AUTOMATIC COLLISION AVOIDANCE FOR MANUALLY TELE-OPERATED UNMANNED AERIAL VEHICLES

by

Jason R Israelsen

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

The University of Utah

August 2014

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Jason R Israelsen**

has been approved by the following supervisory committee members:

Jur van den Berg	, Chair	**03/21/14** Date Approved
Jake Abbott	, Member	**03/31/14** Date Approved
Stephen Mascaro	, Member	**03/31/14** Date Approved

and by **Tim Ameel** , Chair/Dean of

the Department/College/School of **Mechanical Engineering**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

This thesis provides details on the development of automatic collision avoidance for manually tele-operated unmanned aerial vehicles. We note that large portions of this work are also reprinted with permission, from 2014 IEEE International Conference on Robotics and Automation, "Automatic Collision Avoidance for Manually Tele-operated Unmanned Aerial Vehicles", by J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg © 2014 IEEE.

We provide a method to aid the operator of unmanned aerial vehicles. We do this by automatically performing collision avoidance with obstacles in the environment. Our method allows the operator to focus on the overall motion of the vehicle rather than requiring the operator to perform collision avoidance. Where other currently existing systems override the controls of the operator only as a last resort, our approach was developed such that the operator can *rely* on the automatic collision avoidance for maneuverability.

Given the current operator control input, our approach continually determines the future path of the vehicle. If along the future path a collision is predicted, then our algorithm will minimally override the operator's control such that the vehicle will not collide with the obstacles in the environment. Such an approach ensures the safety of the operator's controls while simultaneously maintaining the original intent of the operator. We successfully implemented this approach in a simulated environment, as well as on a physical quadrotor system in a laboratory environment. Our experiments show that, even when intentionally trying to do so, the operator failed to crash the vehicle into environment obstacles.

For my family and friends, particularly my wife, for their continual support through the many hours of work that were put into this research.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis would not have been possible without the support I received from my friends, family, coresearchers, and advisor. I would like to thank Dr. Jur van den Berg for his mentoring through the development of this work. I would like to recognize the work of the IARV (Indoor Aerial Reconnaissance Vehicle) Senior Design Group (consisting of: Dan Stuart, Eric Keeney, Michael Nuttal, Ben Jones, and Randy Neumann), in the development of the simulator and motivation behind our method. I would like to recognize the work of Matt Beall in the further development of the algorithm, creation of the environment for experiments, and recording of experiment results. I would also like to recognize the work of Daman Bareiss in the implementation of the algorithm in ROS (Robot Operating Interface), recording of experiment results, and development of the final algorithm.

CHAPTER 1

INTRODUCTION[1]

1.1 Motivation

The use of unmanned aerial vehicles (UAVs) has dramatically increased in recent years. They have been used in applications such as search and rescue, cinematography, monitoring, policing, entertainment, and exploration of hazardous and/or normally inaccessible locations [4], [5], [6], [7]. Perhaps the most popular recent movement has been the proposal of the use of these vehicles in delivering packages commercially [8]. As the capabilities of UAVs continue to increase, more applications for these types of vehicles open up [9], including their use in indoor and outdoor locations.

One of the challenges with using UAVs is avoiding collisions with obstacles and the environment. Consider a situation where the operator maneuvers a UAV, equipped with cameras, through a building at risk of collapse in search for survivors after a natural disaster, such as an earthquake or fire, or the case where an area has had nuclear, toxic, or other hazardous material released. The operator can use the UAV to observe conditions, or even help to remedy the situation. In these cases the operator has a need to make high level decisions (e.g., where to fly the vehicle, what areas need to be observed more closely, etc.). Even though the operator may be capable of performing such tasks through training, maneuvering UAVs can be difficult. The problem of maneuverability is compounded when feedback to the operator is limited, particularly in environments that do not allow for GPS and where navigation is based off of video feed.

We center our efforts on aiding the tele-operator of UAVs in environments that can be difficult to maneuver in without collisions. The motivation of this thesis is to present an approach that lets the vehicle automatically perform collision avoidance so that the operator of the UAV can focus all of his/her efforts on performing high-level tasks. While operator

[1]Parts of this chapter are reprinted with permission, from 2014 IEEE International Conference on Robotics and Automation, "Automatic Collision Avoidance for Manually Tele-operated Unmanned Aerial Vehicles," by J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg © 2014 IEEE

driven collision avoidance systems exist, such as those found in modern automobiles, these systems typically warn the operator or override operator commands as a last resort [10], [11], [12], [13]. The approach we present is focused on allowing the operator to rely on the automatic collision avoidance system. What we mean by this is that, while our system does ensure that collisions are avoided, we do so while maintaining as much of the original operator command as possible. Our approach thus allows the operator to use the environment and obstacles to help navigate (instead of having the operator attempt to avoid collisions with them). We follow this method in an effort to provide the most intuitive interface for the operator.

Our approach is similar to the concept of *virtual fixtures* that are commonly used in surgical robotics [14]. Virtual fixtures are virtual forces that act on control inputs to prevent operator paths into forbidden regions or to help drive an operator's path to a desired path. We want to obtain similar results to those found in virtual fixtures, but applied to a tele-operated unmanned aerial vehicle. Virtual fixtures do not use the dynamics of the system to make decisions, but due to the complications of the highly inertial dynamics of our vehicle, we take a different approach from virtual fixtures to specifically account for the properties of our system.

While a more detailed description of our method will be presented in Section 3, generally its characteristics can be stated with the following:

1. It continually estimates the future trajectory of the vehicle.
2. It checks if a collision will occur.
   (a) If a collision will occur, it minimally changes the operator inputs such that a collision will not occur.
   (b) If a collision will not occur, it does not change the operator input.

We note that our method is designed to operate in conditions in which the vehicle detects relative locations of the environment and nearby obstacles using sensors in real-time. Since our focus is on collision avoidance, we assume that the robot knows the local environment and relative position within. In implementation this information is determined through a motion capture system. We verified our method in an operator-driven simulation and validated our method on a physical quadrotor helicopter system.

1.2 Research Objectives

Our high-level research objectives are as follows:

1. Develop a simulation environment where we can give live operator control inputs to verify our method.

2. Validate our method on hardware contained within the Algorithmic Robotics Lab.

1.3 Thesis Organization

The work we present in this thesis is structured as follows.

Chapter 2 is a literature review where we present related concepts and work. We provide an introduction to UAVs, particularly quadrotors. We present some of the basic concepts and work done with both virtual fixtures and collision avoidance, to provide the reader with similarities and differences between these methods and our method.

Chapter 3 introduces the approach of our algorithm. We define our problem mathematically. Then we present our mathematical approach to solve our problem. We also present assumptions that are made so that our system is more readily solvable (e.g., about the geometry and future trajectory of the robot).

Chapter 4 presents implementation details and results of our experiments. This chapter provides verification and validation of our algorithm. We present finer details of how we implemented our method. We also present a sample of simulation and implementation results.

Chapter 5 presents the conclusions of this thesis work and recommendations for future work.

CHAPTER 2

LITERATURE REVIEW[1]

In this chapter we present a background on related concepts, a comparison of other work, and a foundation for our thesis work. First, we provide an introduction into our system of interest, UAVs. Second, we present concepts and related work of virtual fixtures. Lastly, we present concepts and related work of collision avoidance.

2.1 Quadrotor Helicopters

Quadrotor helicopters are aerial vehicles, which are used in various applications. They have been used for performing search and rescue operations, obtaining film for cinematography, monitoring environments, assisting police work, providing entertainment for hobbyists, exploring hazardous and/or normally inaccessible locations, and delivering commercial packages [4], [5], [6], [7], [8].

Quadrotors can be described as having four propellers in a cross configuration [9] (similar in fashion to that shown in Fig. 2.1). The adjacent propellers are configured for counter rotation to provide torque stability to the system. Through various combinations of forces from each propeller, different movements can be achieved. Allowable control movements may vary from vehicle to vehicle depending on the control configuration. A common control configuration (which we used) provides control of thrust (translation in Z-axis), roll (tilt along X-axis leading to translation in Y axis), pitch (tilt along Y-axis leading to translation in X axis), and yaw (rotation about the Z-axis).

Quadrotors have become popular due to the many advantages they have compared to other modes of flight, including the following: mechanic simplifications, high payloads, easy control, indoor usage [9], the ability to hover, and the capability for vertical takeoff and landing [15]. Even with these advantages, quadrotors still pose challenges such as

[1]Parts of this chapter are reprinted with permission, from 2014 IEEE International Conference on Robotics and Automation, "Automatic Collision Avoidance for Manually Tele-operated Unmanned Aerial Vehicles," by J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg

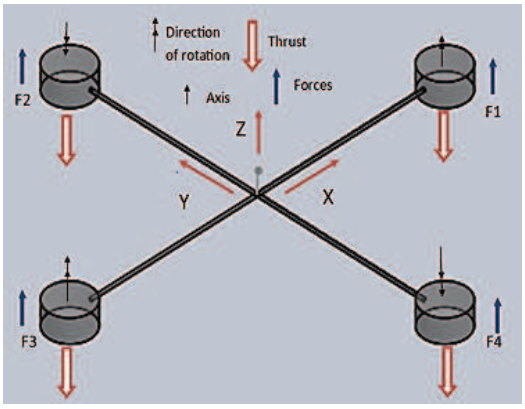

Figure 2.1. General schematic of quadrotor helicopter. Torque on vehicle is balanced through the use of counter-rotating propellers. Different movements are achieved by changing the forces produced from each propeller.

under-actuation and dynamic instablity of the system [9]. These challenges continue to drive research on quadrotor control. Some of the existing collision control techniques created to aid in the control and maneuverability of quadrotors will be discussed in Section 2.3.

2.2 Virtual Fixtures

Virtual fixtures, which are virtual forces that act on control inputs, are used to aid remote operators. They have commonly been used in surgical robotics. The intent of virtual fixtures (also known as "virtual mechanisms," "virtual tools," "virtual paths and surfaces," and "haptically augmented teleoperation" [1]) is to reduce the stress and processing required of an operator to perform a task remotely [14]. The reason for doing this is to help operators achieve tasks that can require precision and accuracy beyond that which human operators can normally handle.

Virtual fixtures can generally be lumped into two categories (see Fig. 2.2) [1]:

1. A guidance virtual fixture, which aids the operator by guiding the robot end effector along a desired path.

2. A forbidden-region virtual fixture, which aids the operator by keeping the robot end effector out of a forbidden configuration.

The end effector of the robot is the device interacting with the environment, like the end of a robotic arm. In the case of forbidden-region virtual fixtures (which are most similar to our method) the end effector of the robot is kept from penetrating the defined fixtures by projecting the range of the robot's Jacobian matrix onto a subspace parallel to

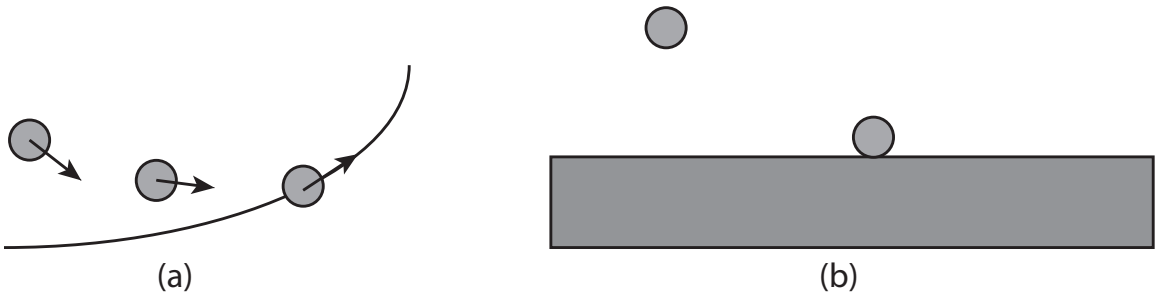

Figure 2.2. Depiction of virtual fixtures. (**a**) Example of a guidance virtual fixture, which assists the operator in guiding the robot along a desired path. (**b**) Example of a Forbidden-region virtual fixture, which assists the operator in keeping the robot out of a forbidden configuration. Modified from [1].

the fixtures [16]. This effect retains the motion parallel to the fixture while also changing the perpendicular component of the control input directed into the fixture, such that the end-effector does not penetrate the fixture (thus avoiding unwanted positions). In the case of guidance virtual fixtures the necessary control inputs are changed similarly to that performed on forbidden-region virtual fixtures, but to guide the operator along a desired path. In many cases virtual fixtures have been used in conjunction with haptic feedback [17],[1].

Virtual fixtures have been used in a wide array of applications. They have commonly been used in surgical robotics to steady the hands and paths of the operators using the system, in particular: Marayong [16] proposed the use of virtual fixtures to improve speed and precision of medical interventions, Bettini [17] used virtual fixture methods so that the operator could work in the millimeter to micrometer scales for applications such as vitreoretinal surgery, Dewan [18] used virtual fixtures to aid in retinal microsurgery [1], and Gibo [19] proposed methods for forbidden virtual-fixtures that move so that work could be performed on moving organs.

Virtual fixtures use the kinematics of the robot to ensure forbidden regions are avoided. Due to our interest in the application of automatic collision avoidance for an operator-driven quadrotor helicopter (a highly inertial dynamic system), we take a different approach from the methods used in virtual fixtures to account for the specific properties of our system.

2.3 Collision Avoidance

The problem of collision avoidance has a long history in robotics. There are a myriad of methods that have been used to achieve collision avoidance; we make note of some of

them here. One set of methods predicts and avoids collisions by considering reachable admissible velocities within a short time interval [20], [21]. Another set uses the velocity space to avoid static and moving obstacles [22], [23], [24]. There are also methods that use an artifical potential field to drive a robot from a beginning configuration, represented as a high potential, to a goal configuration, represented as a low potential, while also driving the robot away from obstacles [25], [26]. Then there are some methods that perform collision avoidance by defining and determining states for which a collision with an obstacle will eventually occur, no matter the future trajectory [27]. Since our primary focus is the implementation of collision avoidance with quadrotor systems, we center our attention on methods that apply to these systems.

2.3.1 Autonomous Collision Avoidance for Quadrotors

Collision avoidance for autonomous quadrotors has been a well explored field of research. Roberts et al. [4] investigated methods for autonomous control of MAVs (Minature Aerial Vehicles) which maneuver in indoor environments. This research used a PD (proportional-derivative) controller to provide collision avoidance in the environment. For implementation, sensor data were retrieved from on-board gyroscopes and cameras, and experiments were limited to environments without obstacles. Nieuwenhuisen et al. [28] introduced methods that used multimodal obstacle detection and multimodal collision avoidance. In this setup omnidirectional obstacle perception was used and applied to a MAV system. Ducard et al. [29] also performed research on controlling indoor quadrotor vehicles, but focused on the use of motion capture systems. This work included implementation on environments containing single and multiple quadrotors.

Similarly, Grzonka et al. [30] and He et al. [5] focused on approaches for autonomous control of indoor quadrotors. These approaches centered on systems that used on-board sensing to determine the environment. Grzonka et al. introduced the use of SLAM (simultaneous localization and mapping) for collision avoidance. He et al. used BRM (belief roadmaps, which are an extension of probalistic roadmaps) to plan trajectories through the environment.

Mellinger et al. [31] presented a collision avoidance method for environments both with obstacles and where teams of quadrotors were present. This research introduced the use of a method that used integer constraints of MIQPs (mixed-integer quadratic programs) to enforce collision avoidance and required the knowledge of the start and end positions of all vehicles used. The method was capable of operating on teams of up to four quadrotors. The research done by Bareiss and van den Berg [32], also presented a collision avoidance

approach for systems consisting of multiple robots. However, this approach introduced the use of reciprocal collision avoidance (an algorithm where robots share the responsibility for collision avoidance) along with LQR (linear quadratic regulator) obstacles as its foundation. This was applied to vehicles observing each others' states and independently selecting their control inputs. This approach was performed in simulation with large quantities of vehicles (some including systems with 100 quadrotors).

Autonomous collision avoidance provides many of the results we want for our system (e.g., collision avoidance with the environment, obstacles, indoor environments, etc.). However, we are unable to use the concepts and ideas presented with these methods because, by definition, autonomous systems make decisions independently (without operator input). Since our interest is in systems that are operator-driven, we diverge from the methods presented for autonomous collision avoidance.

2.3.2 Automatic Collision Avoidance for Operator Driven Quadrotors

Our main focus is on the use of collision avoidance for operator driven quadrotor systems. Interestingly, the research in this area is limited. We present a few of the existing studies.

One of the difficulties in controlling a vehicle remotely is conveying to the operator what the environment is like via cameras. Brandt et al. [33] introduced the use of haptic feedback to aid the operator in avoiding collisions. This is done to convey information about the environment to the operator not only through cameras, but also through haptic feedback. There are three different methods presented in this study to achieve haptic feedback, using what is referred to as TTI (time-to-impact), DPF (dynamic parametric field), and VS (virtual spring). TTI is used to provide greater haptic feedback as a result of smaller times to impact. DPF is used to provide different types of behavior and haptic feedback when in different "zones" (or distances) from a hazard. VS is used to provide haptic feedback based on the distance from the hazard. This study found the use of TTI to be the most beneficial of the three methods and that TTI assisted the user in avoiding collisions.

Hou et al. [34] built upon the findings of Brandt et al., introducing a method that uses dynamic kinesthetic boundaries (DKBs) to help the operator move through a complex environment. This method is used to provide haptic feedback to the operator to warn of obstacles. The DKB is derived from the velocity of the vehicle and the distance from the vehicle to the obstacle. Results showed that with this method, when the system provides perfect velocity tracking, the vehicle will not collide with obstacles. The method was

successfully verified on a robotic system.

Mendes [6] introduced a method that uses FastSLAM and a danger assessment algorithm. FastSLAM is an efficient form of SLAM created to provide information for complicated environments. The danger assessment algorithm uses a classifier to override the operator inputs, if danger is predicted. This classifier is largely based on the TTC (time to collision) of the vehicle with the environment. The TTC was calculated from the estimated current velocity of the vehicle and distance to the obstacle. Based on the TTC value, different responses of collision avoidance were used (e.g., No Action, Slow, Stop, and Evasive Maneuver). This method relied on the operator's definition of when to apply each response and was successfully applied in simulation.

The previous methods for collision avoidance for operator driven quadrotors based their collision prediction on the following:

1. The current velocity of the vehicle.
2. The distance of the vehicle from the obstacle.

For collision avoidance these previous methods did not necessarily provide an intuitive interface for the operator. In this thesis we take a slightly more principled approach, during collision detection and avoidance we specifically take into account the following:

1. The actual dynamics of the vehicle.
2. The current state of the vehicle.
3. The current operator control input.

We use this information to determine the future position (and thus detect future collision) of the vehicle along potentially nonlinear trajectories. If a collision is detected we *minimize* the deviation from the operator input such that collisions will not occur and provide an intuitive response for the operator.

CHAPTER 3

PROBLEM DEFINITION AND APPROACH[1]

In this chapter we introduce our problem definition and approach. We begin by mathematically defining our problem in Section 3.1. Then in Section 3.2, we introduce assumptions made to make our problem solvable. Finally, we define the approach we use in Section 3.3.

3.1 Problem Definition

We use the following notation conventions. We represent matrices with upper-case italics, X. We represent scalars with lower-case italics, x. Vector sets are represented by calligraphics, $\mathcal{X}$. Vectors are represented by boldface, $\mathbf{x}$. The scalar and matrix multiplication are defined by:

$$a\mathcal{X} = \{a\mathbf{x} \mid \mathbf{x} \in \mathcal{X}\}, \quad A\mathcal{X} = \{A\mathbf{x} \mid \mathbf{x} \in \mathcal{X}\} \tag{3.1}$$

We then consider a robot with general nonlinear dynamics and with a state space of arbitrary dimension. Let $\mathcal{X} \subset \mathbb{R}^n$ be the state space of the robot, and let $\mathcal{U} \subset \mathbb{R}^m$ be the control input space of the robot. Let the continuous-time, nonlinear dynamics of the robot be defined by a function $\mathbf{f} \in \mathcal{X} \times \mathcal{U} \times \mathbb{R} \to \mathbb{R}^n$:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{3.2}$$

where t is time, and $\mathbf{u}(t)$ and $\mathbf{x}(t)$ are the operator input and state at time t. Then, given a constant operator input $\mathbf{u}$ along with the current state $\mathbf{x} = \mathbf{x}(0)$ the state of the robot for $t \geq 0$ is defined by:

$$\mathbf{x}(t) = \mathbf{g}(\mathbf{x}, \mathbf{u}, t) \tag{3.3}$$

where $\mathbf{g} \in \mathcal{X} \times \mathcal{U} \times \mathbb{R} \to \mathcal{X}$ represents the solution to differential Eq. (3.2).

---

[1]Parts of this chapter are reprinted with permission, from 2014 IEEE International Conference on Robotics and Automation, "Automatic Collision Avoidance for Manually Tele-operated Unmanned Aerial Vehicles," by J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg

We define the workspace in which the robot maneuvers as $\mathbb{R}^d$, where typically $d = 3$. We define the subset of the workspace occupied by obstacles as $\mathcal{O} \subset \mathbb{R}^d$. Also, in order to stay consistent with the case of on-board sensing, we let $\mathcal{O}$ define the regions of the workspace that are obstructed by the obstacles as seen from the current perspective of the robot (please refer to Fig. 3.1). Let $\mathcal{R}(\mathbf{x}) \subset \mathbb{R}^d$ denote the subset of the workspace occupied by the robot when it is in state $\mathbf{x} \in \mathcal{X}$.

The problem we discuss in this thesis is defined as finding a minimal change $\Delta\mathbf{u} \in \mathcal{U}$ to the operator's control input $\mathbf{u} \in \mathcal{U}$, given the current state $\mathbf{x} \in \mathcal{X}$ of the robot, such that for all times up to a preset time horizon $\tau \in \mathbb{R}$ the robot does not collide with obstacles:

$$\begin{aligned} &\text{minimize:} && \Delta\mathbf{u}^T R \Delta\mathbf{u} \\ &\text{subject to:} && \forall t \in [0, \tau] :: \mathcal{R}(\mathbf{g}(\mathbf{x}, \mathbf{u} + \Delta\mathbf{u}, t)) \cap \mathcal{O} = \emptyset, \end{aligned} \tag{3.4}$$

where $R \in \mathbb{R}^{m \times m}$ is a positive-definite weight matrix.

3.2 Simplifying Assumptions

The optimization problem as defined by Eq. (3.4) is highly nonlinear and nonconvex. Due to the complexities of this and the necessity to solve in real-time, we made some simplifications to make the problem more readily solvable.

We first make the assumption that the position of the robot, $\mathbf{p} \in \mathbb{R}^d$, can be obtained from the robot's current state $\mathbf{x} \in \mathcal{X}$. This can be done through a potentially, but not necessarily, nonlinear projection function $\mathbf{c} \in \mathcal{X} \to \mathbb{R}^d$.

$$\mathbf{p} = \mathbf{c}(\mathbf{x}), \tag{3.5}$$

We also make the assumption that the geometry $\mathcal{R}$ of the robot can be defined as the robot's smallest enclosing sphere with radius r centered at its reference point. This assumption makes the robot's geometry such that it is invariant to orientation (since it is a sphere). Let us denote the subset of the workspace occupied by the robot at its position $\mathbf{p}$ as $\mathcal{R}(\mathbf{p}) \subset \mathbb{R}^d$.

Next, given a change to the operator's control input $\Delta\mathbf{u} \in \mathcal{U}$, and the current state of the robot $\mathbf{x} \in \mathcal{X}$, we let the position of the robot be approximated by a first-order Taylor expansion:

$$\mathbf{p}(t, \Delta\mathbf{u}) \approx \hat{\mathbf{p}}(t) + G(t)\Delta\mathbf{u} \tag{3.6}$$

where

$$\hat{\mathbf{p}}(t) = \mathbf{c}(\mathbf{g}(\mathbf{x}, \mathbf{u}, t)), \qquad G(t) = \frac{\partial(\mathbf{c} \circ \mathbf{g})}{\partial \mathbf{u}}(\mathbf{x}, \mathbf{u}, t). \tag{3.7}$$

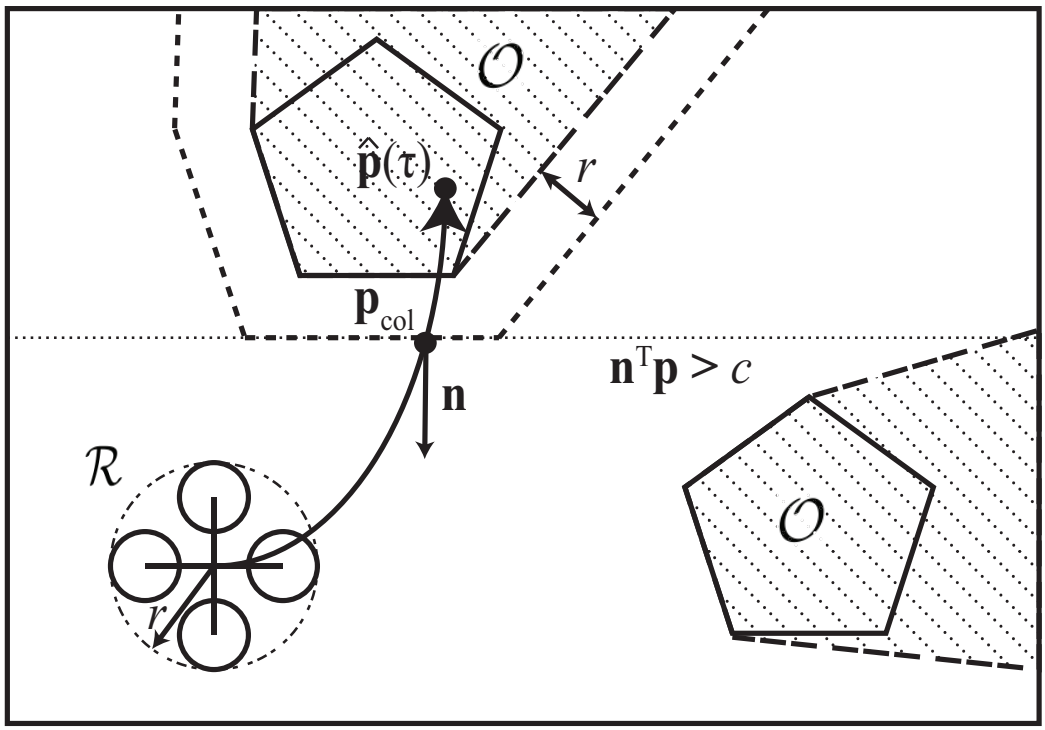

Figure 3.1. A schematic depicting our general approach. Given the current operator input and the current state of the robot, the future trajectory of the robot is extrapolated. This extrapolated trajectory is checked against the obstacles $\mathcal{O}$ (which includes the "shadows" of the obstacles that cannot be seen from the perspective of the robot) for collisions. If a collision is predicted within the specified time horizon τ, we define a halfspace that is tangent to the point of collision. This halfspace creates a constraint for the change to the control input $\Delta\mathbf{u}$.

We then make an assumption about the local linearity of the robots future trajectory for all $t \in [0, \tau]$. We state that if the robot is deemed free of collisions within the specified time horizon τ relative to $(\mathbb{R}^d \setminus \mathcal{O})$, then it is free of collisions for all time from $t \in [0, \tau]$. We note that this assumption is reasonable as long as the specified time horizon τ is not too large, and as long as the convex subset of the free workspace includes the current position of the robot.

3.3 Approach

This section presents our method of determining a safe control-input for general non-linear systems using convex optimization. Within this section we present an approach for defining linear constraints to our system. These constraints are used to find a new and safe future position, while simultaneously minimizing the change to the operators input.

3.3.1 Definition of Linear Constraints

Given a current operator input $\mathbf{u} \in \mathcal{U}$ and the current state $\mathbf{x} \in \mathcal{X}$, the position of the robot can be calculated at time τ in the future. If a collision is not predicted ($\forall t \in [0, \tau] :: \mathcal{R}(\hat{\mathbf{p}}(t)) \cap \mathcal{O} = \emptyset$) then there is no change to the operator control inputs. Otherwise, the inputs will be overwritten with our approach. The intersection between the potentially nonlinear path and the obstacle determines where collisions will occur (see Fig. 3.1). If a collision is encountered, we define $\mathbf{p}_{col}$ as the first position at which the robot is in collision with the obstacles(s) in $\mathcal{O}$:

$$\mathbf{p}_{col} = \hat{\mathbf{p}}(\min\{t \in [0, \tau] \mid \mathcal{R}(\hat{\mathbf{p}}(t)) \cap \mathcal{O} \neq \emptyset\}) \tag{3.8}$$

Let $\mathbf{n}$ be the unit normal vector of $\mathcal{O}$ pointing inward to the free workspace at $\mathbf{p}_{col}$ (see Fig. 3.1). We then define the linear constraint on the position of the robot at the time-horizon τ:

$$\mathbf{n}^T \mathbf{p}(\tau, \Delta\mathbf{u}) > c, \tag{3.9}$$

where $c = \mathbf{n}^T \mathbf{p}_{col}$. Eq. (3.9) defines a halfspace that gives a rough convex approximation of the local free space. By substituting our position from Eq. (3.6) into our constraint in Eq. (3.9) we transform this constraint into the control space:

$$\mathbf{a}^T \Delta\mathbf{u} > b, \tag{3.10}$$

where $\mathbf{a}^T = \mathbf{n}^T G(\tau)$ and $b = c - \mathbf{n}^T \hat{\mathbf{p}}(\tau)$. By replacing the complex constraint defined in Eq. (3.4) with the simple constraint defined in Eq. (3.10), we obtain the closed-form solution for the minimal change to the user input:

$$\Delta\mathbf{u} = bR^{-1}\mathbf{a}/(\mathbf{a}^T R^{-1}\mathbf{a}). \tag{3.11}$$

3.3.2 Iteration through Constraints

We note that initially, our newly found control input $\mathbf{u} + \Delta\mathbf{u}$, will not guarantee the absence of collisions along the future trajectory with respect to every obstacle in $\mathcal{O}$ and for all $t \in [0, \tau]$. This is particularly the case when the vehicle is attempting to maneuver into an edge or corner of the freespace (resulting in multiple collisions). An example of this case is depicted in Fig. 3.2. To solve this issue we iterate through our approach presented in Section 3.3.1.

We let the change in the control input from Eq. (3.11) be denoted as iteration 1 of our algorithm $\Delta\mathbf{u}_1$. Then for each continuing iteration i we extrapolate the new trajectory (based on $\mathbf{u} + \Delta\mathbf{u}_i$). Once this trajectory is created we can recheck the path for possible collisions. This defines a linear constraint $\mathbf{a}_i^T \Delta\mathbf{u} > b_i$ on the change in control input (see Eq. (3.10)). We note that this allows us to solve an optimization problem with respect to the number of i constraints:

$$\begin{aligned} &\text{minimize:} && \Delta\mathbf{u}^T R \Delta\mathbf{u} \\ &\text{subject to:} && \textstyle\bigcap_{j=1}^{i} \{\mathbf{a}_j^T \Delta\mathbf{u} > b_j\} \end{aligned} \tag{3.12}$$

For the optimization problem we let the number of iterations/constraints be maximized by the total dimension of the workspace. This is done to allow for corners of the free space. We repeat the iterative approach presented here for every control cycle. In this way, our approach continually adapts to the operator's control input and ensures that collisions are avoided.

It is important to note that the iterative inclusion of constraints in this optimization problem matches up well with the LP-type algorithm presented by Berg et al. [35]. Such an approach is used to solve low-dimensional convex optimization problems quickly.

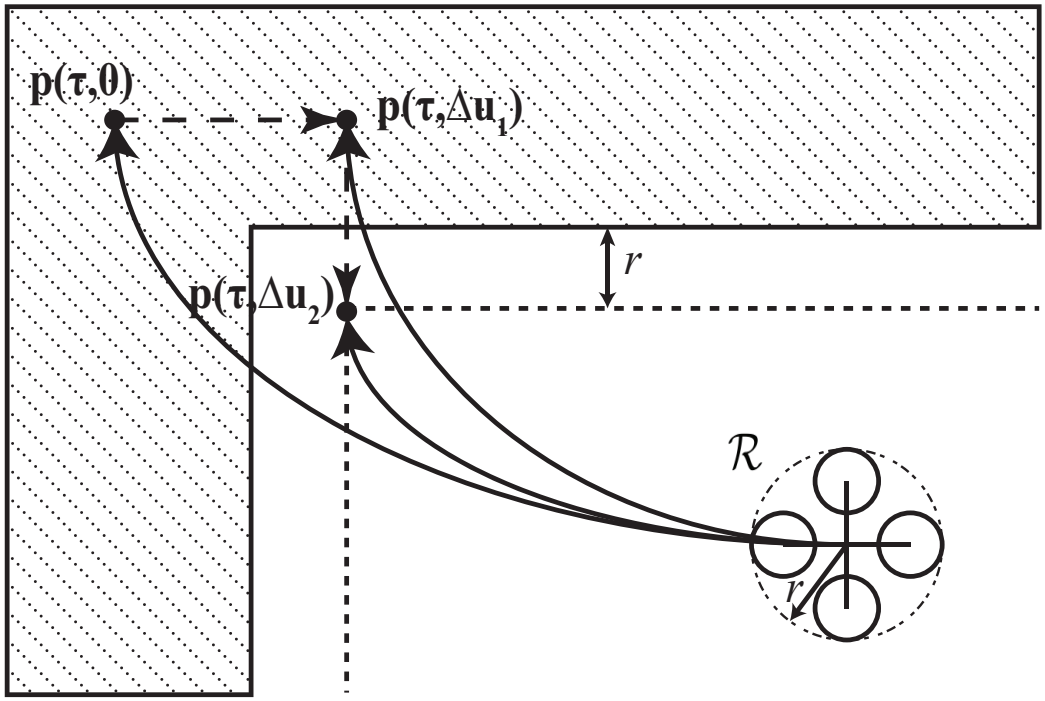

Figure 3.2. A schematic depicting our iterative optimization approach near convex corners of the free workspace. The future trajectory, given the operator's control input, collides with the left wall. A change to the operator input $\Delta\mathbf{u}_1$ is computed such that the vehicle will avoid collision at this instance. However, this change in control input still lets the robot collide with the top wall. Another change in operator control input $\Delta\mathbf{u}_2$ is computed. Since the change in control input $\Delta\mathbf{u}_2$ results in a trajectory free from collision, this is the last iteration and this operator input is then applied to the robot.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTS[1]

In this section we present our implementation details and results. We introduce quadrotor dynamics in Section 4.1. We address the definition of the environment in Section 4.2. In Section 4.3 we define the construction of the future trajectory of the vehicle and the halfspace constraints. Section 4.4 presents our methods for collision detection. We then present our simulation and experimental setup in Sections 4.5 and 4.6, respectively. Next we explore the method for determining an acceptable time horizon in Section 4.7. Finally, we present and discuss results obtained from our simulation and implementation experiments in Section 4.8.

4.1 Quadrotor Dynamics

For implementation we chose to use a Parrot AR.Drone 2.0 quadrotor (see Fig. 4.1). The state of the vehicle was defined as $\mathbf{x} = [\mathbf{p}^T, \mathbf{v}^T, \mathbf{r}^T, \mathbf{w}^T]^T \in \mathcal{X}$. Where $\mathbf{x}$ is 12-dimensional and consists of the quadrotors position $\mathbf{p} \in \mathbb{R}^3$ (m), velocity $\mathbf{v} \in \mathbb{R}^3$ (m/s), orientation $\mathbf{r} \in \mathbb{R}^3$ (rad) (rotation about axis $\mathbf{r}/\|\mathbf{r}\|$ by angle $\|\mathbf{r}\|$), and angular velocity $\mathbf{w} \in \mathbb{R}^3$ (rad/s). The control input was defined as $\mathbf{u} = [u_1, u_2, u_3]^T \in \mathcal{U}$. Where $\mathbf{u}$ is 3-dimensional and consists of the following: u_1 the desired vertical velocity (m/s), u_2 the desired roll (rad), and u_3 the desired pitch (rad).

It should be noted that the AR.Drone 2.0 allows for a fourth control input, yaw. Since yaw does not provide any additional degrees of freedom in our setup, we kept the input to yaw as zero. To ensure that there was no drift in the yaw (r_z), we implemented a proportional controller that continually drove this state to zero.

---

[1]Parts of this chapter are reprinted with permission, from 2014 IEEE International Conference on Robotics and Automation, "Automatic Collision Avoidance for Manually Tele-operated Unmanned Aerial Vehicles," by J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg

Figure 4.1. Photo of AR.Drone 2.0 quadrotor. Data source from [2].

Due to the complexities and internal controls of the Parrot AR.Drone 2.0, we did not fully know its dynamics. We do, however, propose that the dynamics of the quadrotor are defined well by:

$$\dot{\mathbf{p}} = \mathbf{v} \tag{4.1}$$

$$\dot{\mathbf{v}} = -k_{\text{drag}}\mathbf{v} + \exp(\lceil \mathbf{r} \rceil)[0, 0, k_{p1}(u_1 - v_z)]^T \tag{4.2}$$

$$\dot{\mathbf{r}} = \mathbf{w} \tag{4.3}$$

$$\dot{\mathbf{w}} = \begin{bmatrix} k_{p2}(u_2 - r_x) - k_d w_x \\ k_{p2}(u_3 - r_y) - k_d w_y \\ -k_{p3} w_z \end{bmatrix} \tag{4.4}$$

where $\lceil \mathbf{r} \rceil$ is the skew-symmetric cross-product matrix of $\mathbf{r}$ and k_{drag}, k_{p1}, k_{p2}, k_{p3}, and k_d are coefficients and gains found with standard parameter estimation techniques.

4.2 Environment Definition

We define an obstacle as an object consisting of a set of triangles in $\mathbb{R}^3$ [36]. As we mentioned before, while we developed our algorithm to work for on-board sensing systems and since we focus on collision avoidance, all obstacles in the environment are assumed to be known before hand. We note that the definition of the environment should consist of the "shadows" of the obstacles created from the perspective of the robot (see Fig. 3.1). Since we are focused on collision avoidance and the construction of these shadows is quite complicated, we approximated these shadows. We performed this approximation by using the dot product between the vector from the position of the vehicle to a point on an obstacle and the normal of the obstacle:

$$(\mathbf{p}_0 - \mathbf{k}) \cdot \mathbf{n} > 0, \tag{4.5}$$

where $\mathbf{p}_0$ is the current position of the vehicle, $\mathbf{k}$ is any defined point on the obstacle, and $\mathbf{n}$ is the normal of the obstacle. All obstacles that meet the condition of Eq. (4.5) are visible

(see Fig. 4.2). Once visible obstacles are determined, we offset the obstacles along their normals by the bounding sphere radius r of our vehicle so that the robot can be treated as a point.

We note that for on-board sensing systems the implementation of yaw will have no effect on the environmental definition. For full yaw implementation in our setup (where the environment is known before hand) we need to rotate the current frame of the quadrotor with respect to the fixed frame of the environment. We would do this through a rotation matrix and set of translations:

$$\mathcal{O}' = R(-\psi)(\mathcal{O} - \mathbf{t}) + \mathbf{t} \tag{4.6}$$

where $\mathcal{O}'$ is the rotated environment with respect to yaw, $\mathcal{O}$ is the original definition of the environment, ψ is the total angular displacement from the fixed frame due to yaw, and $\mathbf{t}$ is the translation vector from the origin of the fixed frame to the current position $\mathbf{p}$ of the vehicle. We also note that ψ does not equal r_z (from the state of the vehicle). The angular displacement relative to the quadrotor is zero ($r_z = 0$) and the total angular displacement with respect to the fixed frame is stored in ψ.

4.3 Future Trajectory and Halfspace Constraints

Given $\mathbf{u}$ (the current operator input), and $\mathbf{x}$ (the current state of the quadrotor), we estimate the future trajectory of the vehicle (up to time τ) by integrating Eq. (3.6) using

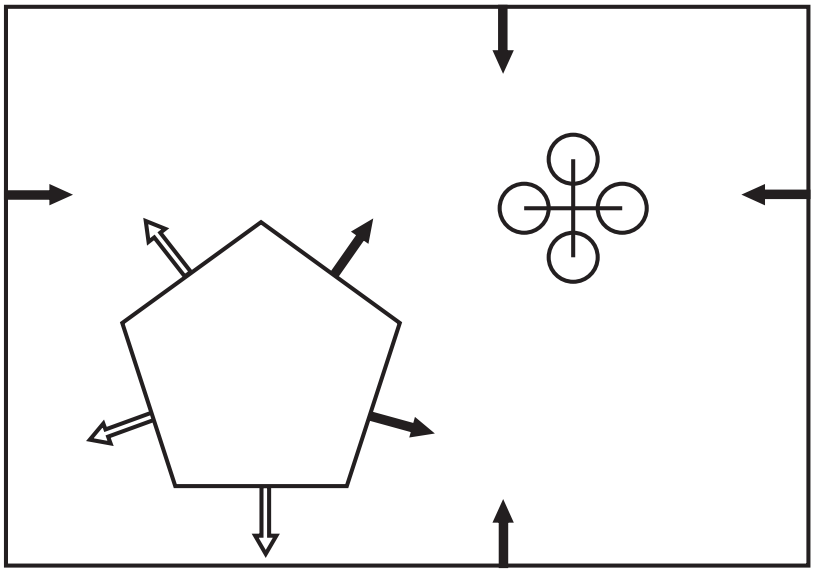

Figure 4.2. Example of which obstacles are visible from the viewpoint of the vehicle, through the use of Eq. (4.5). Visible obstacles are represented by filled normal arrows. The normals of the nonvisible obstacles are not filled.

small time increments with a 4th-order Runge-Kutta. We created the trajectory as a piecewise linear function that uses the increments from our integration and whose number of segments is defined by the user (from one on up to the total number of integration increments).

The halfspace constraint, defined by the intersection of the trajectory and obstacle (see Eq. (3.9)), was offset slightly from the obstacle to create a safety margin to keep the vehicle from getting too close to obstacles. This offset resulted in c (see Eq. (3.9)) being slightly increased. Our matrix $G(\tau)$, which is used in the constraint of a (see Eq. (3.11) or Eq. (3.12)), is approximated with numerical differentiation.

We note that the experimental results presented in this thesis used a system with linearized dynamics about hover (the state where there is zero change in altitude and zero angular displacement). A robot with linear dynamics is given in the traditional state-space formulation:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{4.7}$$

where A is the system matrix and B is control matrix of the system. The solution to Eq. (4.7) given a constant control input is:

$$\mathbf{x}(t) = F(t)\mathbf{x}(0) + G(t)\mathbf{u} \tag{4.8}$$

where

$$F(t) = e^{At}, \qquad\qquad G(t) = \left(\int_0^t e^{A(t-\tau)}\, d\tau\right) B.$$

It is assumed that a constant control input is valid for a human operator, since each sensing-control cycle of the algorithm is fast enough that the input is essentially constant over the period. The position of the robot is determined through a projection matrix $P \in \mathbb{R}^3$:

$$\mathbf{p}(t) = PF(t)\mathbf{x}(0) + PG(t)\mathbf{u} \tag{4.9}$$

With linear dynamics we were able to solely use the first and last point of our future trajectory to predict collisions. If the linear trajectory resulted in a collision, the position resulting from our position constraint $\mathbf{p}_c$ was directly determined by projecting $\mathbf{p}(\tau)$ onto the virtual wall (offset obstacle definition). We then rearranged Eq. (4.9) and solved directly for $\mathbf{u} + \Delta\mathbf{u}$ using our constraint $\mathbf{p}_c$:

$$(\mathbf{u} + \Delta\mathbf{u}) = (PG(\tau))^{-1}(\mathbf{p}_c - PF(\tau)\mathbf{x}(0)) \tag{4.10}$$

4.4 Collision Detection

4.4.1 General Condition

Here, we address the methods used and developed to predict collisions. The intersection between the piecewise linear trajectory and the obstacles determines where collisions will occur. To compute this the segment of the interest (of the future trajectory) is parameterized and given as [36]:

$$\mathbf{l}(a) = \mathbf{p}_i + a(\mathbf{p}_{i+1} - \mathbf{p}_i) \tag{4.11}$$

where $\mathbf{p}_{i+1}$ is at a point in time greater than $\mathbf{p}_i$, $\mathbf{l}$ represents the point between $\mathbf{p}_{i+1}$ and $\mathbf{p}_i$, and a is the parameter for the equation of the line.

We define a point on the obstacle in parametrized form as [36], [37]:

$$\mathcal{O}(v, w) = \mathbf{k}_0 + (\mathbf{k}_1 - \mathbf{k}_0)w + (\mathbf{k}_2 - \mathbf{k}_0)v \tag{4.12}$$

where $\mathcal{O}$ is the set of obstacles, the values $\mathbf{k}_0$, $\mathbf{k}_1$, and $\mathbf{k}_2$ represent vertex points of the obstacle triangle, and w and v are parameterized values of the obstacle triangle (see Fig. 4.3).

Using Eq. (4.11) and Eq. (4.12), and solving for the parameterized variables gives [38, 36]:

$$\begin{bmatrix} a \\ w \\ v \end{bmatrix} = M^{-1} \left[\mathbf{p}_i - \mathbf{k}_0 \right], \tag{4.13}$$

where

$$M = \begin{bmatrix} \mathbf{p}_i - \mathbf{p}_{i+1} & \mathbf{k}_1 - \mathbf{k}_0 & \mathbf{k}_2 - \mathbf{k}_0 \end{bmatrix}$$

It should be noted that a check needs to be performed on the matrix M for singularity, otherwise Eq. (4.13) is not possible to compute. If M is singular then the trajectory of the vehicle is parallel to the obstacle triangle, therefore there is no collision detected. If M is not singular and the solution to Eq. (4.13) meets the following barycentric coordinate conditions [37], then a collision is imminent within the obstacle triangle of Eq. (4.12):

$$a \in [0, 1] \qquad w, v \in [0, 1] \qquad (w + v) \leq 1 \tag{4.14}$$

where the condition on a checks that the line intersects the obstacle triangle only between points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. The w and v conditions check that the intersection is contained within the obstacle triangle points of $\mathbf{k}_0$, $\mathbf{k}_1$, and $\mathbf{k}_2$.

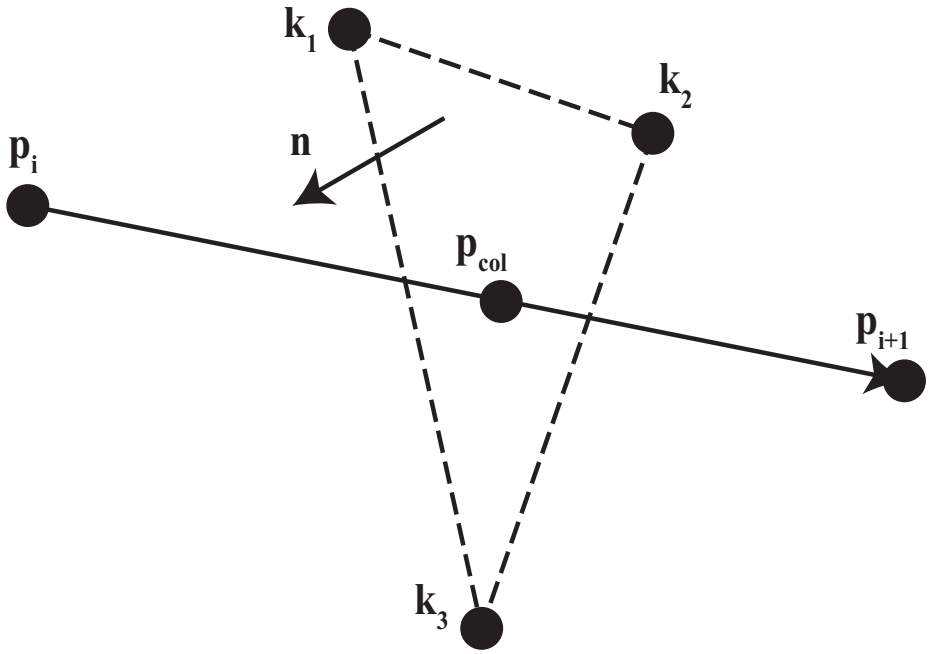

Figure 4.3. Intersection between a line and an obstacle triangle as defined in Eqs. (4.11–4.13). The points $\mathbf{k}$ are contained on the obstacle triangle, the points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are points on the line, the point $\mathbf{p}_{col}$ is the intersection of the line and obstacle triangle, and the vector $\mathbf{n}$ is the normal of the obstacle triangle.

4.4.2 Special Case

In general, the method of Section 4.4.1 will determine collisions the vehicle will encounter in the future. However, if the position of the vehicle passes through one of the previously mentioned offset obstacles (due to such things as: disturbances, noise in sensors, inaccuracies in the model, or a time horizon τ that is too small) collisions will not be detected with the method used in Section 4.4.1. Thus, if the following special condition (in addition to the w and v conditions of Eq. (4.14)) is met then a collision is also detected:

$$(\mathbf{p}_0 - \mathbf{k}_1) \cdot \mathbf{n} \leq 0 \tag{4.15}$$

where $\mathbf{p}_0$ is the current position of the vehicle, $\mathbf{n}$ is the normal of the obstacle triangle, and $\mathbf{k}_1$ is a point on the obstacle triangle.

4.5 Simulation Setup

Our simulations were performed on a laptop machine running Windows 7 Professional 64-bit with an Intel i7-2620M CPU (central processing unit) and 8GB of RAM (random access memory). The simulator was created within a 32 bit R2013a Matlab/Simulink environment along with the Simulink 3D (3 dimensional) Animation Toolbox. An overview of the model can be seen in Fig. 4.4. Visualization was performed with the 3D Animation VRML (virtual reality modeling language) environment. An example of the environment can be seen in Fig. 4.5 and Fig. 4.6. Interface with the simulator was achieved either through the use of a USB X-Box controller or slide controls within the Simulink model.

The simulated environment provided a test bed for our algorithm before implementation and a way in which we could validate our approach. We were able to visualize how our method worked with a live 3D rendered environment, providing feedback on the response of our vehicle and change to the user inputs (see Fig. 4.5 and Fig. 4.6). We were also able to obtain live plots of experimental data. For more details on the simulator model overview please refer to Appendix A.

4.6 Experimental Setup

Our experiments were performed at the Algorithmic Robotics Lab (ARL) located at the University of Utah. This lab is equipped with an OptiTrack V100:R2 motion capture system. All of our algorithm computations were performed with a desktop computer containing an Intel Core i7 3.4GHz processor with 8GB of RAM. Communication with the AR.Drone 2.0 was achieved through a wireless connection of 50Hz with a ROS (Robot Operating System) framework. The operator inputs were given through a USB Xbox controller. We

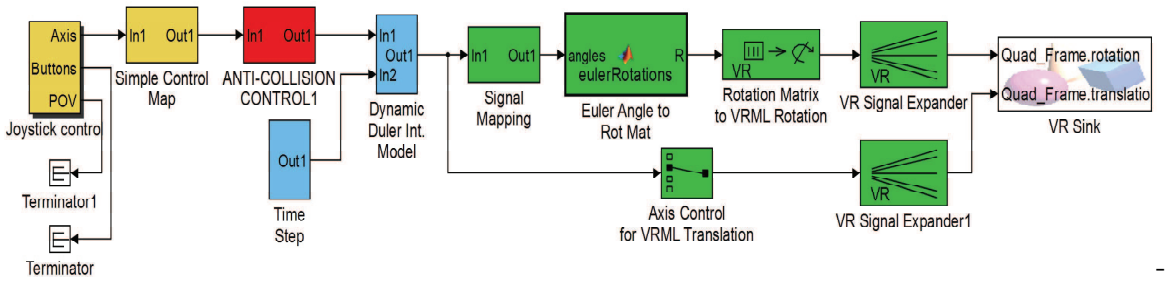

Figure 4.4. High level simulation overview. Yellow blocks represent the human interface and the signal mapping section. Red blocks represent the location of the collision avoidance system. Blue blocks represent the quadrotor dynamics. Green blocks represent the visualization and data mapping to the VRML system

Figure 4.5. Zoomed in view of VRML created environment in Simulink. The yellow beacon represents $\mathbf{p}(\tau, 0)$, the position at time τ given the operator input, current state of the vehicle, and dynamics of the vehicle. The orange beacon represents $\mathbf{p}(\tau, \Delta\mathbf{u})$, the position allowed by our collision avoidance algorithm.

obtained position information through the use of a Motive software interface. Information such as velocity, orientation, and angular velocity were all obtained from sensors on-board the AR.Drone 2.0. We obtained the current state of the vehicle through the use of a Kalman Filter. For more details on the ROS overview and the code used please refer to Appendix B.

4.7 Time Horizon Selection

The time horizon, what we refer to as τ, represents how far into the future our algorithm calculates the future trajectory (see Eq. (3.6)). This is a very important parameter. On one hand, if we set τ too low our algorithm may predict collisions so late that it does not have the capability to avoid them. On the other hand, if we set τ too large our algorithm

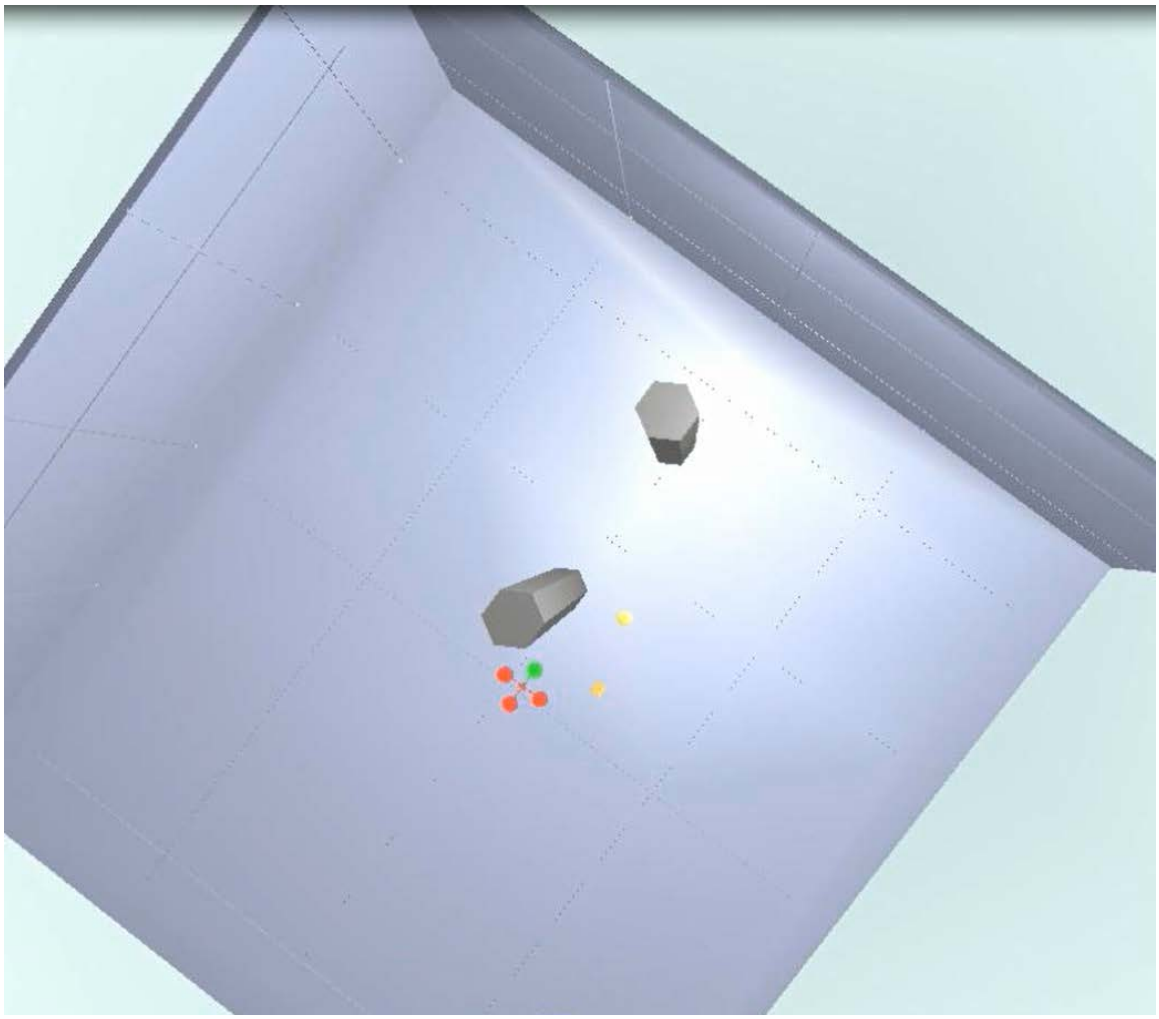

Figure 4.6. Zoomed out view of VRML environment with internal obstacles. The yellow beacon represents $\mathbf{p}(\tau, 0)$, the position at time τ given the operator input, current state of the vehicle, and dynamics of the vehicle. The orange beacon represents $\mathbf{p}(\tau, \Delta\mathbf{u})$, the position allowed by our collision avoidance algorithm.

may predict collisions so early it may continually override the operator inputs (even when there is more than sufficient time to avoid the future collision).

To be able to measure in a quantitative manner the effect of varying τ we setup an experiment to isolate the variable. To do this we repeatably flew the quadrotor directly toward a virtual wall (meaning, on a path perpendicular to the wall) from the opposing virtual wall of our environment, for various values of τ. In order to test the worst case scenario for our particular environment (one that would require the greatest amount of overriding input, and thus the greatest time horizon), we used the two opposing walls which were the furthest distance away from each other. Since there was only one obstacle present at any time, we had only one halfspace constraint relative to the change in our control input $\mathbf{\Delta u}$ (see Eq. 3.9).

The results of our experiments can be seen in Fig. 4.7 and Fig. 4.8. These results show the position of our vehicle over time, for multiple values of τ, as the operator tries to fly directly toward the virtual wall (at approx. 1.5m). From these results we can see that using a value of $\tau = 0.75s$ was too small. It undesirably caused our vehicle to overshoot into the obstacle and oscillate around the position of the virtual wall (under-damped behavior). Using a value of $\tau = 1.75s$ was too large. It also caused an undesirable response, but in this case caused our vehicle to very slowly approach the virtual wall (over-damped behavior). When we used a value of $\tau = 1.25$ we were able to obtain behavior very near to our desired response. This value caused our response to quickly but safely approach the virtual wall (critically damped behavior).

We make a special note of the overshoot that resulted in some cases. Overshoot is particularly dangerous due to the high likelihood of collisions if it should occur. There are a few reasons as to why overshoot behavior arises. First, if the time horizon is too small the controller may attempt to use a control input that is above the saturation limit allowed by the system (see Fig. 4.9, note that this does not result in overshoot since this is a simulation). Since these values are not allowed, due to saturation, the vehicle is unable to stop before passing the virtual wall. From inspection of Fig. 4.8, where the saturation limit was set at 0.85 rad, we can see that saturation did not present overshoot for this experiment. Second, overshoot arises from the limitations of only using position constraints to avoid collisions. While a position constraint will change the operator input such that the position of the vehicle will not exceed the boundary at time τ, it does not ensure that the velocity (directed into the virtual wall) at this point will also be zero (which will result in overshoot). However, the implementation of velocity constraints introduces another level

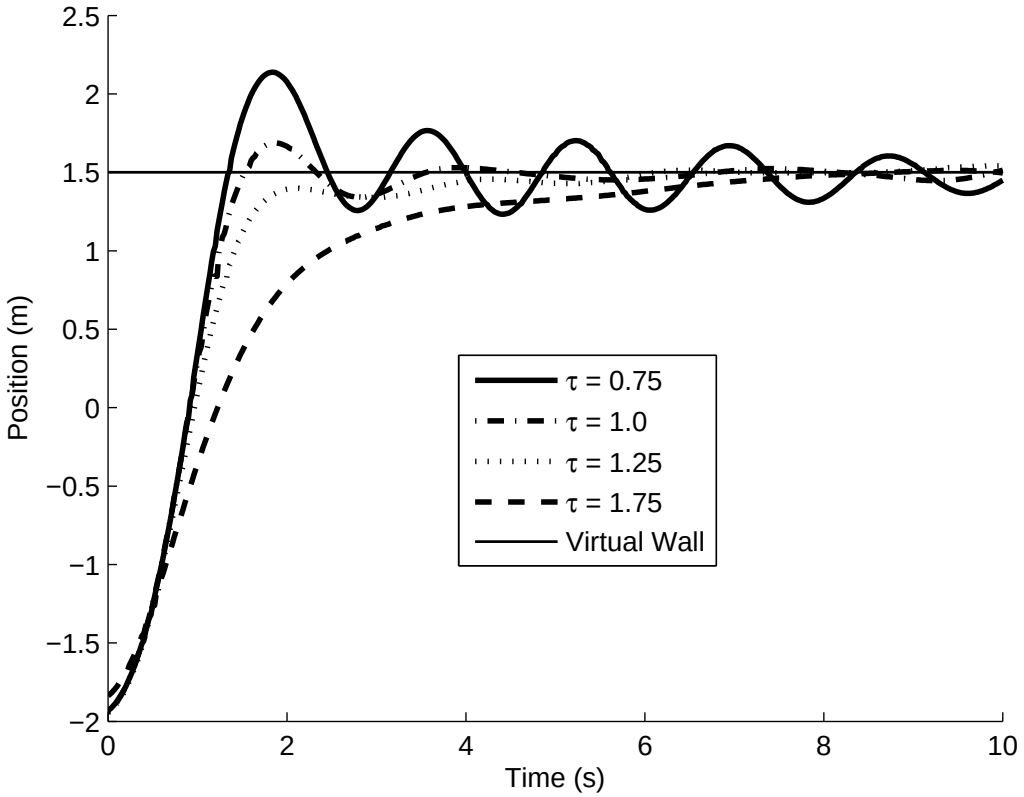

Figure 4.7. The results of experimentally varying the time horizon τ under similar conditions, compare to Fig. 4.8. The robot was flown directly toward the virtual wall (located approx. at 1.5m). These results show the position of the vehicle over time as it approaches the wall. The response of our system changes as our time horizon value is increased, for our system from under-damped ($\tau = 0.75$s), to critically damped ($\tau = 1.25$s), to overdamped ($\tau = 1.75$s). Modified from [3].

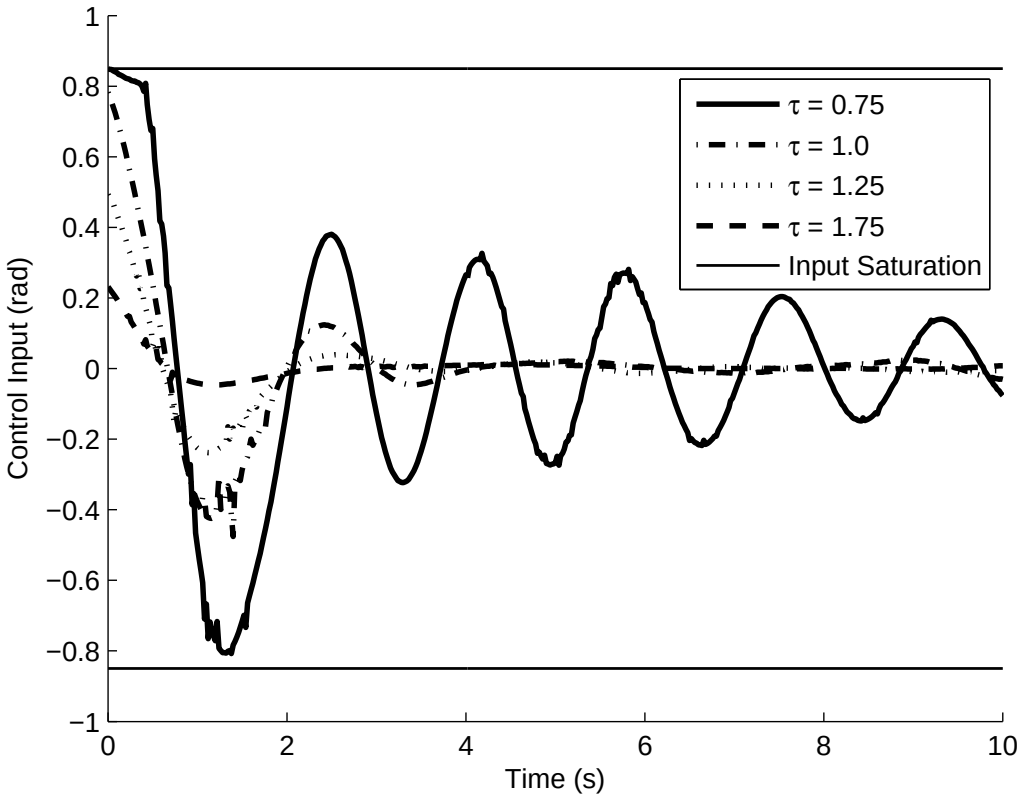

Figure 4.8. The results of inputs for experimentally varying the time horizon τ under similar conditions, compare to Fig. 4.8. For readability purposes and due to noise in feedback in the state of our vehicle from motion capture, these data have been smoothed with a filter. The robot was flown directly toward the virtual wall (located approx. at 1.5m). These results show the position of the vehicle over time as it approaches the wall. It should be noted that the response of our system changes as our time horizon value is increased, for our system from under-damped ($\tau = 0.75$s), to critically damped ($\tau = 1.25$s), to overdamped ($\tau = 1.75$s).

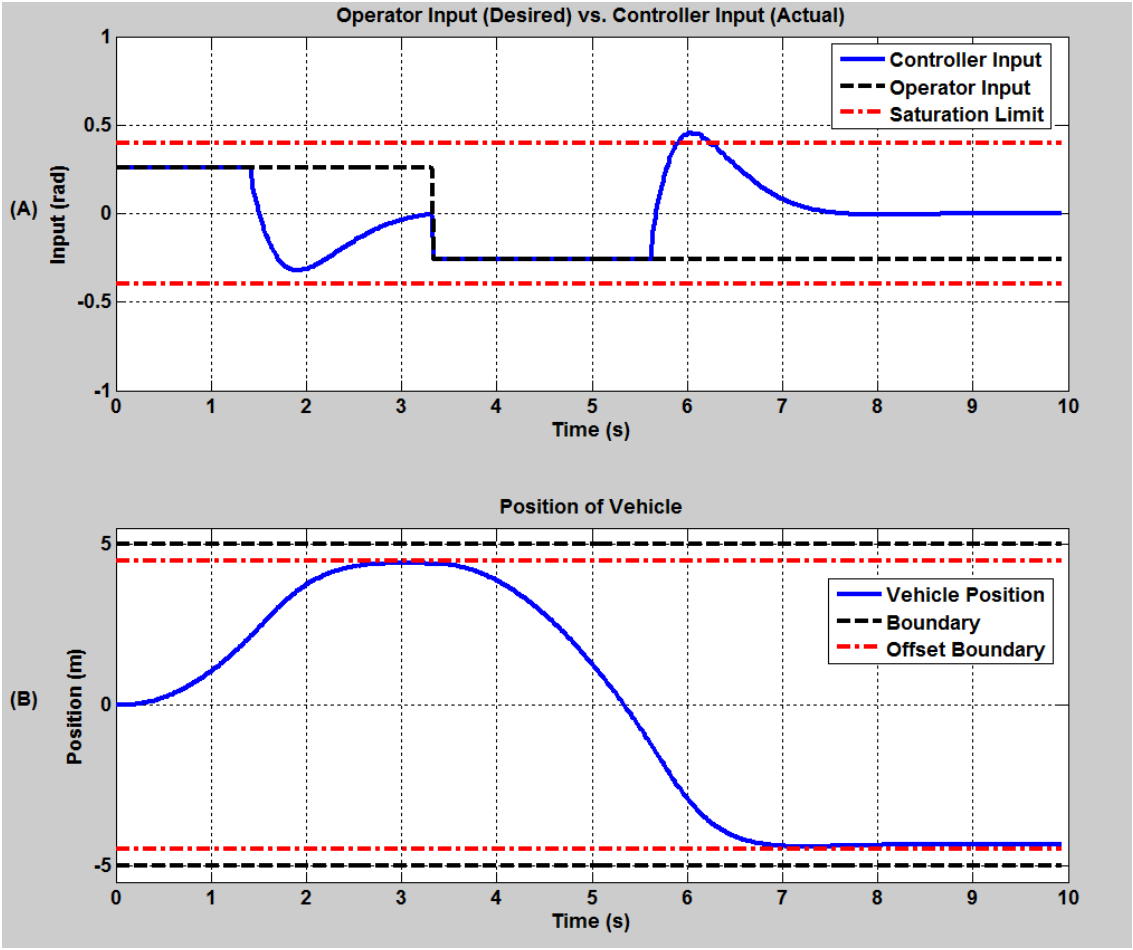

Figure 4.9. Sample of live collision avoidance graph from simulator for pitch operator inputs. These results were obtained by the operator attempting to pitch the vehicle into two walls (one in front of the vehicle and the other behind), along a path perpendicular to these walls. (**A**) Shows a comparison of operator inputs (black), controller inputs (blue), and vehicles input saturation limit (red). This plot also shows the ability of the controller to smoothly and sufficiently override the operator inputs. (**B**) Shows comparison of position of the vehicle (blue) and the obstacles (real in black, offset by bounding sphere radius and safety margin in red) and that the controller does prevent collisions with the obstacles.

of complexity. Lastly, overshoot can arise from limitations of the system model. If the differences between the model of the system and the actual system are large then there will be large amounts of error in the prediction and avoidance of collisions (resulting in overshoot).

It is also important to note that the optimal time horizon we found from this experiment is only for this particular system. If our algorithm is applied to different systems or vehicles, the optimal time horizon will vary depending on the dynamics and response of the robot. One way of determining this value for other systems is through experiments similar to the one presented here.

4.8 Results

In this section we present our results. We make note that while we will do our best to explain our results in the context of a thesis, the nature of our algorithm is difficult to express in a thesis. We thus recommend that the reader reference our video presented at **http://arl.cs.utah.edu/research/aca** for a complete overview of our experimental results. In our thesis we first present results obtained from simulation, followed by results obtained from implementation on an AR.Drone 2.0.

In simulation and implementation we used the basic method of Section 3.3.2, but instead of using the LP algorithm (see Chapter 3) we iterated for our solution through the use of Eq. (3.11) and for the experimental results presented in this thesis, through the use of Eq. (4.10) (see Section 4.3). Differences between the results of these methods should be minimal, if any difference at all. We note that when the vehicle overshot the boundary it may need more than three iterations to find a solution, thus we allowed the method to iterate up to 10 times.

4.8.1 Simulations

First we make note that while our original goal was to achieve real-time simulation, we were unable to do so. Our simulator runs at about one-tenth of real-time speed. Discrepancies in this can be accounted for due to limitations with computer computation power, difficulty in real-time rendering, and limitations of some Simulink function blocks (noncompiled).

We present a sample of results in Fig. 4.9. These results were obtained by the operator attempting to pitch the vehicle into two walls (the first in front and the other behind the vehicle), along a path perpendicular to these walls. This set of results compares two plots. Plot A shows operator inputs (depicted with a dashed black line) versus inputs which were

sent to the vehicle after passing through our algorithm (depicted with a solid blue line) versus the saturation input level (depicted with a dashed red line) over time. Plot B shows the position of the vehicle (depicted with a solid blue line) relative to obstacles within the environment (original obstacles depicted with a dashed black line, and obstacles offset by bounding sphere radius and safety margin depicted in red) over time.

First we make note of the latching and unlatching capabilities of our algorithm (shown in Plot A). If operator inputs will not result in a collision within the time horizon, then there is no change to operator input. However, if an operator input will result in a collision the inputs are overwritten. We make note that the overwritten operator inputs result in a smooth curve, giving an intuitive response for the operator.

We then point out the critically damped response of the position of the vehicle (shown in Plot B). The vehicle does not overshoot the boundary. The vehicle is also capable of stopping quickly.

Also, we are able to see a noticeable change in the position plot when the controller becomes active (shown in Plot B). When the overriding controls first become active, as the vehicle approaches the first wall (approx. $1.3s$), then the curvature of the position plot changes from concave to convex. Similarly when the overriding controls become active, as the vehicle approaches the second wall (approx. $5.75s$), the curvature changes from convex to concave.

We lastly note that at approximately 6 seconds our overriding control input exceeds the saturation limit of our system. While in simulation such an effect is harmless (since there are no saturation effects), in implementation this behavior can be dangerous. As we mentioned in Section 4.7, if the controller tries to utilize inputs which are above the saturation limit of the vehicle, overshoot of the virtual wall can occur and result in collisions.

We also present qualitative results obtained from maneuvering our quadrotor into an edge (see Fig. 4.10). These results show the quadrotor in six progressive time frames, where time moves from left to right. The quadrotor was given a constant roll and pitch input by the operator. The operator input is depicted with a filled arrow tip and the algorithm inputs are shown with unfilled arrow tips. We note that the algorithm input for the last frame of this simulation is zero.

We can see from these results, that the vehicle is capable of avoiding collisions while minimizing the change to the operator input. In frame 3 of Fig. 4.10 the algorithm detects a collision (adds a constraint) and begins overwriting the operator inputs. The inputs are overwritten in such a way as to preserve the original intent of the operator, in this case allowing the vehicle to strafe along the wall from frames 3 to 5. Once another collision is

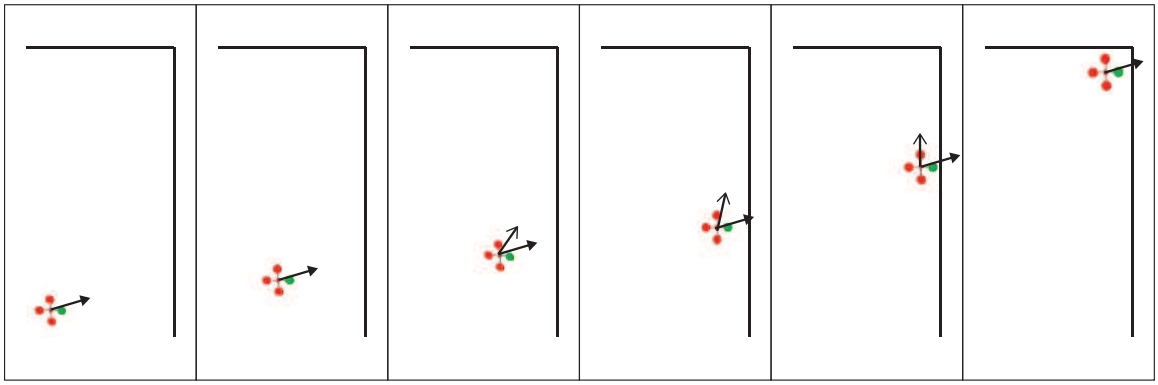

Figure 4.10. Example movement and response of the quadrotor vehicle in simulation. Frames are shown in progressive time intervals from left to right. Original operator inputs are depicted with filled arrow tips. In this case the operator input is constant. Collision avoidance inputs are depicted with unfilled arrow tips. The algorithm causes the vehicle to follow the original operator input as much as it can while avoiding collisions. This avoidance is shown by the vehicle moving to the obstacle as the user desires but not past it and then moving along the wall in a way that the vehicle can safely perform.

detected (and another constraint added) the algorithm minimally overwrites the operator inputs until the algorithm has no choice but to bring the vehicle to a complete stop with the current operator inputs, as in frame 6.

4.8.2 Experiments

For our experiments we constructed an environment containing four virtual walls, a floor, a ceiling, and, in some cases, internal obstacles with different orientations. A sample of results obtained from experiments in this environment are shown in Figs. 4.11, 4.12, and 4.13.

We reiterate that part of our results were performed while experimentally determining an appropriate time horizon τ. The results of tuning this parameter are clearly shown in Fig. 4.7 and a full discussion is presented in Section 4.7.

We also refer the reader to a video presentation of some of the results presented in this thesis, as well as to some which have not been included within the text of this thesis (refer to **http://arl.cs.utah.edu/research/aca**). In one of the experimental results presented in the video (not shown in text) the quadrotor was flown along each boundary of the environment with operator inputs that originally would have resulted in collisions. This experiment showed the ability of our approach in handling edges and corners created by our halfspace constraints. It also shows the critically damped behavior achieved from the

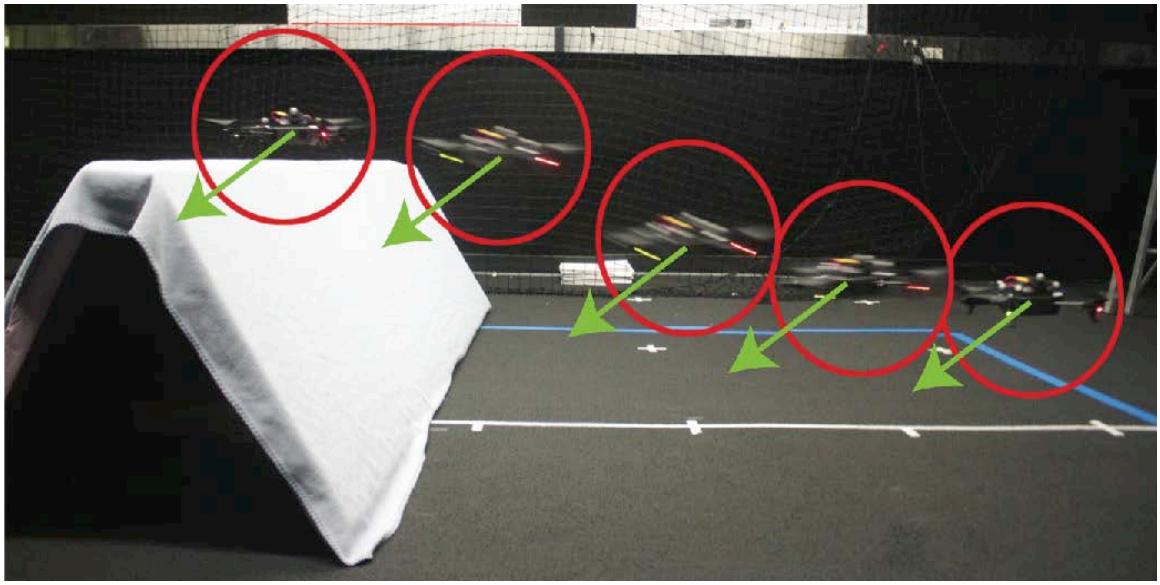

Figure 4.11. A time-lapsed picture (time progresses from right to left) showing our approach on an environment where an internal obstacle is placed on the floor. The green arrows are a depiction of the operator inputs given to fly the vehicle into the floor and into the obstacle. Our algorithm minimizes changes to the operator inputs while ensuring that collisions are avoided by allowing the vehicle to fly forward and around the speed bump. We refer the reader to **http://arl.cs.utah.edu/research/aca** for videos of the experiment.

choice of our time-horizon. Another one of our experimental results shown within our video (not shown in text) presents a case of flying the vehicle into one of the obstacle edges. This experiment is similar to those presented in Fig. 4.11, but shows the ability of our approach in avoiding collisions with another combination of operator inputs.

Fig. 4.11 presents a time-lapse picture, as the vehicle moves from the right to the left. The vehicle in this experiment was given a constant user input directing the vehicle down into the floor and with a forward pitch (forward direction) as indicated by the green arrow. The red circle surrounding the quadrotor is to help visualize the bounding sphere about the vehicle. Our algorithm prevented the quadrotor from doing the following: (1) colliding into the floor during the first few frames and (2) colliding with the internal obstacle "speed bump" during the last few frames. We also note that the vehicle deflects off of the speed bump (instead of just stopping when this obstacle is reached). This behavior is explained by the minimization of the change to the operator input. Since the operator is continuously directing the quadrotor forward the algorithm still allows a portion of that input that would not result in a collision.

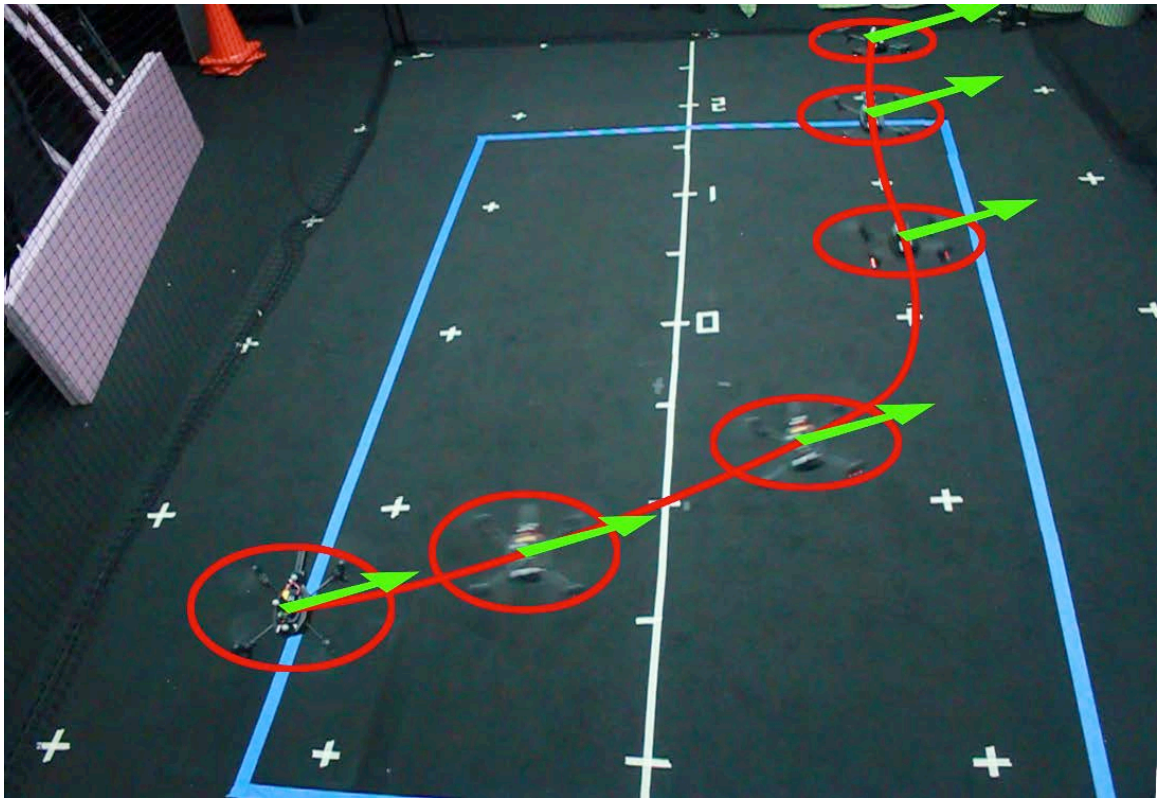

Figure 4.12. A time-lapsed picture (time progresses from left to right) showing the behavior of our approach when the operator attempts to fly the quadrotor into a wall at an angle. Our approach minimizes the change to the operator inputs by decreasing the input perpendicular to the wall until the wall is reached and then only using the control input that is parallel to the obstacle. We refer the reader to the video at **http://arl.cs.utah.edu/research/aca** for videos of our experiment.

In Fig. 4.12 we show the results of an operator trying to maneuver our vehicle into one of the environment walls and then corners. The time-lapse in this figure moves from the left to the right. The operator is attempting to fly the quadrotor at an angle into the right wall and then continues to give this input to the far wall. From this result we can see that our approach is capable of working in this workspace dimension (front and side wall) and operator control input dimension (pitch and roll) just as well as the dimensions shown in Fig. 4.11 (floor and internal obstacle, pitch and thrust control input). We again note our approaches capability of avoiding collisions while minimizing the change to the operator inputs. In this case our approach does this by minimizing the change to the operator inputs perpendicular to the side wall, until the wall is reached, and then proceeds to only use the control input that is parallel to the obstacle.

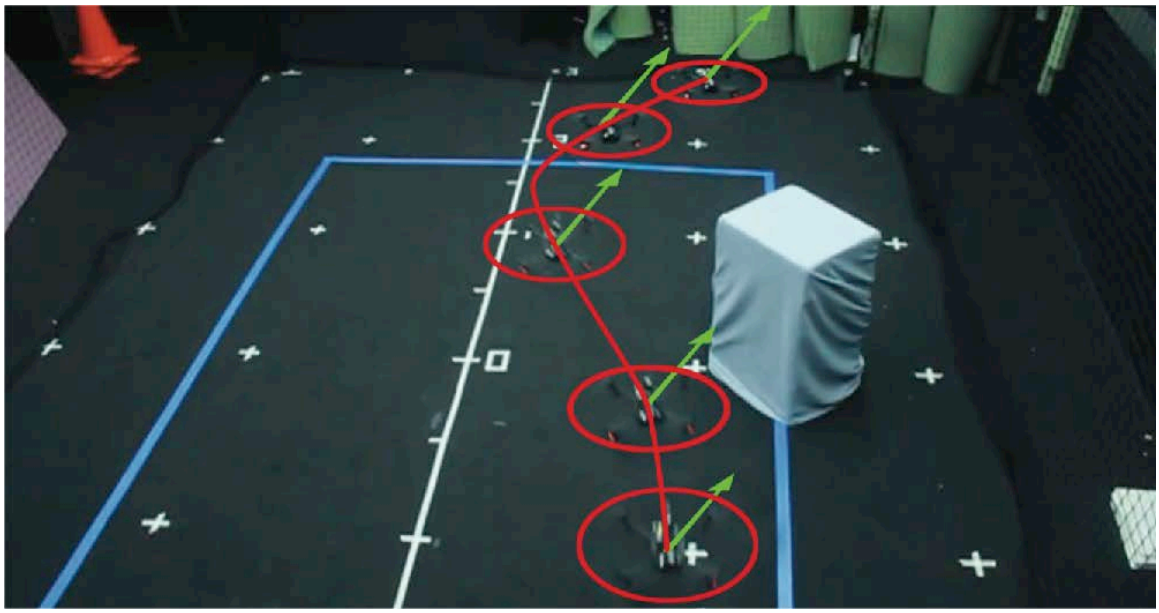

Figure 4.13. A time-lapse picture (time progresses from bottom to top) showing the behavior of our approach when the operator attempts to fly the vehicle into a wall where an obstacle is protruding from. Our approach first prevents collision with the wall being flown into and then performs an evasive maneuver in which our approach helps fly the vehicle around the obstacle. We refer the reader to the video at **http://arl.cs.utah.edu/research/aca** for videos of our experiment.

We lastly present results of our algorithm being implemented on an environment where an obstacle is protruding from one of the side walls (see Fig. 4.13). This experiment provides more insight into the evasive capabilities of our approach. The operator attempts to continually fly the vehicle forward and into the wall, as they do so, the vehicle not only avoids the wall but goes around the internal obstacle. The operator control input that is perpendicular to the wall was negated at the beginning of the experiment (since it would result in a collision). In the middle of the experiment the operator control input perpendicular to the wall is overwritten, such that the resulting input is opposite from the operator input and allows the vehicle to maneuver around the obstacle.

CHAPTER 5

CONCLUSIONS[1]

Collision avoidance is a common problem with a long history in robotics. One particular application for collision avoidance is operator driven vehicles, which can be difficult to control. This is especially true for systems that have complex dynamics, are nonlinear, and tele-operated.

The purpose of this thesis was to develop an algorithm to perform automatic collision avoidance for manually tele-operated unmanned aerial vehicles (UAVs). Our approach continually estimated the future trajectory of the vehicle up to some time horizon τ (using the vehicle's current state, the vehicle's dynamics, and the current operator input). If a collision was detected along the future trajectory, the change to the operator control input was minimized such that a collision would not occur. If a collision did not occur, there were no changes performed on the operator control inputs.

The results we obtained verified and validated the workings of our system. We showed that our approach worked in a Matlab/Simulink simulation, as well as on an AR.Drone 2.0. With the use of our approach we found the following:

1. The operator was unable to:
   (a) Collide with the environment.
   (b) Collide with internal obstacles that were in different orientations.
   (c) Collide with the environment and internal obstacles even when intentionally attempting to do so.
2. While avoiding collisions, changes to the operator input were minimized to present an intuitive environment for the operator.

---

[1]Parts of this chapter are reprinted with permission, from 2014 IEEE International Conference on Robotics and Automation, "Automatic Collision Avoidance for Manually Tele-operated Unmanned Aerial Vehicles," by J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg © 2014 IEEE

3. Our approach created a response such that the operator could use the environment and internal obstacles for navigation.

4. Avoidance of obstacles could be achieved with a critcally damped response.

We have been able to present promising results. We would like to suggest a few topics for future work with our method. These topics are discussed and listed with the following:

- **On-board Sensing**: While our approach was created for systems where the environments are determined locally by the robot, we focused on collision avoidance and used preprogrammed obstacles. Due to using preprogrammed obstacles we also estimated the visible environment (see Section 4.2). To fully make our approach applicable to unknown indoor environments we would need to add some form of on-board SLAM [39],[40],[41], or a detection method using multimodal sensing as outlined by Niewenhuisen et al. [28].

- **LP-type Algorithm**: The construction of our problem and solution fit well with the use of an LP-type algorithm [35] (see Section 3.3.2). We recommend the use of the RVO-3D library to implement this in the future [42].

- **Haptic Feedback**: Other methods similar to ours provided haptic feedback to the operator [33, 34]. The use of a haptic feedback mechanism could potentially be useful in our approach as well.

- **Time Horizon (Selection)**: We selected a time horizon τ using experimental results of our system. Performing such experiments may not always be feasible or desired or provide optimal results. There may be other methods that can be applied to help select an optimal value for this.

- **Time Horizon (Fixed)**: The time horizon τ we used was always assumed to be fixed. A fixed time horizon does have its benefits for computations and smooth responses. However, a dynamic time horizon could be more versatile for changing environments, robots that experience drastic changes in state, or variable dynamic systems.

- **Moving Obstacles**: In our work we explored environments that were static and did not include other robots. Due to the inherent dynamics in many environments, adding the capability to our approach to handle such situations would be worthwhile.

- **Implementation on Other Systems**: We were able to successfully implement our algorithm on an AR.Drone 2.0. Due to the generalities of our algorithm, the method

we developed should have capabilities of being applied to other mobile systems (e.g., hovercrafts, submersible robots, etc.).

- **Velocity Constraints**: We used position constraints to prevent collisions. As mentioned in Section 4.7, while a position constraint on the future trajectory will change the input such that the position of the vehicle will not exceed the virtual wall at time τ, it does not ensure that the velocity (directed into the virtual wall) at this point will also be zero. If the velocity (directed into the virtual wall) is not zero when the vehicle is at the boundary, overshoot of the boundary will result. The implementation of velocity constraints could thus be useful, but it should be noted that these would introduce another level of complexity to the solution of our problem.

APPENDIX A

SIMULATION SAMPLES

A.1 Simulink Samples

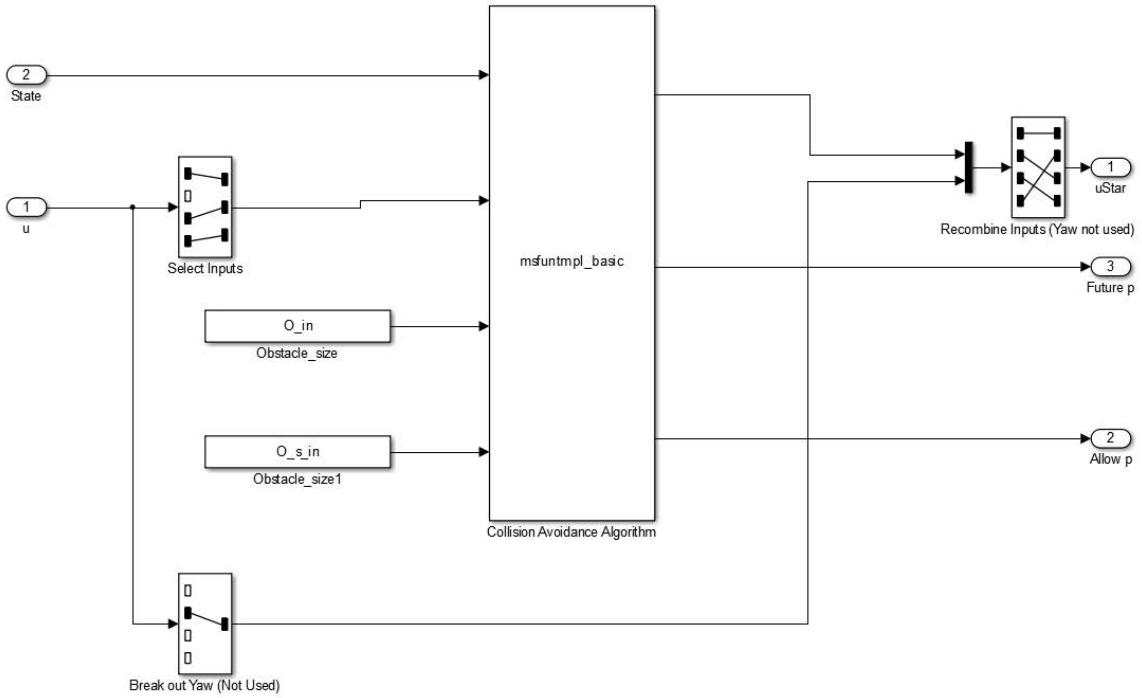

Figure A.1. View of Simulink collision avoidance block. Our collision avoidance algorithm is located in the custom made "msfuntmpl basic" block.

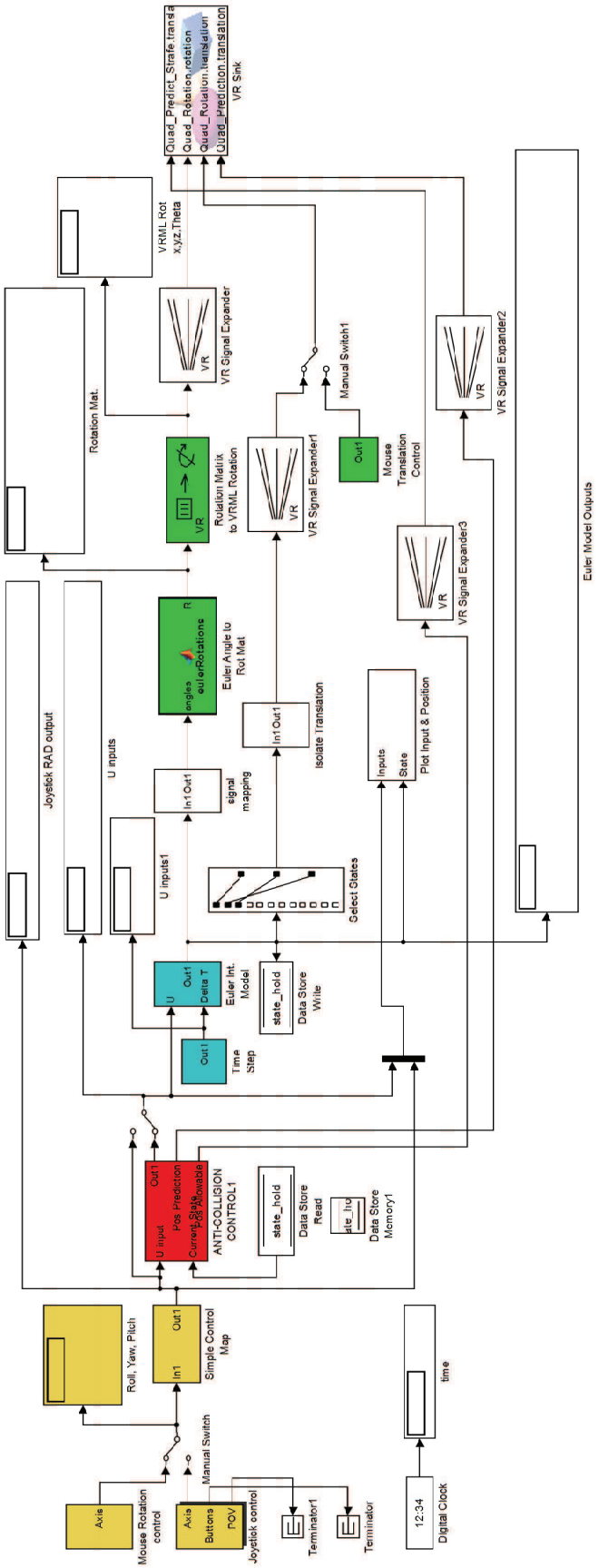

Figure A.2. Full view of Simulink model.

APPENDIX B

ROS SAMPLES

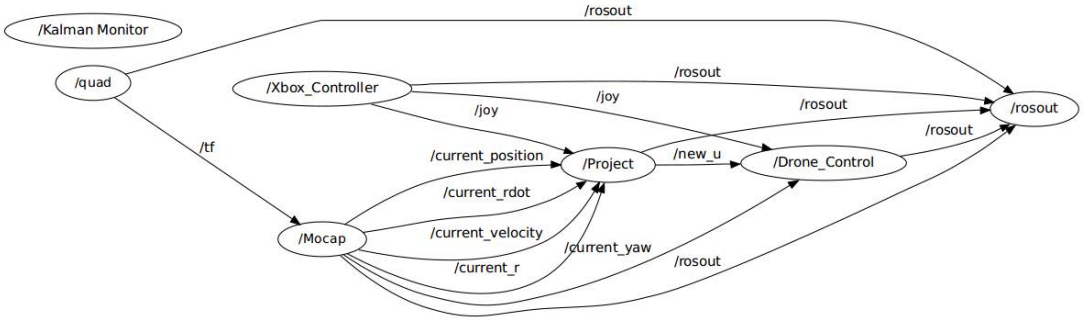

Figure B.1. Full overview of ROS graph. The Collision Controller is located in the project node.

APPENDIX C

PURCHASED MATERIALS

Table C.1. Simulation Material

Material	**No. Units**	**Part No.**	**Vendor**	**Cost (USD)**
MS XBox 360 Controller	1	52A-00004	Microsoft	40
Matlab/Simulink R2013a	1	SVDWL13A	Mathworks	99
Simulink 3D Animation Toolbox	1	VRWIN63	Mathworks	20

REFERENCES

[1] J. J. Abbott, P. Marayong, and A. M. Okamura, "Haptic virtual fixtures for robot-assisted manipulation," in *Proc. Int. Symp. on Robotics Research*, 2007, pp. 49–64.

[2] Parrot. (2014) Parrot ar.drone 2.0 photos. [Online]. Available: http://ardrone2.parrot.com/photos/photo-album/

[3] J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg, "Automatic collision avoidance for manually tele-operated unmanned aerial vehicles." IEEE, 2014.

[4] J. Roberts, T. Stirling, J.-C. Zufferey, and D. Floreano, "Quadrotor using minimal sensing for autonomous indoor flight," in *Proc. of the European Micro Air Vehicle Conference and Flight Competition*, 2007.

[5] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a gps-denied environment," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 2008, pp. 1814–1820.

[6] J. Mendes and R. Ventura, "Assisted teleoperation of quadcopters using obstacle avoidance," *Journal of Automation, Mobile Robotics & Intelligent Systems*, vol. 7, no. 1, 2013.

[7] L. Hull, "Drone makes first UK arrest as police catch car thief hiding under bushes," *Daily Mail*, vol. 12, 2010.

[8] Amazon. (2014) Amazon prime air. [Online]. Available: http://www.amazon.com/b?node=8037720011

[9] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5, 2004, pp. 4393–4398.

[10] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 4, no. 3, pp. 143–153, 2003.

[11] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, "Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations," *Vehicle System Dynamics*, vol. 44, no. 7, pp. 569–590, 2006.

[12] J. C. McCall and M. M. Trivedi, "Driver behavior and situation aware brake assistance for intelligent vehicles," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 374–387, 2007.

[13] J. Cao, H. Liu, P. Li, and D. J. Brown, "State of the art in vehicle active suspension adaptive control systems based on intelligent methodologies," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 9, no. 3, pp. 392–405, 2008.

[14] L. B. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," in *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, 1993, pp. 76–82.

[15] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2007, pp. 1–20.

[16] P. Marayong, M. Li, A. M. Okamura, and G. D. Hager, "Spatial motion constraints: Theory and demonstrations for robot guidance using virtual fixtures," in *Robotics and Automation (ICRA), 2003 IEEE International Conference on*, vol. 2, 2003, pp. 1954–1959.

[17] A. Bettini, P. Marayong, S. Lang, A. M. Okamura, and G. D. Hager, "Vision-assisted control for manipulation using virtual fixtures," *Robotics, IEEE Transactions on*, vol. 20, no. 6, pp. 953–966, 2004.

[18] M. Dewan, P. Marayong, A. M. Okamura, and G. D. Hager, "Vision-based assistance for ophthalmic micro-surgery," in *Proc. Medical Image Computing and Computer-Assisted Intervention*, 2004, pp. 49–57.

[19] T. L. Gibo, L. N. Verner, D. D. Yuh, and A. M. Okamura, "Design considerations and human-machine performance of moving virtual fixtures," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 2009, pp. 671–676.

[20] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[21] P. Saranrittichai, N. Niparnan, and A. Sudsang, "Robust local obstacle avoidance for mobile robot based on dynamic window approach," in *Int. Conf. on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 2013.

[22] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[23] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, "Collision avoidance under bounded localization uncertainty," in *IEEE Int. Conf. on Intelligent Robots and Systems*, 2012.

[24] J. van den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, "LQG-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 346–353.

[25] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[26] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 19, pp. 1179–1187, 1989.

[27] T. Fraichard 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, 2003, pp. 388–393.

[28] M. Nieuwenhuisen, D. Droeschel, J. Schneider, D. Holz, T. Labe, and S. Behnke, "Multimodal obstacle detection and collision avoidance for micro aerial vehicles," in *Mobile Robots (ECMR), 2013 European Conference on.* IEEE, 2013, pp. 7–12.

[29] G. Ducard and R. D'Andrea, "Autonomous quadrotor flight using a vision system and accommodating frames misalignment," in *Industrial embedded systems, 2009. SIES'09. IEEE international symposium on*, 2009, pp. 261–264.

[30] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a navigation system for autonomous indoor flying," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 2009, pp. 2878–2883.

[31] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 477–483.

[32] D. Bareiss and J. van den Berg, "Reciprocal collision avoidance for robots with linear dynamics using LQR-obstacles," in *IEEE Int. Conf. Robotics and Automation*, 2013.

[33] A. M. Brandt and M. B. Colton, "Haptic collision avoidance for a remotely operated quadrotor UAV in indoor environments," in *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, 2010, pp. 2724–2731.

[34] X. Hou and R. Mahony, "Dynamic kinesthetic boundary for haptic teleoperation of aerial robotic vehicles," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 4549–4950.

[35] M. De Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications.* Springer, 2008.

[36] M. E. Mortenson, in *Geometric Modeling.* John Wiley and Sons, 1985.

[37] D. Badouel, "An efficient ray-polygon intersection," in *Graphics gems.* Academic Press Professional, Inc., 1990, pp. 390–393.

[38] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, 1997.

[39] S. Thrun, M. Diel, and D. Hähnel, "Scan alignment and 3-d surface modeling with a helicopter platform," in *Field and Service Robotics*, 2006, pp. 287–297.

[40] A. Bry, A. Bachrach, and N. Roy, "State estimation for aggressive flight in GPS-denied environments using onboard sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2012)*, St Paul, MN, 2012.

[41] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 229–241, 2001.

[42] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Int. Symp. on Robotics Research*, 2011, pp. 3–19.