivtoraiko, M., R. A. Knepper and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices", Journal of Field Robotics **26**, 3, 308–333 (2009).

Redding, J., J. N. Amin, J. D. Boskovic, Y. Kang, K. Hedrick, J. Howlett and S. Poll, "A real-time obstacle detection and reactive path planning system for autonomous small-scale helicopters", in "AIAA Navigation, Guidance and Control Conference, Hilton Head, SC", (2007).

Reif, J. H., "Complexity of the movers problem and generalizations extended abstract", in "Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science", pp. 421–427 (1979).

Richards, A. and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming", in "American Control Conference, 2002. Proceedings of the 2002", vol. 3, pp. 1936–1941 (IEEE, 2002).

Sampedro, C., C. Martinez, A. Chauhan and P. Campoy, "A supervised approach to electric tower detection and classification for power line inspection", in "Neural Networks (IJCNN), 2014 International Joint Conference on", pp. 1970–1977 (2014).

Saunders, J. B., B. Call, A. Curtis, R. W. Beard and T. W. McLain, "Static and dynamic obstacle avoidance in miniature air vehicles", AIAA Infotech@ Aerospace (2005).

Scherer, S., S. Singh, L. Chamberlain and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments", The International Journal of Robotics Research **27**, 5, 549–574 (2008).

Sekhavat, S., P. Svestka, J.-P. Laumond and M. H. Overmars, "Multilevel path planning for nonholonomic robots using semiholonomic subsystems", The international journal of robotics research **17**, 8, 840–857 (1998).

Shen, S., N. Michael and V. Kumar, "Autonomous indoor 3d exploration with a micro-aerial vehicle", in "Robotics and Automation (ICRA), 2012 IEEE International Conference on", pp. 9–15 (IEEE, 2012).

Shevchenko, A. M., "Energy-based approach for flight control systems design", Automation and Remote Control **74**, 3, 372–384 (2013).

Shi, J. and C. Tomasi, "Good features to track", in "Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on", pp. 593–600 (IEEE, 1994).

Shiller, Z., F. Large and S. Sekhavat, "Motion planning in dynamic environments: obstacles moving along arbitrary trajectories", in "Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on", vol. 4, pp. 3716–3721 vol.4 (2001).

Shim, D. H. and S. Sastry, "An evasive maneuvering algorithm for uavs in see-and-avoid situations", in "American Control Conference, 2007. ACC'07", pp. 3886–3891 (IEEE, 2007).

Shkel, A. M. and V. Lumelsky, "Classification of the dubins set", Robotics and Autonomous Systems **34**, 4, 179–202 (2001).

Simmons, R., "The curvature-velocity method for local obstacle avoidance", in "Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on", vol. 4, pp. 3375–3382 (IEEE, 1996).

Stentz, A., "Optimal and efficient path planning for partially-known environments", in "Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on", pp. 3310–3317 (IEEE, 1994).

Sugiyama, S., J. Yamada and T. Yoshikawa, "Path planning of a mobile robot for avoiding moving obstacles with improved velocity control by using the hydrodynamic potential", in "Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on", pp. 1421–1426 (2010).

Sutherland, I. E., R. F. Sproull and R. A. Schumacker, "A characterization of ten hidden-surface algorithms", ACM Computing Surveys (CSUR) **6**, 1, 1–55 (1974).

Templeton, T., D. H. Shim, C. Geyer and S. S. Sastry, "Autonomous vision-based landing and terrain mapping using an mpc-controlled unmanned rotorcraft", in "Robotics and Automation, 2007 IEEE International Conference on", pp. 1349–1356 (IEEE, 2007).

Thrun, S. *et al.*, "Robotic mapping: A survey", Exploring artificial intelligence in the new millennium pp. 1–35 (2002).

Tsenkov, P., J. K. Howlett, M. Whalley, G. Schulein, M. Takahashi, M. H. Rhinehart and B. Mettler, "A system for 3d autonomous rotorcraft navigation in urban environments", in "AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, HI", (2008).

Van Den Berg, J., D. Ferguson and J. Kuffner, "Anytime path planning and replanning in dynamic environments", in "Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on", pp. 2366–2371 (IEEE, 2006).

Wainwright, A. and J. Ford, "Fusion of morphological images for airborne target detection", in "Information Fusion (FUSION), 2012 15th International Conference on", pp. 1180–1187 (2012).

Warren, W., B. Fajen and D. Belcher, "Behavioral dynamics of steering, obstacle avoidance, and route selection", Journal of Vision **1**, 3, 184–184 (2001).

Watanabe, Y., A. J. Calise and E. N. Johnson, "Vision-based obstacle avoidance for uavs", in "AIAA Guidance, Navigation and Control Conference and Exhibit", pp. 20–23 (2007).

Watanabe, Y., A. J. Calise, E. N. Johnson and J. H. Evers, "Minimum-effort guidance for vision-based collision avoidance", in "AIAA Atmospheric Flight Mechanics Conference and Exhibit", pp. 21–24 (2006).

Weiss, B., M. Naderhirn and L. del Re, "Global real-time path planning for uavs in uncertain environment", in "Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE", pp. 2725–2730 (2006).

Wu, A. and J. P. How, "Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles", Autonomous robots **32**, 3, 227–242 (2012).

Yang, X., L. M. Alvarez and T. Bruggemann, "A 3d collision avoidance strategy for uavs in a non-cooperative environment", Journal of Intelligent & Robotic Systems **70**, 1-4, 315–327 (2013).

Zilberstein, S. and S. Russell, "Approximate reasoning using anytime algorithms", in "Imprecise and Approximate Computation", pp. 43–62 (Springer, 1995).

APPENDIX A

VIDEOS OF SOME EXPERIMENTS

In this appendix, videos link of some experiments are listed:

- Potential field based reactive avoidance methods:

  `https://youtu.be/ZOqQ0Qcgp0Y` and

  `https://youtu.be/UNERMzG1NfE`.

- Indoor flight experiments for Closed-Loop RRT based methods:

  `https://youtu.be/ggS5-x0rjNM`

- Software-In-the-Loop Simulation for greedy version of Closed-Loop RRT based methods:

  `https://youtu.be/bBjqc573cJM`

- Hardware-In-the-Loop Simulation for intermediate points based method:

  `https://youtu.be/y75JoHAbaqg`

- Outdoor real flight experiments for intermediate points based method:

  `https://youtu.be/n-pacvmoPZs`

APPENDIX B

ANALYSIS OF THE MOTION PLANNERS

In this appendix, we prove that the motion planning methods for moving obstacle avoidance in chapter 6 and chapter 7 are probabilistically complete. That means, the probability of not being able to find a path if one exists will approach zero when the number of samples goes asymptotically to infinity.

B.1 Expansive State×Time Space

Most of contents in this section are taken from Hsu *et al.* (2002) because the authors analyzed similar problem to ours.

Given two points (s,t) and (s',t') in free state×time space $\mathcal{F}$, (s',t') is defined as reachable from (s,t) if there exists an collision-free trajectory induced by the closed-loop system as in Fig. 6.1 from (s,t) to (s',t'). Let $\mathcal{R}(p)$ denote the set of points reachable from some point p; it is called the *reachable set* of p. For any subset $\mathcal{S} \subset \mathcal{F}$, the reachable sets of all points in $\mathcal{S}$:

$$\mathcal{R}(\mathcal{S}) = \bigcup_{p \in \mathcal{S}} \mathcal{R}(p) \tag{B.1}$$

The lookout of a set $\mathcal{S} \subset \mathcal{F}$ is defined as the subset of all points in $\mathcal{S}$ whose reachable set overlap

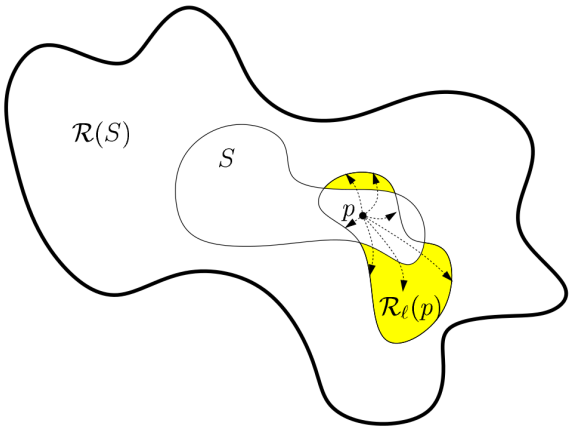


Figure B.1: The Lookout of a Set $\mathcal{S}$.

significantly with the reachable set of $\mathcal{S}$ that exclude $\mathcal{S}$. The idea is illustrated as the yellow area in Fig. B.1 (taken from Hsu *et al.* (2002)).

DEFINITION 1. Let β be a constant in (0,1]. The β-*lookout* of a set $\mathcal{S} \subset \mathcal{F}$ is

$$\beta - \text{LOOKOUT}(\mathcal{S}) = \{p \in \mathcal{S} | \mu(\mathcal{R}(p) \setminus \mathcal{S}) \geqslant \beta \mu(\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}\} \tag{B.2}$$

where $\mu(\mathcal{X})$ stands for the volume of a set $\mathcal{X} \subset \mathcal{F}$

DEFINITION 2. Let α and β be two constants in (0,1]. For any $p \in \mathcal{F}$, the set $\mathcal{R}(p)$ is $(\alpha, \beta) - expansive$ if for every connected subset $\mathcal{S} \subset \mathcal{R}(p)$,

$$\mu(\beta - \text{LOOKOUT}(\mathcal{S})) \geqslant \alpha\mu(\mathcal{S}) \tag{B.3}$$

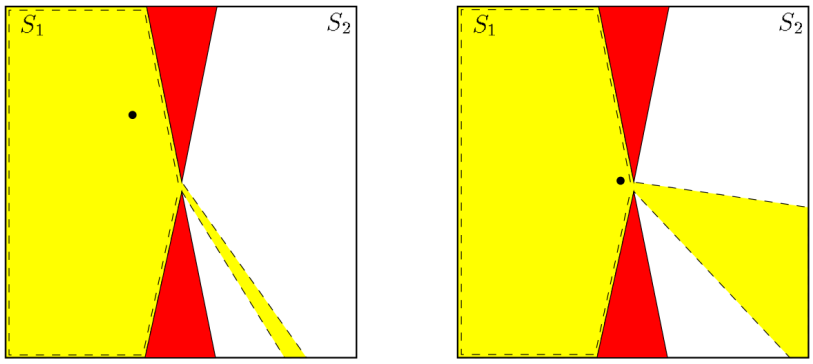


Figure B.2: An Example to Explain $\beta-$Outlook and $(\alpha, \beta) - Expansive$. The black point has a small β because it can only reach a small portion of S_2. However, a lot of points around it also have same reachable area. That gives a large α. The point on the right has a larger reachable area in S_2, but only points near the narrow passage have such a large reachable set. Therefore, the α value is small.

Fig B.2 (taken from Hsu (2000)) is used to explain the two definitions. On the left figure, the black point can only reach a small portion of S_2 due to the geometry constraint posed by the red obstacles. Therefore, it has a small β value in definition 1. However, a lot of other points has the reachable set of the same size. Therefore, the α value in definition 2 is large. The point on the right figure, on the contrary, is able to reach a larger portion of S_2. It corresponds to a larger β value, but only a small number of points close to the narrow passage can have such a large reachable area. Therefore, it has a smaller α value.

Suppose that $M = (m_0, m_1, m_2, ...)$ is a sequence of collision free milestones generated by sampling strategies in section 8.2.1 or 8.1.1. A milestone is a point $(s, t) \subset \mathcal{F}$. The state s is sampled and time t is determined in the procedure of trajectory simulation. Let M_i be the first i milestones in M. m_i is called a *lookout point* if it lies in the $\beta-$lookout of $\mathcal{R}(M_{i-1})$. It is proved in LEMMA 2 of Hsu *et al.* (2002) that:

A sequence of r milestones contains k lookout points with probability at least $1 - k(1-\alpha)^{r/k}$, where $k(1-\alpha)^{r/k}$ is probability that the whole sequence does not contain k lookout points. To facilitate the prove in the coming section, we define the reaching set of $p \in \mathcal{F}$ as the set of all points in $\mathcal{F}$ that can reach p. It is denoted by $\mathcal{RI}(p)$. It is defined reversely to a reachable set.

B.2 Proof of Probabilistic Completeness

Assuming that the state×time space where the UAV operates in is $(\alpha, \beta) - expansive$, we need to prove that we can find one milestone sequence $M = (m_0, m_1, m_2, .., m_N), N \geqslant 0$ that starts from the current UAV state (s_0, t_0) and ends in the goal (s_g, t_g). Here s_g is given and t_g is flexibale and can be determined by trajectory simulation. We assume that

$$\mu((s_g, t_g))/\mu(\mathcal{S}') = \gamma \tag{B.4}$$

where $\gamma \in (0, 1)$ and $\mathcal{S}'$ is the subset where we consider our problem.

The sequence M should satisfy three conditions to be a path:

1. m_0 lies in the the reachable set of (s_0, t_0). This makes sure that the sequence can be connected from the UAV's start state.

2. The part of the sequence that starts from m_1 must be lookout points. This makes sure that the sequence is not broken by any obstacle.

3. The last milestone m_N must be able to reach the goal. This makes sure that the sequence can finally reach the goal.

The probability that condition 1 and 2 are satisfied is at least $1 - (N + 2)(1 - \alpha)$. Here $(N + 2)(1 - \alpha)$ is the probability that any milestone is not a lookout point or m_0 does not lie in the reachable set of (s_0, t_0). The probability that condition 3 is statisfied is γ according to equation B.4. Therefore, the probability that a milestone sequence satisfy all three conditions is:

$$Pr(L) \geqslant \gamma[1 - (N + 2)(1 - \alpha)] \tag{B.5}$$

That is to say, the probability that a sequence fail to be a path is

$$\bar{Pr}(L) \leqslant 1 - \gamma[1 - (N + 2)(1 - \alpha)] \tag{B.6}$$

Since $1-\gamma[1-(N+2)(1-\alpha)]$ is a variable that stands for probability, we have $1-\gamma[1-(N+2)(1-\alpha)] \leqslant 1$. It is equivalent to

$$(1 - \alpha) \leqslant \frac{1}{N + 2} \tag{B.7}$$

Assuming from the tree structure, we can obtain n_1 sequences of 1 milestone, n_2 sequences of 2 milestones,... n_M sequences of M milestones. Substituting N by $M - 1$, we have

$$(1 - \alpha) \leqslant \frac{1}{M + 1} \tag{B.8}$$

The probability that all types of milestone sequence fail to qualify as paths are:

$$\begin{aligned} Pr(No) &\leqslant \{1-\gamma[1-2(1-\alpha)]\}^{n_1}\{1-\gamma[1-3(1-\alpha)]\}^{n_2}...\{1-\gamma[1-(M+1)(1-\alpha)]\}^{n_M} \\ &\leqslant [1-\gamma+\gamma\frac{2}{M+1}]^{n_1}[1-\gamma+\gamma\frac{3}{M+1}]^{n_2}...[1-\gamma+\gamma\frac{M}{M+1}]^{n_{M-1}} \\ &< (1-\gamma\frac{1}{M+1})^{N_m-1} \end{aligned} \tag{B.9}$$

where N_m is the total number of milestone sequences. It is a monotone increasing function of number sampling milestones. Therefore, when the number of sampling increases to infinity, N_m also increases to infinity. According to equation B.9, the probability that all milestone sequences fail to be a path approaches to zero when N_m increases to infinity. Therefore, we prove that a path connecting the start and goal lying in the free time$\times$space $\mathcal{F}$ can be found if one does exist. This proves the probabilistic completeness of the algorithm in chapter 6. The algorithms in chapter 7 are the special cases when $N = 3$ in the proof above.