Low-Latency Trajectory Planning for High-Speed Navigation in Unknown Environments

by

Brett Thomas Lopez

B.S., Aerospace Engineering
University of California - Los Angeles (2014)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Signature redacted

Author

Signature redacted

Department of Aeronautics and Astronautics

June 18, 2016

Signature redacted

Certified by...

Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by

Paulo Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee
Abstract
The ability for quadrotors to navigate autonomously through unknown, cluttered environments at high-speeds is still an open problem in the robotics community. Advancements in light-weight, small form factor computing has allowed the application of state-of-the-art perception and planning algorithms to the high-speed navigation problem. However, many of the existing algorithms are computationally intensive and rely on building a dense map of the environment. Computational complexity and map building are the main sources of latency in autonomous systems and ultimately limit the top speed of the vehicle.

This thesis presents an integrated perception, planning, and control system that addresses the aforementioned limitations by using instantaneous perception data instead of building a map. From the instantaneous data, a clustering algorithm identifies and ranks regions of space the vehicle can potentially traverse. A minimum-time, state and input constrained trajectory is generated for each cluster until a collision-free trajectory is found (if one exists). Relaxing position constraints reduces the planning problem to finding the switching times for the minimum-time optimal solution, something that can be done in microseconds.

Our approach generates collision-free trajectories within a millisecond of receiving perception data. This is two orders of magnitude faster than current state-of-the-art systems. We demonstrate our approach in environments with varying degrees of clutter and at different speeds.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics
Acknowledgments

I would first like to thank my adviser Jonathan How. You spend countless hours giving me advice and steering me in the right direction. You’re always able to provide the insight I need to overcome the roadblocks I encounter in my research. Thank you for giving me the opportunity to study and do research here at MIT.

I would like to the thank the members of the Aerospace Controls Lab that have made my time here at MIT enjoyable. Thanks to Michael Klinker, Shayegan Omidshafiei, Justin Miller, Michael Everett, Steven Chen, Michael Everett, Jack Quindlen, Kris Frey, and James Wiken. A special thanks to my two closest friends here at MIT: Byron Patterson and Mark Cutler. Byron, you always provided the comic relief I needed to get through the day. Thanks for sharing so many fun hours in the gym and running around Boston. Mark, thank you for always being there and giving me sound advice about work and life when I needed it. I always enjoyed spending time with you and your family and I wish our time at MIT overlapped more.

Thank you to my family for always supporting my endeavors. Whether it was wanting to play hockey or study Aerospace engineering, you were always there to support and help me in any way possible. I hope I have made you proud.

Last, and certainly not least, I would like to thank my wonderful partner Kimiko Kossler. Thank you for being so understanding when I had to work long hours during the week and weekend. You, and our three dogs Koda, Penny, and Franklin, always cheered me up after those long days and gave me the motivation to keep going. I hope I bring as much happiness to your life as you do to mine.

It is also necessary to acknowledge the financial support that enabled my educational pursuits at MIT. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374. The National Science Foundation supported the majority of my time at MIT. The author would also like to acknowledge Boeing Research & Technology for support of the indoor flight facility and hardware used in this work.
# Contents

1 Introduction 17
   1.1 Motivation ............................................. 17
   1.2 Literature Review ...................................... 18
   1.3 Contributions ......................................... 23

2 Control 25
   2.1 Overview ............................................... 25
   2.2 Dynamics .............................................. 25
      2.2.1 Quaternions ......................................... 25
      2.2.2 Equations of Motion ................................ 27
   2.3 Position and Attitude Control ......................... 28
   2.4 Dynamically Consistent Angular Rates ................. 32
   2.5 Feedforward ........................................... 34
      2.5.1 Motivation .......................................... 34
      2.5.2 Experimental Verification ......................... 35
   2.6 Stability ............................................. 40
      2.6.1 Attitude Controller ............................... 40
      2.6.2 Position Controller ............................... 41
   2.7 Summary ................................................ 43

3 Perception and Planning 45
   3.1 Overview ............................................... 45
   3.2 System Architecture .................................... 45
3.3 Obstacle Clustering .................................................. 47
3.4 Constrained Decoupled Velocity Planning ....................... 50
  3.4.1 Modeling State and Actuator Constraints .................... 53
  3.4.2 Model Order ..................................................... 55
  3.4.3 Decoupled Planning .............................................. 56
  3.4.4 Minimum-Time Velocity Planning ............................. 59
  3.4.5 Collision Checking .............................................. 66
3.5 Safety ................................................................. 68
3.6 Algorithm Benchmarking .............................................. 69
3.7 Summary ............................................................... 70

4 Hardware and Software ............................................... 75
  4.1 Overview ............................................................ 75
  4.2 Hardware ............................................................ 75
  4.3 Software ............................................................. 78
    4.3.1 Flight Computer ............................................... 78
    4.3.2 Autopilot ....................................................... 80
    4.3.3 Measurable Augmented Reality .............................. 81
  4.4 Summary ............................................................. 83

5 Results ................................................................. 85
  5.1 Overview ............................................................ 85
  5.2 Flight 1 .............................................................. 86
  5.3 Flight 2 .............................................................. 87
  5.4 Flight 3 .............................................................. 89
  5.5 Summary ............................................................. 92

6 Conclusion and Future Work ......................................... 101
  6.1 Conclusion .......................................................... 101
  6.2 Future Work ........................................................ 103
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Quadrotor model and coordinate system definition.</td>
<td>27</td>
</tr>
<tr>
<td>2-2</td>
<td>Successive loop closure example. The output of the outer-loop controller ( C_\phi ) is the input to the inner loop controller ( C_i ). The output of the inner loop controller is sent to the plant ( G ).</td>
<td>29</td>
</tr>
<tr>
<td>2-3</td>
<td>Overview of control strategy. Feedback on position and velocity generate an attitude and angular rate setpoint for the inner loop controller. Feedback on attitude and angular rate generates motors commands. Feedforward acceleration and jerk are used to supplement the outer-loop controller.</td>
<td>29</td>
</tr>
<tr>
<td>2-4</td>
<td>Simple block diagram that uses a feedforward model ( F ) to improve tracking of a reference signal. The model inversion technique produces zero tracking error.</td>
<td>35</td>
</tr>
<tr>
<td>2-5</td>
<td>A curve ( C ) parameterized by variable ( s ). The acceleration and jerk required to traverse ( C ) are known functions of the speed ( v ), curvature ( \kappa ), and rate of change of curvature ( \kappa' ). These can be used for feedforward.</td>
<td>36</td>
</tr>
<tr>
<td>2-6</td>
<td>Real-time visualization of relevant experimental data using the Measurable Augmented Reality (MARS) system. The feedforward model significantly improves tracking performance by decreasing amplitude error and phase lag. The control effort, measured by throttle, is also lowered.</td>
<td>38</td>
</tr>
<tr>
<td>2-7</td>
<td>Position and velocity tracking with and without feedforward. The vehicle responds faster to position and velocity commands when feedforward is used. Fast response time is crucial for collision avoidance.</td>
<td>39</td>
</tr>
</tbody>
</table>
Comparison of the feedforward model to the total commanded acceleration and jerk. Acceleration has a normalized error less than 18% in the beginning stage but grows to 38% because of velocity overshoot in the latter stage of the acceleration phase. Jerk has a normalized error less than 30%. This is small enough to validate the instantaneous jerk assumption.

Typical system architecture for navigating in unknown environments. Map building and trajectory generation are the primary contributors to system latency. High-speed navigation requires low-latency perception and planning.

The proposed system architecture for navigating in unknown environments. Obstacle clustering efficiently processes instantaneous data and removes the need to build a local map. Traversable regions are identified and a velocity plan is generated to avoid obstacles.

Key features from the laser scan data, such as obstacles and free space, are represented by a set of clusters. The size of the obstacle and free space dictate how many clusters are used to represent them. Clusters are always placed at the boundary of large obstacles.

Illustration of the cost of each cluster. $\phi_{last}$ is used to prioritize clusters that are in the same general direction as the last selected cluster. $\theta_{goal}$ and $r$ are used to guide the vehicle to the goal and prevent obstacle hugging.

An example of why using a triple integrator model is more accurate than a double integrator model. The double integrator model does not account for the initial acceleration $a_0$. As a result the vehicle deviates from the desired path and crashes. The triple integrator accounts for $a_0$ and is thus more accurate.
3-6 Bang-bang and bang-off-bang control strategy for inactive and active state constraints. The resulting acceleration and velocity are piecewise linear and quadratic functions.

3-7 The stop and swerve distance definition. The stop distance is always longer than the swerving distance since stopping requires complete dissipation of the vehicle's kinetic energy. A faster speed can be achieved if the vehicle is forced to swerve around obstacles.

3-8 The stop and swerve curve for the planner where the critical sensing range is minimum range required to swerve or stop at a given speed. There are three distinct regions: safe, marginally safe, and inevitable collision state (ICS).

3-9 The collision checking approach using safety certificates. The closest obstacle is calculated when a new trajectory is generated. The trajectory is then re-evaluated at the next possible collision time, which occurs at the edge of the collision-free sphere. This process is repeated until the goal is reached or a collision is detected.

3-10 A linear growth rate for the safe buffer to account for state and obstacle uncertainty. The growth rate and initial size are function of initial uncertainty and dictate how conservative the planner is.

4-1 Quadrotor used for flight experiments. A Hokuyo 2D laser is used for perception. An on-board flight computer runs the perception, planning, and control algorithms developed in the previous chapters.

4-2 The flight computer is a lightweight, small form factor computer that permits running perception, planning, and control algorithms on-board the vehicle. It sends attitude commanded to a custom autopilot that performs attitude control.
The overall software and data flow diagram. The flight computer receives Vicon data over WIFI and laser scan data through USB. The clustering and planning algorithms generates a trajectory that is tracked by the acceleration controller. The attitude commands are sent from the flight computer to the autopilot via a hard wire serial line. The autopilot performs high-rate feedback and estimation.

The Measurable Augmented Reality system provides useful context for robot experiments by displaying perception, planning, and control data in real-time.

Obstacle configuration for Flight 1. The vehicle starts at the top of the image and flies towards a goal at the bottom. The vehicle cannot see the obstacles from its starting location nor are the obstacle locations known a priori.

Commanded and actual path for Flight 1. The vehicle clips the first obstacle so the test was deemed unsuccessful. Note each obstacle has been inflated to account for the vehicle's buffer size.

Lateral position and velocity tracking for Flight 1. The absence of feedforward in Flight 1 lead to poor tracking thus causing the vehicle to clip an obstacle.

Commanded and actual path Flight 2. Note each obstacle has been inflated to account for the vehicle's buffer size. The vehicle successfully reached the goal.

Lateral position and velocity tracking for Flight 2. Feedforward significantly improves tracking performance and is thus responsible for the successful flight test.
Comparison of the feedforward model to the total commanded acceleration and jerk for Flight 2. Acceleration has a normalized error less that 34%. Jerk has a normalized error of 35%. The peak errors in acceleration and jerk coincide with the corners of the jerk command. This is because instantaneous jerk is not physically possible.

Obstacle configuration for Flight 3. The vehicle starts at the top of the image and flies towards a goal at the bottom. The vehicle cannot see the obstacles from its starting location nor are the obstacle locations known a priori.

Commanded and actual path for Flight 3. The vehicle was able to successfully navigate through the obstacle course. Note each obstacle has been inflated to account for the vehicle's buffer size.

Comparison of the feedforward model to the total commanded acceleration and jerk for Flight 3. Acceleration has a normalized error of 31% during the collision avoidance maneuvers. Jerk has a normalized error of 28%. The peak errors in acceleration and jerk coincide with the corners of the jerk command. This is because instantaneous jerk is not physically possible. The large acceleration error at the end of the trajectory is attributed to a damaged motor.

Roll and roll rate tracking for Flight 3. Roll and roll rate are indicators of the aggressiveness of lateral maneuvers. The peak roll and roll rate values are consistent with the acceleration and jerk limits, respectively. The values are larger than reported by other researchers performing similar experiments.

Flight 2 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles.
5-11 Flight 2 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles. .......................................................... 96

5-11 Flight 2 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles. .......................................................... 97

5-12 Flight 3 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles. .......................................................... 98

5-12 Flight 3 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles. .......................................................... 99
List of Tables

2.1  Position Controller Performance Metrics. ........................................... 37
3.1  Planning Algorithm Benchmarking. ....................................................... 70
5.1  Flight Experiments For Minimum-Time Trajectory Planning with Clustering. ................................................................................. 85
Chapter 1

Introduction

1.1 Motivation

Quadrotors have seen a significant rise in their popularity over the past two decades. Advancements in battery, MEMS, and computing technology has expanded the use of quadrotors to more than just aerobatic vehicles for hobbyists and researchers. They are finding applications in areas such as search-and-rescue [1], package delivery [2], and agriculture [3]. Quadrotors are ideal for these applications because they are mechanically simple, can hover, and are easy to operate.

The biggest limitation of current quadrotors is their inability to navigate in cluttered, GPS-denied environments at high-speed. The vehicle must perceive the world, plan within the perceived world, and execute that plan almost instantaneously. Latency in any of the three aforementioned system components will cause the vehicle to travel a significant distance before attempting any collision-avoidance maneuver; increasing the likelihood of collision. Since distance travelled scales linearly with latency and velocity, the negative impact of latency is exacerbated at higher speeds. This makes latency a critical property for all autonomous system.

Each component of the high-speed navigation problem (perception, planning, and control) present unique challenges. First, the perception system must generate an accurate and efficient representation of the world from large volumes of high-rate sensor data. Next, the planning system must produce a collision-free, dynamically
feasible path or trajectory (the distinction between the two will be made shortly) using the world representation from the perception system. Finally, the controller must follow the path or track the trajectory with minimal error.

This thesis investigates the effectiveness of planning collision-free, dynamically feasible trajectories using instantaneous perception data at high-speeds. This perception-planning architecture permits low-latency solutions to the high-speed navigation problem because it bypasses the computationally intensive process of map building. Planning with instantaneous perception data has disadvantages but is beneficial when intelligent, almost instantaneous decisions can be made. Flight experiments demonstrate successful navigation through different cluttered environments at different speeds while only using instantaneous perception data.

1.2 Literature Review

This literature review first clarifies the difference between path following and trajectory tracking and presents the more common path following algorithms and their limitations. It then summarizes the work in trajectory planning for quadrotors in known, static environments and explains why these methods are not suitable for the high-speed navigation domain. Lastly, recent work in high-speed navigation for quadrotors in unknown environments and how their shortcomings are addressed by this work is described.

The term paths and trajectories are commonly used interchangeably in the literature but they have fundamental differences in performance and stability. Trajectories are time-parameterized reference signals to be tracked by a low-level controller [4]. Trajectory generation can be viewed as designing a state and control constrained reference signal. Conversely, a path has no notion of time, it is parameterized by distance or some other metric. Path following has not received the same attention as trajectory generation in the literature but it has been used for navigation in unknown environments so a literature review would be incomplete without it.

Path following is the process of determining a heading command so a vehicle will
track a predefined path. Pure Pursuit [5, 6], originally designed for ground vehicles, is the most common path following algorithm because its simple, stable, and can outperform PID control of cross-track error [7, 8]. Pure Pursuit was expanded to UAV's by Park et al. [8] and shown to be asymptotically stable. Pure Pursuit and its variants use a virtual target point on the path to steer the vehicle onto the path. The distance to the virtual target, known as the lookahead distance, dictates stability and overall performance [6, 7, 8]. Unfortunately, picking the lookahead distance, which is generally a function of vehicle speed, requires extensive testing to achieve adequate performance [9]. Also, encoding state and input constraints into existing algorithms is difficult so the desired path must already be dynamically feasible. Even though path followers can be more robust to external disturbances [4] and outperform trajectory tracking for non-minimum phase systems [10, 11], the lack of stability and performance guarantees for varying speed, the inability to incorporate state and input constraints, and the need for another planner to generate dynamically feasible paths are strong arguments against using these type of algorithms for high-speed navigation in unknown environments. See [4] for a more a detailed review of path following algorithms and how they perform relative to trajectory tracking.

Trajectory planning for quadrotors in known, static environments has received the most attention by researchers. In particular, optimization-based trajectory planners became popular when quadrotors were proved to be differentially flat [12]. Differentially flat systems have the property that all state and control variables can be expressed by flat variables and a finite number of their derivatives [13]. This property allowed researchers to design and optimize over trajectories that satisfied state and input constraints. Mellinger et al. [12] formulated the optimization as a Quadratic Program (QP) that minimized snap (fourth derivative of position) of a ninth order polynomial. The arrival time at pre-selected waypoints was found using gradient descent. Cutler et al. [14] also used a ninth order polynomial and predefined sequence of waypoints to find minimum time, actuator-constrained trajectories for a variable-pitch quadrotor. Richter et al. [15] reformulated the work in [12] to be an unconstrained QP to improve numeric stability. In addition, Richter et al. used a sampling-based
planning algorithm to generate a sequence of waypoints as opposed to hand selection. Hehn et al. [16] found minimum time trajectories using optimal control techniques [17] by decomposing the optimization problem into three smaller optimizations. Only a terminal waypoint was required removing the computational complexity of several waypoint constraints. With the exception of [16], all the aforementioned trajectory planners require intensive computation and are not fast enough to run in real-time without further simplification. Also, each algorithm needs a higher-level planner to select waypoints, which further increases computation time.

With sensors and computers becoming smaller, lighter and cheaper, trajectory planning for quadrotors in unknown, dynamic environments has become the primary focus of the robotics community. A receding horizon approach, adopted from Model Predictive Control [18], is typically used for this domain. Planning with a receding horizon entails designing a trajectory over the planning horizon. The planning horizon can be a fixed-time duration or a fixed distance from the current vehicle position. The trajectory is implemented over a shorter execution horizon and then redesigned with the vehicle’s new state and perception data.

The major technical challenge of receding horizon planning is designing a low-latency planner that efficiently incorporates perception data and a world model. By far the most common approach is to model the world as an occupancy grid map [19, 20], find a discrete path from the robot’s current position to a goal by performing graph-based search on the map using Dijkstra’s algorithm [21] or A* [22], and finally plan a trajectory within the planning horizon using the local result of the graph-based search as waypoints. For instance, Liu et al. [23] used a forward facing RBG-D camera [24] to build an occupancy grid map with the most recent perception data. A minimum-jerk trajectory was designed to pass through waypoints generated by an A* search over the local map. The time allocation between waypoints was found using an approximate method [25] instead of gradient descent as in [12, 15] to achieve real-time performance; 150 milliseconds was required to process the perception data and plan a trajectory. A top speed of 1m/s in simulation and hardware experiments was reported for cluttered environments. Chen et al. [26] presented a novel approach that
finds a corridor within the occupancy grid that guarantees all trajectories within it are collision free. This method was used with a modified version of the minimum snap formulation in [15] and achieved a replan time of less than 60 milliseconds, much faster than reported by Liu et al, though only a top speed of 1.8m/s was achieved. Burri et al. [27] also used the minimum snap formulation from [15] but included velocity and acceleration constraints in the optimization. Trajectory optimization run-times greater than 48ms were reported. All of these methods are much faster than the algorithms previously presented but still have high sensor-to-command latency. In addition, the reported top speeds are considerably slower than what quadrotors can achieve.

Another approach is to use the occupancy grid map as a cost map to evaluate and select a trajectory from a set of pre-computed trajectories known as motion primitives [28]. In general, this architecture is less computationally intensive because a large trajectory library is computed offline (as opposed to the previous methods that generate a single trajectory online). Pivtoraiko et al. [29] is one of the first works to perform this type of replanning on UAV’s. Simulation results of a quadrotor navigating through an unknown cluttered environment were presented. Though a top speed wasn’t specified, a dynamically feasible path was selected in 230 milliseconds, much faster than state-of-the-art planners at the time. It’s important to note that the 230 millisecond runtime does not include the time to process perception data. More recent results by Dey et al. [30] used motion primitives to navigate a quadrotor through a forest using a single camera. Motion primitives were evaluated on a map that contained a short history of the observed world. Keeping a history prevented collisions based on a narrow FOV and allowed emergency backtracking if needed. Unfortunately, no analysis of the required decay rate was presented. Replanning occurred at 5Hz, a similar rate to Liu et al. but slower than Chen et al. Using motion primitives in this domain can drastically decrease the computation load of the planner. However, as the authors of [29] point out, it’s difficult to capture the vehicle’s full capability since primitives are samples from a continuum.

Another technical challenge is guaranteeing safe planning without sacrificing speed.
Intelligent planners will try to avoid or reason about states in which a collision will occur no matter what evasive action is taken. These states are known as an inevitable collision states (ICS) [31]. An example of an ICS is when a vehicle's stopping distance is longer than its sensing range; a collision is inevitable because the vehicle is unable to stop if an obstacle appears in front of it.

There are two common approaches to ensure a vehicle does not enter an ICS. The first is to have all trajectories that extend to the sensing horizon terminate with zero velocity [9, 23, 29]. This ensures that the vehicle can stop if an obstacle is present at the edge of the sensing horizon. It's clear that this approach guarantees safety but it can severely limit top speed if the sensing range is short. In addition, selecting where trajectories end also impacts speed. For instance, Liu et al. had trajectories end at the max sensing range and at frontiers caused by occlusion. Since occlusion frontiers are contained within the max sensing range, the top speed will be low to accommodate trajectories that end within the sensing horizon. The second approach is to plan trajectories only within the known space [9, 23, 30]. This is equivalent to treating unknown space as occupied space. This constraint is very restrictive when planning with instantaneous data from a sensor with limited field of view (FOV). Liu et al. also used this strategy which contributed to the low reported top speed. Dey et al. did not build a globally-consistent map but did keep a short history of the local environment to expand the planning space. As stated previously, Dey et al. presented no analysis as to how long of a history was needed to improve performance. Recent work by Richter et al. [32] has tried to address the shortcomings of the aforementioned strategies by using machine learning to estimate the likelihood of collision in unknown space. Their strategy entails learning features from a training set representative of the operating domain. 80% speed improvement without sacrificing safety in environments with ample training data was reported. Even though the presented results are impressive, this approach is still in its infancy and more experimentation is required before its utility can be fully accessed. Key limitations of this work are acquiring sufficient and representative training data and selecting the appropriate features for accurate collision prediction.
1.3 Contributions

There are three main contributions of this work:

1. An in-depth analysis of using feedforward control for high-speed navigation.

2. A low-latency perception and planning algorithm that generate minimum-time, state and input constrained trajectories using instantaneous perception data.

3. Flight experiments that verify the algorithm's ability to navigate in unknown, cluttered environments at relatively high-speeds.

Chapter 2 presents the low-level position and attitude controller used in this thesis. The cascaded feedback control architecture maps position and velocity error to attitude setpoints for a quaternion-based, nonlinear attitude controller. A closed-form relation between the vehicle's jerk and angular rates is derived. This result is necessary to maintain stability during aggressive maneuvers. A simple analysis, verified by two experiments, demonstrates why feedforward is necessary for high-speed navigation.

Chapter 3 presents the approach taken for perception and planning. Key features are extracted from perception data and used to identify potentially traversable regions. A minimum-time, decoupled optimal control problem is solved so the vehicle can fly towards safe regions. State and obstacle uncertainty is captured in an efficient collision checking method adapted from the sampling-based planning community. Planning safe trajectories is also discussed. Lastly, the computation times for each component of the algorithm is presented.

The hardware and software used in this thesis is presented in Chapter 4. The vehicle uses a flight computer to enable on-board perception, planning, and control. A 2D laser scanner is used for perception. The flight computer interfaces with a custom autopilot that performs attitude control.

Chapter 5 presents flight test results that demonstrate the perception, planning, and control system's ability to navigate through a unknown, cluttered environment. Feedforward is again shown to be necessary to successfully navigate through cluttered
environments. Speed and degree of cluttered is varied to test the system’s performance in different scenarios.

Finally, Chapter 6 presents concluding remarks and discusses future directions for this work.
Chapter 2

Control

2.1 Overview

This chapter presents the low-level position and attitude controller used in this thesis. A review of quaternions and the dynamics of a quadrotor is first presented. Next, a cascaded feedback control architecture that maps position and velocity error to attitude setpoints for a quaternion-based, nonlinear attitude controller is introduced. A closed-form relation between the vehicle's jerk and angular rates is then derived. This result is necessary for high-precision control because it guarantees the commanded attitude and angular rate are dynamically consistent. A simple analysis, validated by two experiments, demonstrating the importance of feedforward in aggressive flight is also provided. Lastly, stability proofs for the attitude and position controllers are presented for completeness.

2.2 Dynamics

2.2.1 Quaternions

Before the equations of motion for a quadrotor are presented, quaternions and their relevant properties are reviewed. Quaternions concisely represent rotations and are commonly used in attitude control because they have none of the singularity prob-
lems associated with Euler angles and are more numerically stable than Rotational Matrices [33].

Quaternions consist of a scalar part $q^0$ and a vector part $\vec{q} \in \mathbb{R}^3$. It can be shown (see [34], for instance) that a quaternion can represent a rotation about the unit vector $\vec{n}$ through an angle $\theta$ by (2.1).$$ q = \begin{bmatrix} q^0 \\ \vec{q} \end{bmatrix} = \begin{bmatrix} \cos \frac{\theta}{2} \\ \sin \frac{\theta}{2} \vec{n} \end{bmatrix} \tag{2.1} $$

This property can be used to rotate a vector in frame $A$ to frame $B$, as shown in (2.2).$$ \begin{bmatrix} 0 \\ \vec{v}^B \end{bmatrix} = q \otimes \begin{bmatrix} 0 \\ \vec{v}^A \end{bmatrix} \otimes q^* \tag{2.2} $$

where $\otimes$ is the quaternion product operator and $q^* = \left[q^0 - \vec{q}^T\right]^T$ is the conjugate of $q$ with the property $q \otimes q^* = 1$.

Quaternion factorization is another useful property, especially for determining attitude errors for feedback. Consider a quaternion $q_{A \rightarrow C}$ that represents the rotation from frame $A$ to frame $C$. It is possible to express $q_{A \rightarrow C}$ as a sequence of rotations through an intermediate frame $B$. This is shown mathematically in (2.3). Note that $q_{B \rightarrow C}$ can be interpreted as the error between $q_{A \rightarrow B}$ and $q_{A \rightarrow C}$. This is a key property to remember for Section 2.3.

$$ q_{A \rightarrow C} = q_{A \rightarrow B} \otimes q_{B \rightarrow C} \tag{2.3} $$

Lastly, the kinodynamic equation relating the rate of change of $q_{A \rightarrow C}$ and the angular rates of frame $A$, $\omega^A$, is shown in (2.4). This property is used extensively in Section 2.4.

$$ \dot{q}_{A \rightarrow C} = \frac{1}{2} q_{A \rightarrow C} \otimes \begin{bmatrix} 0 \\ \omega^A \end{bmatrix} \tag{2.4} $$
2.2.2 Equations of Motion

Let the position of the quadrotor in Fig. 2-1 with mass $m$ and inertia tensor $J$ be described in an inertial frame $I$ by vector $r^I$. The vehicle’s orientation and body angular rates are represented by the quaternion $q_a$ and vector $\omega^B$, respectively. Note that $q_a$ is the rotation from the vehicle’s body frame to the inertial frame.

The quaternion representation of the Newton-Euler equations of motion are given by:

$$\dot{q}_a = \frac{1}{m} q_a \otimes \left[ \begin{array}{c} 0 \\ 0 \\ f^B \\ 0 - \omega^B \end{array} \right]$$

(2.5)

$$\dot{\omega}^B = J^{-1} [M^B - \omega^B \times J \omega^B]$$

(2.6)

where $g = [0 \ 0 \ g]^T$ is the gravity vector, $F^B = [0 \ 0 \ F_{total}]^T$ is the body-frame force vector, and $M^B$ is the body-frame moment vector.

Let $F_i$ be the force produced by motor $i$, $l$ be the distance from the center of mass to motor $i$ (assumed to be the same for each motor), and $c$ be the drag produced by the propellers (again assumed to be the same for each motor), then the total body-$z$
force $F_{total}$ and body-frame moment vector $M^B$ are related to the motor forces by:

$$
\begin{bmatrix}
F_{total} \\
M^B
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & 1 \\
-l & l & -l & l \\
-l & l & -l & l \\
-c & -c & c & c
\end{bmatrix}
\begin{bmatrix}
F_1 \\
F_2 \\
F_3 \\
F_4
\end{bmatrix}
$$

With the left hand side known, (2.7) can be inverted to find the individual motor commands. Observe that (2.7) is a constrained Linear Program since each $F_i$ is constrained to be within the physically bounds of the motor, shown in (2.8). In practice $F_{min} > 0$ since most motors have a minimum throttle (and thus thrust) setpoint to remain spinning.

$$F_{min} \leq F_i \leq F_{max}, \ i = 1, 2, 3, 4$$

2.3 Position and Attitude Control

The attitude dynamics presented in (2.6) typically evolve much faster in time than the position dynamics in (2.5) for quadrotors. This is due to the inertia tensor $J$ being small compared to the mass $m$ so large angular rates can be achieved almost instantaneously. This property permits successive loop closure [35]. Successive loop closure entails having nested feedback loops where the output of one controller serves as the reference for another. An example of this control architecture is shown in Fig. 2-2 where $C_o$ is the outer-loop controller, $C_i$ is the inner-loop controller, and $G$ is the plant. State and input variable limits can be easily handled with this approach but inner/outer loop interactions can degrade performance if not designed carefully [35].

An overview of the controller is presented before the relevant equations for clarity. Fig. 2-3 should be used as a reference for the following discussion. A desired trajectory, generated by a planner, is used as the reference for position and velocity control. Feedforward acceleration and jerk (Section 2.5 describes feedforward and its
successive loop closure example. The output of the outer-loop controller $C_o$ is the input to the inner loop controller $C_i$. The output of the inner loop controller is sent to the plant $G$. importance in more detail) supplement position and velocity feedback to generate a desired attitude, expressed as a quaternion, and angular rate. Using successive loop closure, feedback on the desired attitude and angular rate generated by the outer-loop controller is performed. The quaternion-based inner-loop controller generates motor commands that are sent to vehicle’s speed controllers. The controller was first presented in [36].

The relevant equations for the controller are now presented. First, define the inertial feedback acceleration as:

$$\ddot{\tau}_f = K_p e + K_i \int_0^t e(\tau) \, d\tau + K_d \dot{e}$$

(2.9)

where the error, $e$, and error rate, $\dot{e}$, are the difference between the actual and desired
inertial position and velocity, respectively:

\[ e = r_d^I - r^I \]  \hspace{1cm} (2.10)

\[ \dot{e} = \dot{r}_d^I - \dot{r}^I \]  \hspace{1cm} (2.11)

and \( K_p, K_i, K_d \) are diagonal, positive semi-definite matrices representing the proportional, integral, and derivative gains of the controller.

The inertial feedback acceleration is supplemented with feedforward acceleration \( \dot{r}_d^I \) and gravity \( g \) to produce a desired force in the inertial frame, given by (2.12). The feedforward term \( \dot{r}_d^I \) is important because it acts as a predictive term and thus improves the controller’s transient response.

\[ F' = m (\dot{r}_d^I + \dot{r}_f^I + g') \]  \hspace{1cm} (2.12)

The vehicle must align the body-frame force vector \( F^B \) with the inertial-frame force vector \( F' \) to track the desired trajectory. This is accomplished by applying the quaternion rotation property to \( F^B \) [34]. Let \( \tilde{q}_d \) be the quaternion that accomplishes the necessary rotation. Rearranging (2.5) and normalizing \( F' \) and \( F^B \), the rotation is given by:

\[
\begin{bmatrix}
0 \\
\tilde{F}'
\end{bmatrix} = \tilde{q}_d \otimes \begin{bmatrix}
0 \\
\tilde{F}^B
\end{bmatrix} \otimes \tilde{q}_d^* \]  \hspace{1cm} (2.13)

There are many ways to solve for the desired attitude \( \tilde{q}_d \) in (2.13). The most efficient way for the vehicle to align \( \tilde{F}^B \) and \( \tilde{F}' \) is to use the minimum-angle rotation derived in [37]. The minimum rotation between \( \tilde{F}^B \) and \( \tilde{F}' \) can be shown to be:

\[
\tilde{q}_d = \frac{1}{\sqrt{2 \left(1 + \tilde{F}^B \cdot \tilde{F}'\right)}} \begin{bmatrix}
1 + \tilde{F}^B \cdot \tilde{F}' \\
\tilde{F}^B \times \tilde{F}'
\end{bmatrix} \]  \hspace{1cm} (2.14)

Yaw is a differentially flat variable meaning that it can be independently chosen [12]. Hence, the minimum-angle rotation quaternion in (2.14) must be composed with a quaternion that represents the desired rotation about the body z-axis.
full desired quaternion \(q_d\) is shown in (2.15) where \(\psi_d\) is the desired yaw angle.

\[
q_d = \tilde{q}_d \otimes \begin{bmatrix} \cos(\psi_d/2) & 0 & 0 & \sin(\psi_d/2) \end{bmatrix}^T
\]  

(2.15)

There are different ways of generating desired angular rate, some of which are only valid for small attitude angles [14] or require hand-tuning [38]. Our method is suspended until Section 2.4 where a more detailed analysis is presented. Assume for the rest of this section that a desired angular rate is known.

With a desired attitude and angular rate known, error terms can be generated and used for feedback control. Using the composition property of quaternions [34], the desired and actual quaternion can be related by an error quaternion \(q_e\). This relation is given by:

\[
q_d = q_a \otimes q_e
\]

(2.16)

which can be rearranged to:

\[
q_e = q_a^* \otimes q_d
\]

(2.17)

It is important to understand which frame the quaternions in (2.17) are expressed in. The desired and actual attitude, \(q_d\) and \(q_a\) respectively, are expressed in the inertial frame where the error quaternion \(q_e\) is in the body frame. Hence, the error quaternion represents the necessary rotation in the body frame to achieve the desired attitude. With this formulation, a direct relation between the error quaternion and body frame moments exists. Similar to other quaternion-based attitude control laws [33, 35, 39], the required body-frame moments are achieved using proportional-derivative control on the attitude and rate errors:

\[
M^B = \text{sgn} (q_e^0) K_p q_e^\tau + K_d (\omega_d^B - \omega^B)
\]

(2.18)
where $q^0_e$ and $\bar{q}^e_e$ are the real and vector part of the error quaternion. The gains $K_p$ and $K_d$ are diagonal, positive definite matrices.

### 2.4 Dynamically Consistent Angular Rates

One result of this work is how desired angular rates are generated. A desired angular rate must be dynamically consistent with the desired attitude for the controller in (2.18) to maintain stability during large attitude, fast angular rate maneuvers; the type of maneuvers required for high-speed flight. The approach taken by [36] used the Transport Theorem [40] to generate desired angular rates. This procedure is only valid for small attitudes and can lead to instability for large attitude commands. [38] related the desired angular rates to the attitude errors through a proportional gain. A hand-tuned gain requires excess experimentation and no definitive right choice exists. The result presented here is not limited to small angles and requires no hand tuning.

The desired attitude rate is found by differentiating (2.13). Recall $\hat{F}^B = [0 \ 0 \ 1]^T$ so $\dot{\hat{F}}^B = 0$. Applying the chain rule to (2.13) leads to:

$$
\begin{bmatrix}
0 \\
\dot{\hat{F}}^l
\end{bmatrix} = \dot{\hat{q}}_d \otimes \begin{bmatrix}
0 \\
\hat{F}^B
\end{bmatrix} \otimes \bar{q}^*_d + \bar{q}_d \otimes \begin{bmatrix}
0 \\
\hat{F}^B
\end{bmatrix} \otimes \bar{q}^*_d
$$

(2.19)

where

$$
\dot{\hat{q}}_d = \frac{1}{2} \bar{q}^*_d \otimes \begin{bmatrix}
0 \\
\omega_d
\end{bmatrix}
$$

(2.20)

$$
\dot{\bar{q}}^*_d = -\frac{1}{2} \begin{bmatrix}
0 \\
\omega_d
\end{bmatrix} \otimes \bar{q}^*_d
$$

(2.21)

and $\omega_d = [\omega_{zd} \ \omega_{yd} \ \omega_{zd}]^T$ represents the desired body angular rates. Note that the superscript $B$ has been dropped for clarity. Substituting (2.20) and (2.21) into (2.19)
and using the distributive property of quaternions [34],

\[
\begin{bmatrix}
0 \\
\omega_d
\end{bmatrix} = \frac{1}{2} \tilde{q}_d \otimes \left( \begin{bmatrix}
0 \\
\omega_d
\end{bmatrix} \otimes \begin{bmatrix} 0 \\ \hat{F}^B \end{bmatrix} - \begin{bmatrix} 0 \\ \hat{F}^B \end{bmatrix} \otimes \begin{bmatrix} 0 \\
\omega_d
\end{bmatrix} \right) \otimes \tilde{q}_d^* \tag{2.22}
\]

The two quaternion products in parenthesis are given by:

\[
\begin{bmatrix}
0 \\
\omega_d
\end{bmatrix} \otimes \begin{bmatrix} 0 \\ \hat{F}^B \end{bmatrix} = \begin{bmatrix}
-\omega_d \cdot \hat{F}^B \\
\omega_d \times \hat{F}^B
\end{bmatrix}, \tag{2.23}
\]

\[
\begin{bmatrix} 0 \\ \hat{F}^B \end{bmatrix} \otimes \begin{bmatrix} 0 \\
\omega_d
\end{bmatrix} = \begin{bmatrix}
-\omega_d \cdot \hat{F}^B \\
-\omega_d \times \hat{F}^B
\end{bmatrix} \tag{2.24}
\]

Taking the difference of (2.23)-(2.24) and substituting the result into (2.22) gives

\[
\begin{bmatrix}
0 \\
\hat{F}^I
\end{bmatrix} = \frac{1}{2} \tilde{q}_d \otimes \left( \begin{bmatrix} 0 \\ 2\omega_d \times \hat{F}^B \end{bmatrix} \otimes \tilde{q}_d^* \right) \tag{2.25}
\]

The cross product can be expressed as a skew-symmetric matrix and vector multiplication, rearranging (2.25) leads to:

\[
\tilde{q}_d \otimes \left( \begin{bmatrix} 0 \\
\hat{F}^I
\end{bmatrix} \right) \otimes \tilde{q}_d = \begin{bmatrix}
0 & -\omega_{zd} & -\omega_{yd} & -\omega_{zd} \\
\omega_{zd} & 0 & \omega_{yd} & -\omega_{yd} \\
\omega_{yd} & -\omega_{zd} & 0 & \omega_{zd} \\
\omega_{zd} & \omega_{yd} & -\omega_{zd} & 0
\end{bmatrix} \begin{bmatrix} 0 \\ \hat{F}^B \end{bmatrix}. \tag{2.26}
\]

Since \( \hat{F}^B = [0 \ 0 \ 1]^T \), (2.26) becomes:

\[
\tilde{q}_d \otimes \left( \begin{bmatrix} 0 \\
\hat{F}^I
\end{bmatrix} \right) \otimes \tilde{q}_d = \begin{bmatrix}
-\omega_{zd} \\
-\omega_{yd} \\
\omega_{zd} \\
0
\end{bmatrix} \begin{bmatrix} 0 \\ -\omega_{yd} \\
\omega_{zd} \\
0
\end{bmatrix} = \begin{bmatrix} 0 \\ -\omega_{yd} \\
\omega_{zd} \\
0
\end{bmatrix}. \tag{2.27}
\]

Equation (2.27) relates the inertial jerk vector to angular rates through a quaternion.
rotation. The left-hand side of the equation will always have a zero real part so the above derivation will always produce $\omega_{zd} = 0$. As a result, $\omega_{zd}$ can be independently chosen.

The inertial jerk vector is calculated as follows [14]:

$$\dot{\mathbf{F}} = \frac{\mathbf{F}' - \mathbf{F} \cdot \left( \mathbf{F}' \cdot \mathbf{F}' \right)}{\| \mathbf{F}' \|^3}$$

(2.28)

where $\dot{\mathbf{F}} = m \left( \ddot{r}_d + \dot{r}_{fb} \right)$. The feedback jerk $\dot{r}_{fb}$ is found by numerically differentiating the feedback acceleration in (2.9).

### 2.5 Feedforward

#### 2.5.1 Motivation

Navigating in unknown environments at high-speeds requires very precise control; any tracking error can have catastrophic consequences. Unfortunately, there is an inherent delay in feedback control because there must be error for the controller to be active. A common approach to improve tracking performance is by using feedforward. Feedforward predicts the necessary plant input to achieve a desired output. The most simple form of feedforward is model inversion - the model of the plant is inverted and used to supplement the controller [41]. If an accurate model is available, feedforward can drive tracking error to zero. To see this, consider the block diagram in Fig. 2-4 where $C$ is the controller, $G$ is the plant, and $F$ is the reference feedforward model. It can be shown that the tracking error $e$ can be written as:

$$e = \left( 1 - \frac{FG}{1 + CG} \right) r$$

(2.29)

where $r$ is the reference signal and $y$ is the plant output. By carefully picking $F$ the right hand side of (2.29) can be made to be identically zero. For instance, $F = G^{-1}$ accomplishes this.

This simple analysis shows that zero tracking error can be accomplished through
model inversion. Unfortunately it is rarely the case that a perfect model is available to achieve perfect tracking. Nonetheless, a simplified version of the dynamics presented in Section 2.2 can be used to improve overall performance. Again using the fact that the attitude dynamics are much faster than position dynamics, we can approximate (2.5) as a double integrator system. This implies that feedforward model inversion for a quadrotor is equivalent to adding acceleration to the position controller. This is not completely correct, however. Recalling the result from Section 2.4, jerk is also required so the attitude and angular rate commands are consistent. Thus a combination of a double and triple integrator model is equivalent to model inversion for a quadrotor.

2.5.2 Experimental Verification

The analysis in Section 2.5.1 proved that perfect reference tracking can be achieved through model inversion if a perfect model of the system is known. However, this is rarely the case in physical systems. This subsection presents experimental verification of two feedforward models by showing significant improvement in reference tracking.

Geometry-Based Model

The first model experimentally verified used the geometric properties of a desired path as feedforward. This model is useful when a dynamically feasible path is already
available and can supplement a path following algorithm. Though the limitations of path following were already highlighted in Section 1.2, this is still a useful result worth presenting.

Consider a curve $C$, shown in Fig. 2-5, that is parameterized by variable $s$ with radius of curvature $\kappa$.

![Diagram of curve C with variables](image)

Figure 2-5: A curve $C$ parameterized by variable $s$. The acceleration and jerk required to traverse $C$ are known functions of the speed $v$, curvature $\kappa$, and rate of change of curvature $\kappa'$. These can be used for feedforward.

The acceleration $a_{ff}$ and jerk $j_{ff}$ to traverse curve $C$, expressed in $n-t$ coordinates, can be shown to be:

$$a_{ff} = (a_t) e_t + (\kappa v^2 + C_d v^2) e_n \tag{2.30}$$
$$j_{ff} = (j_t - \kappa^2 v^3 + 2C_D v a_t) e_t + (\kappa v^3 + C_d \kappa v^3 + \kappa v a_t) e_n \tag{2.31}$$

where $v$ is the vehicle's speed, $C_d$ is the vehicle's drag coefficient, and $\kappa'$ is the rate of change of curvature with respect to $s$, and $a_t$ and $j_t$ are the tangential acceleration and jerk of the speed profile. $C_d$ is typically found through wind-tunnel testing or through steady-state straight flight tests. Compensating for drag is especially important at high speeds.

The above feedforward model was tested by having a quadrotor fly in a 1m radius circle at $4m/s$ (see Chapter 4 for experimental setup). Since traversing a circle is equivalent to tracking a sinusoid in inertial $x$ and $y$ coordinates, the performance metrics used to evaluate the model were amplitude error and phase lag. Table 2.1 shows the performance metrics for three different cases: no feedforward, geometric
feedforward (drag not included), and the complete feedforward model. The complete feedforward model has the lowest amplitude error and phase lag, as desired. Fig. 2-6 shows the real-time performance of the vehicle with and without feedforward. Using Fig. 2-6a as a legend, Fig. 2-6b illustrates the poor performance of pure feedback. The lag between the commanded and actual position is concerning since any lag in responsiveness only increases the likelihood of collision. Adding feedforward, shown in Fig. 2-6c, essentially eliminates lag.

Table 2.1: Position Controller Performance Metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>No FF</th>
<th>Geometric FF</th>
<th>Geom. and Drag FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amp. Error (%)</td>
<td>7</td>
<td>-8</td>
<td>-4</td>
</tr>
<tr>
<td>Phase Lag (°)</td>
<td>40</td>
<td>15</td>
<td>4.6</td>
</tr>
</tbody>
</table>

**Time-Based Model**

The second experimental verification used the acceleration and jerk from a desired trajectory for feedforward. Section 3.4 discusses the trajectory generation process in detail. For now, a state and control input constrained minimum-time trajectory is generated with jerk as the input.

In this set of experiments, the vehicle was commanded to accelerate in a straight line to a top speed of 3m/s with and without feedforward. Fig. 2-7 shows the position and velocity tracking for both experiments. The position and velocity tracking without feedforward, Fig. 2-7a and Fig. 2-7b respectively, show very poor transient performance: a 250ms and 150ms delay before the vehicle responds is observed for position and velocity, respectively. Adding feedforward, Fig. 2-7c and Fig. 2-7d, reduces the delay to 50ms for velocity and is negligible for position. The overshoot in Fig. 2-7d indicates the controller gains must be retuned if feedforward is used. Lastly, there is a steady-state error in position for both cases. This is attributed to drag acting on the vehicle and can be eliminated if drag is used in the feedforward model or by increasing the integral gain.

Fig. 2-7 showed that feedforward significantly improved position and velocity tracking. One metric to quantify the feedforward model is comparing the feedfor-
Figure 2-6: Real-time visualization of relevant experimental data using the Measurable Augmented Reality (MARS) system. The feedforward model significantly improves tracking performance by decreasing amplitude error and phase lag. The control effort, measured by throttle, is also lowered.
ward command to the total acceleration and jerk command. The total command in this context is the sum of the feedforward and feedback commands. Fig. 2-8 shows how the feedforward and total command compare. Qualitatively, the feedforward acceleration and jerk in Fig. 2-8a and Fig. 2-8b, respectively, are in good agreement with the total command. The difference between the feedforward and total command, normalized by the max total command, is shown in Fig. 2-8c for acceleration and Fig. 2-8d for jerk. The acceleration error is less than 18% in the beginning of the acceleration phase. The error grows, however, to 38% in the latter stage of the acceleration phase. This is attributed to the large overshoot in velocity, which can be removed by retuning the controller. The jerk error is less than 30% for the entire acceleration phase. Since instantaneous changes in jerk are not physically possible, the peak errors following large changes in the feedforward command is logical.
Figure 2-8: Comparison of the feedforward model to the total commanded acceleration and jerk. Acceleration has a normalized error less than 18% in the beginning stage but grows to 38% because of velocity overshoot in the latter stage of the acceleration phase. Jerk has a normalized error less than 30%. This is small enough to validate the instantaneous jerk assumption.

2.6 Stability

2.6.1 Attitude Controller

Stability analysis for the attitude controller was originally presented in [33] where feedback for large angle maneuvers was studied. The below derivation follows closely to that presented in [39] and uses Lyapunov’s direct method [42] to prove stability.

Recall the quaternion kinematics and attitude dynamics presented in Section 2.2:

\[
\dot{q} = \frac{1}{2} q \otimes \begin{bmatrix} 0 \\ \omega^B \end{bmatrix} 
\]  

\[
\omega^B = J^{-1} \left[ M^B - \omega^B \times J \omega^B \right] 
\]

40
where

\[ M^B = \text{sgn} (q^0) K_p \ddot{q} - K_d \dot{\omega}^B \tag{2.18} \]

is the controller. We want to show that (2.18) stabilizes the system of equations (2.4) and (2.6) for all possible states. This is accomplished by finding a scalar function \( V(x) \) such that \( V(x) > 0 \ \forall x \neq 0 \) and \( \dot{V}(x) \leq 0 \ \forall x \neq 0 \) [42]. This function is known as a Lyapunov function.

Consider the candidate Lyapunov function:

\[ V(q, \omega^B) = \frac{1}{2} \dot{q} \cdot K_p \ddot{q} \ + \frac{1}{2} \omega^B \cdot J \omega^B \tag{2.32} \]

which is strictly positive except for \( q = \omega^B = 0 \). This point corresponds to the body at rest and at the identity rotation. Taking the time derivative of (2.32),

\[ \dot{V}(q, \omega^B) = \dot{q} \cdot K_p \ddot{q} \ + \omega^B \cdot J \dot{\omega}^B \tag{2.33} \]

Using (2.4) and (2.6),

\[ \dot{V} = \frac{1}{2} \dot{q} \cdot K_p \left( q^0 \omega^B + \dot{q} \times \omega^B \right) - \omega^B \cdot \left( \omega^B \times J + \frac{q^0}{2} K_p \ddot{q} + K_d \omega^B \right) \tag{2.34} \]

\[ = -\omega^B \cdot K_d \omega^B \leq 0 \tag{2.35} \]

Thus the equilibrium point \((q, \omega)\) that corresponds to the body at rest at the identity rotation is globally asymptotically stable, as desired.

### 2.6.2 Position Controller

In this section, stability of the position controller is proven by again employing Lyapunov's direct method. The original derivation was presented in [36] and is included here for completeness. The below analysis follows closely the framework of approximate-model-inversion based control (e.g. [43]).
The true quadrotor dynamics can be expressed as:

\[ \ddot{\mathbf{r}}^l = f(\mathbf{r}^l, \dot{\mathbf{r}}^l, \delta) \]  
(2.36)

where \( \delta \) is the input to the attitude controller. Note the time argument of all the above variables is omitted for clarity. Since \( f \) is unknown, we assume an approximate model \( \hat{f}(\mathbf{r}^l, \dot{\mathbf{r}}^l, \delta) \) is available and is invertible with respect to \( \delta \):

\[ \delta = \hat{f}^{-1}(\mathbf{r}^l, \dot{\mathbf{r}}^l, \nu) \]  
(2.37)

where \( \nu \) is the control acceleration. The function \( \hat{f}^{-1} \) maps the control acceleration \( \nu \) to the attitude and attitude rate commands for the attitude controller. The construction of \( \hat{f}^{-1} \) entails using (2.9)-(2.15) and (2.27)-(2.28).

The control acceleration encompasses both feedforward and feedback acceleration:

\[ \nu = K_p e + K_d \dot{e} + \ddot{\mathbf{r}}_d^l + \nu_i \]  
(2.38)

where \( \nu_i \) is the integral term of the feedback controller.

The modeling error between the true and approximate dynamics is:

\[ \Delta = \hat{f}(\mathbf{r}^l, \dot{\mathbf{r}}^l, \delta) - f(\mathbf{r}^l, \dot{\mathbf{r}}^l, \delta) \]  
(2.39)

It is assumed that the modeling error does not change with time, \( \dot{\Delta} = 0 \). In practice this is a valid assumption for flights where battery voltage changes very little. Substituting (2.39) into (2.36), the error tracking dynamics become:

\[ \ddot{e} = \ddot{\mathbf{r}}_d^l - \dot{\mathbf{r}}^l = \ddot{\mathbf{r}}_d^l - \dot{f}(\mathbf{r}^l, \dot{\mathbf{r}}^l, \delta) + \Delta \]  
(2.40)

\[ = \ddot{\mathbf{r}}_d^l - K_p e - K_d \dot{e} - \ddot{\mathbf{r}}_d^l - \nu_i + \Delta \]  
(2.41)

\[ = -K_p e - K_d \dot{e} - \nu_i + \Delta \]  
(2.42)
Putting into state-space form,

\[
\dot{\mathbf{e}} = \begin{bmatrix}
\dot{\mathbf{e}} \\
\dot{\mathbf{e}}
\end{bmatrix} = \begin{bmatrix}
0 & \mathbf{I} \\
-\mathbf{K}_p & -\mathbf{K}_d
\end{bmatrix} \begin{bmatrix}
\mathbf{e} \\
\mathbf{e}
\end{bmatrix} + \begin{bmatrix}
0 \\
\mathbf{I}
\end{bmatrix} (\Delta - \nu_i) \tag{2.43}
\]

\[
= A\mathbf{e} + B (\Delta - \nu_i) \tag{2.44}
\]

where \(\hat{e}\) is the tracking error. The gains \(\mathbf{K}_p\) and \(\mathbf{K}_d\) are chosen such that \(A\) is Hurwitz. Hence, for any positive definite matrix \(Q\), there exists a unique positive definite matrix \(P\) that is a solution to the Lyapunov equation:

\[
0 = A^T P + PA + Q \tag{2.45}
\]

Let the derivative of the integral term \(\nu_i\) be defined as \(\dot{\nu}_i = B^T P \hat{e}\). In addition, let \(\hat{\nu}_i = \Delta - \nu_i\) be the error between the modeling error and integral term. We want to show the controller drives \(\hat{\nu}_i \to 0\).

A Lyapunov function candidate is:

\[
V = \frac{1}{2} \hat{\mathbf{e}}^T P \hat{\mathbf{e}} + \frac{1}{2} \hat{\nu}_i^T \hat{\nu}_i \tag{2.46}
\]

where \(V \geq 0\), and \(V = 0\) when \(\hat{e} = \hat{\nu}_i = 0\). Taking the time derivative of (2.46),

\[
\dot{V} = \frac{1}{2} \hat{\mathbf{e}}^T P \dot{\hat{\mathbf{e}}} + \frac{1}{2} \hat{\mathbf{e}}^T P \hat{\mathbf{e}} + \hat{\nu}_i^T \dot{\hat{\nu}}_i
\]

\[
= \frac{1}{2} \hat{\mathbf{e}}^T P (A \hat{\mathbf{e}} + B (\Delta - \nu_i)) + \frac{1}{2} (A \hat{\mathbf{e}} + B (\Delta - \nu_i))^T P \hat{\mathbf{e}} - \hat{\nu}_i^T B^T P \hat{\mathbf{e}} \tag{2.47}
\]

\[
= -\frac{1}{2} \hat{\mathbf{e}}^T Q \hat{\mathbf{e}} \leq 0 \tag{2.48}
\]

The controller drives \(\hat{e} \to 0\) as \(t \to 0\) and is therefore asymptotically stable, as desired.

### 2.7 Summary

This chapter presented the low-level position and attitude controller used in this thesis. The equations of motion for a quadrotor were summarized. The cascaded feed-
back control architecture that maps position and velocity error to attitude setpoints for a quaternion-based, nonlinear attitude controller was introduced. A closed-form relation between the vehicle's jerk and angular rates was then derived. This result is necessary to maintain stability and improve the controller's performance during large attitude, fast angular rate maneuvers. Substantial improvement in the vehicle's transient response to commands when feedforward is used was then demonstrated. Lastly, stability for the attitude and position controller was proven.
Chapter 3

Perception and Planning

3.1 Overview

This chapter presents the approach taken for perception and planning. First, the system architecture typically used for navigation is reviewed and its limitations are discussed. How this work addresses the aforementioned limitations is then highlighted. The process of representing the world by extracting key features from perception data and identifying traversable regions via clustering is then presented. Next, the design requirements for a low-latency trajectory planner and the basis for using a third-order, decoupled kinematic model is described. Trajectory generation is then formulated as a minimum-time optimal control problem. Relaxing the final position constraint leads to a closed form solution, which is derived for arbitrary initial and final conditions. Following, an efficient collision checking method using "safety certificates" is presented. Later, safe planning and how some domain knowledge can substantially increase the vehicle's top speed is displayed. Finally, the low computation times of the key components of the perception and planning algorithm is demonstrated.

3.2 System Architecture

There are a number of domain-specific requirements that must be considered when selecting a system architecture for an autonomous vehicle. Typical questions that
arise during preliminary system design are which sensors to use, how to transform sensor data into a representation of the world, and how to safely plan within the known world.

By far the most common system architecture used in robotics is shown in Fig. 3-1. Perception data, from a lidar, camera, or RGB-D camera, is used to build a map. Many researches use occupancy grid maps [19] to represent the world. Octomap [20] is an efficient type of occupancy grid map that uses a probabilistic model to represent occupancy and is the current standard in robotics. The resulting map is typically used for local collision avoidance and global planning (e.g. general desired heading).

Once the map is updated with new perception data, a graph-based search is performed; typically with A* or Dijkstra's algorithm. The grid cells in the octomap are represented as nodes in the graph. Each node has a user-defined cost that typically consists of distance to closest obstacle and total distance travelled. The nodes selected by the graph-based search algorithm are used as waypoints to generate a trajectory, such as a minimum snap trajectory or some other polynomial-based trajectory.

Figure 3-1: Typical system architecture for navigating in unknown environments. Map building and trajectory generation are the primary contributors to system latency. High-speed navigation requires low-latency perception and planning.

The demands of high-speed navigation raise a number of issues with the above system diagram. For instance, updating and querying the map can take upwards of tens to hundreds of milliseconds in practice. This latency is further exacerbated when the map is very large. Keeping a local map for collision avoidance, instead of using a global map, alleviates this problem but requires a two-map architecture: a local map for collision avoidance and a global map for desired heading information. This places additional computational burden on an already limited \(^1\) flight computer.

The graph-based search block in Fig. 3-1 typically runs fast for small maps in 2D.

\(^1\)Light-weight flight computers for aerial vehicles have limited computation capabilities
However, graph-based search scales poorly for 3D paths and for large maps. This is because $A^*$ and Dijkstra's scale like $O(|V|)$ and $O(|V|\log|V|)$, respectively, where $V$ is the number of nodes in the graph [22].

The trajectory generation block is responsible for planning a dynamically-feasible, collision-free trajectory through the waypoints produced by the graph-based search. Collision checking and dynamic feasibility are computationally intensive so this block is a large contributor to system latency. Typical run-times of state of the art algorithms are between 60-100ms, as presented in Section 1.2.

The above issues are addressed by modifying the blocks in Fig. 3-1 so that low-latency solutions for high-speed navigation are possible. The new system diagram is shown in Fig. 3-2. The need to build a dense local/global map for collision avoidance is removed by working with instantaneous sensor data. Sensor data, in the form of a laser scan or pointcloud, is clustered into regions with consistent depth measurements. The resulting clusters are sorted based on a user-defined cost. A velocity plan (to be defined later) is iteratively generated for the list of sorted clusters until a collision free path is found. The rest of this chapter discusses each block in Fig. 3-2 in detail.

![Figure 3-2](image)

Figure 3-2: The proposed system architecture for navigating in unknown environments. Obstacle clustering efficiently processes instantaneous data and removes the need to build a local map. Traversable regions are identified and a velocity plan is generated to avoid obstacles.

### 3.3 Obstacle Clustering

The computational demands of updating and querying a map inhibit low-latency trajectory planning and collision avoidance. An alternative to mapping is to plan with instantaneous perception data. Working with instantaneous data, generally in
the form of a pointcloud or laser scan, can significantly reduce latency but does present different challenges. For instance, planners using sensors with limited field-of-view forget previously seen obstacles. Two common solutions for this are to plan only within the field-of-view or to keep a short history of sensor data. The former is typically what’s done but can be very restrictive for limited field-of-view sensors. The latter is essentially mapping and defeats the purpose of using instantaneous data.

In this work, a 2D laser scanner (see Chapter 4) was used for perception. As the name suggests, the output of the sensor is depth measurements for a 2D slice of the world. Hence, no information for above and below the sensor is available. An example of a laser scan with obstacles is shown in Fig. 3-3a. The horizontal field-of-view of a laser scanner is much larger than that of RGB-D and standard cameras though at the expense of having no vertical field-of-view. This removes the ability to fly in 3D but lasers typically have a farther sensing range than state-of-the-art camera systems allowing for faster flight. In addition, the computational load to acquire and process laser scan data is significantly less.

The fundamental question for planning with instantaneous data and without a map is how to represent and plan in the surrounding environment. The goal is to identify traversable regions and concisely represent obstacles in the perception data. The approach taken here entails extracting key features from the perception data and using those features to plan trajectories. To be more specific, parts of the scan that strike obstacles, like those in Fig. 3-3a, are represented as a set clusters. The size of the obstacle dictates how many cluster are used to represent it. An example of the clustering approach is shown in Fig. 3-3b. Only a single cluster is used for the small obstacles to the left of vehicle while more clusters are used for the larger obstacles to the right. Two clusters are used to capture the boundary of larger obstacles. If the obstacle is very large, addition clusters are placed within the interior of the two boundary clusters. The remaining parts of the scan, which are potentially traversable regions, are also represented by clusters. The additional clusters are not shown in Fig. 3-3b for clarity. Note that the final goal location is always included in the set of clusters. Also, the clustering is performed every time new perception data is available.
Figure 3-3: Key features from the laser scan data, such as obstacles and free space, are represented by a set of clusters. The size of the obstacle and free space dictate how many clusters are used to represent them. Clusters are always placed at the boundary of large obstacles.
The goal of clustering is to identify traversable regions and concisely represent obstacles. The next step is to select the cluster the vehicle should fly towards. This is done by finding the cluster that minimizes a cost function. The cost function must consider the effort required to transition to a new cluster, distance to the cluster, and progress to the goal. The first term prioritizes clusters that are in the same general direction as the previous selected cluster. This prevents performing expensive collision checking on clusters that are likely to lead to a collision. The second term prevents the planner from hugging obstacles since preference is given to clusters farther away. The last term ensures the vehicle makes progress to the final goal.

The cost for cluster $i$ is given by (3.1) where $\phi_{last,i}$ is the angle difference between the last selected cluster and cluster $i$, $\theta_{goal,i}$ is the angle difference between the final goal and cluster $i$, and $r_i$ is the distance to cluster $i$. This is also shown pictorially in Fig. 3-4. The cost can be broken up into two components: a stage cost and a terminal cost. The stage cost is responsible for prioritizing clusters that are likely collision free while the terminal cost guides the vehicle to the goal. The gains $k_1$, $k_2$, and $k_3$ are hand-tuned for desired performance. In practice, $k_1 = k_3 = 2k_2$ was found to work best. Note that if a cluster is closer than a specified distance, it is removed entirely from the cluster list.

$$J_i = k_1 \phi_{last,i} + k_2 \theta_{goal,i} + k_3 \left( \frac{1}{r_i} \right)$$  

After sorting the cluster by the cost in (3.1), a dynamically-feasible trajectory is generated in the direction of each cluster until a collision-free path is found. The next section discusses the trajectory generation process in detail.

### 3.4 Constrained Decoupled Velocity Planning

Section 3.3 presented a method that represents potentially traversable regions in raw perception data as clusters. The clusters are then used as possible headings for the vehicle's velocity vector. This entire process can be loosely interpreted as
Figure 3-4: Illustration of the cost of each cluster. $\phi_{\text{last}}$ is used to prioritize clusters that are in the same general direction as the last selected cluster. $\theta_{\text{goal}}$ and $r$ are used to guide the vehicle to the goal and prevent obstacle hugging.

Defining design requirements is a critical step in developing any new technology because it aids in identifying technical challenges. The first requirement for any planner is to incorporate state and control input constraints to ensure the trajectory is dynamically feasible. The second requirement is to accurately model the vehicle’s dynamics so the desired trajectory is representative of the vehicle’s capabilities. This is especially important so the collision checking on the desired trajectory is valid for the true trajectory. The third is to include state and obstacle uncertainty so the planner can make intelligent decisions and not be overly conservative. Lastly, the vehicle must never enter a state where a collision is inevitable. The design requirements for
the this work are summarized below.

**Design requirements**

- Incorporate state and control input constraints
- Accurately model the vehicle’s dynamics for collision checking
- Incorporate state and obstacle uncertainty
- Guarantee a state of inevitable collision is never entered

It is also constructive to outline the assumptions made in this work. The first of which is that a desired heading is always available. This is a necessary assumption because the vehicle needs to have some sense of direction. This assumption is less restrictive than range and bearing to a goal so it is reasonable. The second assumption is that a velocity estimate is always available. The controller presented in Chapter 2 needs at minimum an accurate velocity estimate to perform feedback. A local (as opposed to global) position estimate is also assumed available. Control strictly based on velocity is possible in practice but the controller in Chapter 2 is position-driven. As a result, it’s performance increases if position information is available. The availability of a velocity and local position estimate is reasonable with current visual-inertial odometry algorithms. The last assumption is that there are no dead-ends in the environment. The planner developed in this thesis is strictly responsible for collision avoidance. If a dead-end is present, a global planner that has access to a global map is required. The assumptions are summarized below.

**Assumptions**

- Heading to the goal is always available
- An accurate velocity estimate is available
- A local estimate of position is available
- No dead-ends
3.4.1 Modeling State and Actuator Constraints

Trajectory planners must be cognizant of state and input constraints to maximize performance. Correctly modeling and incorporating constraints is especially important for planning in real-time because accurate predictions of the vehicle's behavior is needed to check for collisions. For a quadrotor, control input constraints come in the form of motor thrust limitations. State constraints arise from mission requirements, such as maximum speed and acceleration or waypoint constraints, and from sensor limitation, maximum allowable jerk to prevent camera blur, for instance. The rest of this subsection discusses how previous methods included state and control input constraints and why the presented method is more suitable for low-latency planning.

Previous works have relied heavily on the differential flatness property of quadrotors to check control input constraints. For instance, [12, 14, 15] iteratively modified trajectories until the actuator constraints were satisfied. To illustrate their approach, let a desired trajectory $r_d$ be known. Recalling the equations presented in Section 2.2.2, the thrust and body moments to follow a trajectory can be written as:

$$F_{\text{total}} = ||F'|| = m ||\ddot{r}_d||$$
$$M_B = J\dot{\omega}_d^B + \omega_d^B \times J\omega_d^B$$

As shown in Section 2.4, the angular rates $\omega_d^B$ can be written as a function of jerk $\dddot{r}_d$ and acceleration $\ddot{r}_d$. The angular acceleration $\dot{\omega}_d^B$ can be found in a similar manner. Hence (3.2) and (3.3) can be rewritten as:

$$F_{\text{total}} = h_1(r_d, \dot{r}_d, \ddot{r}_d, \ldots)$$
$$M_B = h_2(r_d, \dot{r}_d, \ddot{r}_d, \ldots)$$

where $h_1(\cdot)$ and $h_2(\cdot)$ are nonlinear function of the desired trajectory and it's derivatives. The individual motor commands can then be found by substituting (3.4) and (3.5) into (2.7) and inverting. If a constraint is violated, $r_d$ is modified and the procedure is repeated.
More recent work in [23, 27, 26] has expanded the aforementioned work to include state constraint checking along the entire trajectory through a similar process. The key limitation of using the differential flatness approach is that the equations of motion must be forward simulated in time to verify state and input constraints are satisfied. In addition, forward simulation has to be done in an iterative manner to correct the desired trajectory. This is very computationally intensive and slow, making it not the ideal way to incorporate state and control input constraints for low-latency planning.

A different approach is to model state and input constraints as kinematic constraints. This approach has a long-standing history in the optimal control community and a number of closed form solutions exist (e.g. [45, 46]). Kinematic constraints are also far more intuitive and easier to model than actuator constraints. For instance, it is far easier for an engineer to model the deceleration characteristics of a car as an acceleration constraint than to include braking dynamics in the model.

Modelling state constraints as kinematic constraints is usually trivial since state constraints are kinematic in nature. Conversely, modelling actuator constraints as kinematic constraints can be challenging. Either a closed form relation between the two constraint models exists or approximations have to be made. To be more precise, functions $l_i(\cdot)$ for $i = 1, \cdots, n$ that relate motor forces $F_j$, $j = 1, \cdots, 4$ to acceleration, jerk, snap, $\cdots$, $n + 2$ derivative of position must be found. This is mathematically shown in (3.6)-(3.7) where $a_k$ and $j_k$ are the acceleration and jerk along the $k$th axis. Alternatively, lower bounds on $l_i(\cdot)$, such as $a_{\max}$, $j_{\max}$, $\cdots$ can be chosen through intuition or flight experiments.

$$a_{\min} \leq \sqrt{a_x^2 + a_y^2 + (a_z + g)^2} \leq a_{\max} \leq l_1(F_1, F_2, \cdots)$$  \hspace{1cm} (3.6)  \\
$$\sqrt{j_x^2 + j_y^2 + j_z^2} \leq j_{\max} \leq l_2(F_1, F_2, \cdots)$$  \hspace{1cm} (3.7)$$

Choosing the model order $n$ requires a trade off between model accuracy and solution complexity. Previous work by [16] argued that a third order model is appropriate since the inertia of most quadrotors is negligible and instantaneous angular rates
(and hence jerks from Section 2.4) can be achieved. Further justification of using a third order model to represent the vehicle dynamics is discussed next.

### 3.4.2 Model Order

Since modelling state and actuator constraints as kinematic constraints removes the need to perform forward simulation, a decision must be made on the model order. In general, increasing model order captures more of the vehicle's true dynamics but at the cost of increasing complexity. Thus a balance must be made between accuracy and complexity.

[16] used a triple integrator model and argued that treating jerk as a control input is valid since near-instantaneous rates can be achieved. This physical intuition is generally true in practice. This subsection further supports using a triple integrator model because it more accurately represents the vehicle's true dynamics. An intuitive example is presented to support this claim.

Consider the vehicle in Fig. 3-5a travelling with initial speed $v_0$ and lateral acceleration $a_0$. The obstacles have already been clustered, represented by the red circles, and an intermediate goal, yellow diamond, has been deemed collision free and dynamically feasible for a double integrator model. Since this model naively assumes instantaneous changes in acceleration are achievable, the model is ignorant of $a_0$. The time required to change the acceleration vector causes the vehicle to stray off-course into an obstacle.

Now consider a triple integrator mode, shown in Fig. 3-5b, that is cognizant of $a_0$. Note that the intermediate goal is now at a different location because the model accurately predicted the collision that would have occurred with the intermediate goal in Fig. 3-5a. Modelling a quadrotor as a third order system is more representative of the attitude dynamics so it is more accurate. This intuition is further substantiated by the successful flight experiments in Chapter 5.
Figure 3-5: An example of why using a triple integrator model is more accurate than a double integrator model. The double integrator model does not account for the initial acceleration $a_0$. As a result the vehicle deviates from the desired path and crashes. The triple integrator accounts for $a_0$ and is thus more accurate.

### 3.4.3 Decoupled Planning

Now that using a third order model has been motivated, state and input constraints that capture actuator constraints must be found. This section follows closely to the derivation presented in [16].

Let $\mathbf{a}$ represent the acceleration vector:

$$||\mathbf{a}|| = m^{-1}||\mathbf{F}|| = \sqrt{\dot{x}^2 + \dot{y}^2 + (\ddot{z} + g)^2}$$

(3.8)

The total acceleration is constrained to be:

$$a_{\text{min}} \leq ||\mathbf{a}|| \leq a_{\text{max}}$$

(3.9)

where $a_{\text{max}}$ is the maximum acceleration the motors can produce and $a_{\text{min}}$ is the minimum acceleration required to keep the rotors spinning.
Recall the relation between jerk and angular rates:

\[
\begin{bmatrix}
0 \\
-\omega_{yd} \\
\omega_{xd} \\
0
\end{bmatrix} = \ddot{q}_d \otimes \begin{bmatrix}
0 \\
\dot{\hat{\mathbf{a}}}
\end{bmatrix} \otimes \ddot{q}_d
\tag{2.27}
\]

where \(\hat{\mathbf{a}}\) is the unit norm of \(\mathbf{a}\). Recall \(\omega_{xd}\) is removed since it can be chosen independently. Using the unit norm property of quaternions \([34]\) and the Cauchy-Schwarz inequality \([47]\), taking the 2-norm (2.27) gives:

\[
\omega_{x,y} = \sqrt{\omega_{yd}^2 + \omega_{yd}^2} \leq ||\dot{\hat{\mathbf{a}}}||
\tag{3.10}
\]

Expanding the right hand side of (3.10),

\[
\omega_{x,y} \leq \left\| \left( \frac{\mathbf{a}}{||\mathbf{a}||} - \frac{(\mathbf{a}\mathbf{a}^T) \dot{\mathbf{a}}}{||\mathbf{a}||^2} \right) \right\| = \left\| \left( \mathbf{I} - \frac{\mathbf{a}\mathbf{a}^T}{||\mathbf{a}||^2} \right) \frac{\dot{\mathbf{a}}}{||\mathbf{a}||} \right\|
\tag{3.11}
\]

Again using the Cauchy-Schwarz inequality,

\[
\omega_{x,y} \leq \left\| \left( \frac{\mathbf{I}}{||\mathbf{a}||^2} - \frac{\mathbf{a}\mathbf{a}^T}{||\mathbf{a}||^2} \right) \right\| \left\| \frac{\dot{\mathbf{a}}}{||\mathbf{a}||} \right\|
\tag{3.12}
\]

Noting the first term on the right hand side of (3.12) is the projection operator, (3.12) becomes:

\[
\omega_{x,y} \leq \left\| \frac{\dot{\mathbf{a}}}{||\mathbf{a}||} \right\|
\tag{3.13}
\]

If the angular rates are to remain bounded by a maximum angular rate \(\omega_{xy,max}\), then a necessary condition to satisfy the constraint is:

\[
\frac{||\dot{\mathbf{a}}||}{||\mathbf{a}||} \leq \omega_{xy,max} \implies ||\dot{\mathbf{a}}|| \leq ||\mathbf{a}|| \omega_{xy,max}
\tag{3.14}
\]
(3.14) can be further simplified by realizing $||a||$ is lower bounded by $a_{\text{min}}$:

$$\sqrt{\ddot{x}^2 + \ddot{y}^2 + \ddot{z}^2} \leq j_{\text{max}} \leq a_{\text{min}} \omega_{\text{xy, max}}$$

(3.15)

Hence, the angular rates can be bounded by $\omega_{\text{xy, max}}$ by selecting a maximum jerk $j_{\text{max}}$ such that (3.15) is satisfied.

The kinematic constraints in (3.9) and (3.15) are coupled nonlinear equations. This prohibits trajectories being independently generated for $x$, $y$, $z$ and thus increases the problem complexity. However, the problem can be decoupled at the expense of optimality (see [16]) but with the benefit of significantly reducing the problem complexity. The decoupling derivation is shown below.

First consider the minimum acceleration constraint in (3.9). This constraint is always satisfied if a minimum $z$ acceleration $\ddot{z}_{\text{min}}$ is specified:

$$\ddot{z} \geq \ddot{z}_{\text{min}}$$

$$\ddot{z}_{\text{min}} + g \geq a_{\text{min}} \implies \ddot{z}_{\text{min}} \geq a_{\text{min}} - g$$

(3.17)

It may seem desirable to make $\ddot{z}_{\text{min}}$ as small as possible. However, as will be shown shortly, there is a tradeoff between making $\ddot{z}_{\text{min}}$ small and the overall maneuverability of the vehicle.

The maximum acceleration constraint is satisfied by allocating the same acceleration to each axis:

$$\ddot{x}_{\text{max}} = \ddot{y}_{\text{max}} = \ddot{z}_{\text{max}} + g = \frac{a_{\text{max}}}{\sqrt{3}}$$

(3.18)

It should be noted that this approach limits $\ddot{z}_{\text{max}}$ since vertical acceleration must compensate for gravity.

The maximum jerk constraint is satisfied by allocating the same amount of jerk to each axis:

$$\dddot{x}_{\text{max}} = \dddot{y}_{\text{max}} = \dddot{z}_{\text{max}} = a_{\text{min}} \omega_{\text{xy, max}} \frac{1}{\sqrt{3}} = \frac{(\ddot{z}_{\text{min}} + g) \omega_{\text{xy, max}}}{\sqrt{3}}$$

(3.19)
The consequences of making \( \ddot{z}_{\text{min}} \) very small results in a lower maximum allowable jerk along each axis. This makes the vehicle less agile. However, making the minimum vertical acceleration large prevents the vehicle from accelerating downward. Therefore, a careful design process that requires some domain knowledge and consideration of the vehicle's physical capabilities is needed.

3.4.4 Minimum-Time Velocity Planning

Optimal Control

By showing the kinematic constraints can be decoupled in Section 3.4.3, a trajectory can be planned in each axis independently. This subsection formally presents the optimal control problem for a triple integrator model with state and control input constraints. It is shown that relaxing the final position constraint leads to a closed form solution for the control input switching times. The derivation is for a trajectory along the \( x \) coordinate system but is applicable to any of the inertial coordinates.

Let \( x = (x_1, x_2, x_3) = (x, \dot{x}, \ddot{x}) \) represent the position, velocity, and acceleration of the vehicle in the \( x \) direction. The goal of optimal control is to find the control input \( u \) such that a cost is minimized. In this case, the control input is jerk. Since the goal is to point the velocity vector in a desired direction as quickly as possible, minimizing the final time of the maneuver is an appropriate cost function. The standard form of a minimum-time optimal control problem for a triple integrator system is:

\[
\begin{align*}
    u^* &= \arg \min_{u} t_{f,x} \\
    \text{subject to kinematic dynamics:} & \\
    \frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u
\end{align*}
\]

subject to kinematic dynamics:

\[
\begin{align*}
    \frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u
\end{align*}
\]
boundary conditions:

\begin{align*}
    x_1(t = 0) &= x_0 & \quad x_1(t = t_f) &= x_f, \quad \text{free} \quad (3.22) \\
    x_2(t = 0) &= \dot{x}_0 & \quad x_2(t = t_f) &= \dot{x}_f \quad (3.23) \\
    x_3(t = 0) &= \ddot{x}_0 & \quad x_3(t = t_f) &= \ddot{x}_f \quad (3.24)
\end{align*}

and state constraints:

\begin{align*}
    |\ddot{x}| &\leq \ddot{x}_{\max} \quad (3.25) \\
    |u| &\leq \dddot{x}_{\max} = u_m \quad (3.26)
\end{align*}

It is convenient to define the Hamiltonian function:

\[ \mathcal{H} = 1 + \lambda^T \begin{bmatrix} A \dot{x} + Bu \end{bmatrix} \quad (3.27) \]

where \( \lambda = [\lambda_1 \lambda_2 \lambda_3]^T \) is the costate vector that represents the marginal cost of the constraints.

There are a number of ways to solve an optimal control problem for a linear system with state and input constraints. The approach taken here is the direct adjoint method [17], which entails augmenting the Hamiltonian in (3.27) with the state constraints:

\[ \mathcal{H}_a = 1 + \lambda^T \begin{bmatrix} A \dot{x} + Bu \end{bmatrix} + \eta_1 (x_3 + \dddot{x}_{\max}) + \eta_2 (x_3 - \dddot{x}_{\max}) \quad (3.28) \]

with

\begin{align*}
    \eta_i &\geq 0, \quad i = 1, 2 \quad (3.29) \\
    \eta_1 &= 0, \quad \text{if } x_3 > -\dddot{x}_{\max} \quad (3.30) \\
    \eta_2 &= 0, \quad \text{if } x_3 < \dddot{x}_{\max} \quad (3.31)
\end{align*}
A necessary condition for optimality is the costates must satisfy:

\[ \dot{\lambda} = -\frac{\partial H}{\partial x} \]  

(3.32)

which gives:

\[ \dot{\lambda}_1 = 0 \]  

(3.33)

\[ \dot{\lambda}_2 = \lambda_1 \]  

(3.34)

\[ \dot{\lambda}_3 = \lambda_2 + \eta_1 - \eta_2 \]  

(3.35)

By Pontryagin's minimum principle [45], the optimal control \( u^* \) is given by:

\[ u^* = \arg\min_u H_a \]  

(3.36)

\[ = \arg\min_u \lambda_3 u \]  

(3.37)

The optimal control \( u^* \) must take the opposite sign of \( \lambda_3 \) to minimize the augmented Hamiltonian \( H_a \). Intuitively, \( u^* = 0 \) must be true if the state constraint is active. Further, \( \lambda_3 = 0 \) must also be true for an active state constraint. Therefore, the optimal control input is:

\[ u^* = \begin{cases} 
  u_m & \lambda_3 < 0 \\
  0 & \lambda_3 = 0 \\
  -u_m & \lambda_3 > 0 
\end{cases} \]  

(3.38)

Satisfying the final position constraint significantly increases the problem's complexity. As shown in [16], a bisect algorithm is required to find the final time. This is not ideal since this increases computational load and latency. Also, from a philosophical perspective, one can argue that for high-speed navigation position is only important for collision checking. Hence, specifying a final position is not required. By allowing the final position to be free, the costate equation in (3.33) vanishes by the transversality condition on \( H_a \). Removing (3.33) is equivalent to reducing the
model order by one, which is equivalent to reformulating the problem in velocity space. Because of this, the solution is called a velocity trajectory.

Closer observation of (3.38) provides information about the number of times the control input changes. For instance, if the state constraint is inactive, the control input switches at most one. If the state constraint is active, the control input switches at most twice. This is useful information that will be used in deriving the closed form solution.

Solution Procedure

The previous analysis established the theoretical foundation for minimum-time trajectory planning with a relaxed final position constraint. This subsection derives the closed form solution for the optimal control problem previously presented. The initial position, velocity and acceleration are assumed to be known. The desired final velocity is also assumed known.

First define the boundary conditions:

\[
\begin{align*}
    x(t=0) &= x_0, & x(t=t_{f,x}) &= \text{free} \quad (3.39) \\
    v(t=0) &= v_0, & v(t=t_{f,x}) &= v_f \quad (3.40) \\
    a(t=0) &= a_0, & a(t=t_{f,x}) &= 0 \quad (3.41)
\end{align*}
\]

where the final position constraint is relaxed to permit a closed-form solution. Note the final acceleration is constrained to be zero so the vehicle will be in cruise at the end of the maneuver. The state and input constraints are:

\[
\begin{align*}
    |a| &\leq \ddot{x}_{\text{max}} = a_{x,\text{max}} \quad (3.42) \\
    |j| &\leq \dddot{x}_{\text{max}} = j_{x,\text{max}} \quad (3.43)
\end{align*}
\]

(3.38) showed that the optimal control input switches at most once if the state constraint is inactive and at most twice when the state constraint is active. This leads to the control strategy "bang-bang" and "bang-off-bang", respectively. An example
of bang-bang control is shown in Fig. 3-6a where \( t_i \) is the \( i^{th} \) switch time. Bang-bang control entails maximum control effort until the final conditions are satisfied. The resulting acceleration and velocity profile for bang-bang control are shown in Fig. 3-6b and Fig. 3-6c for arbitrary initial conditions \( a_0 \) and \( v_0 \). Conversely, the bang-off-bang control input, Fig. 3-6d, is zero when the acceleration constraint is active, as is the case in Fig. 3-6e. The corresponding velocity profile is shown in Fig. 3-6f. Since the control input is constant, acceleration is a piecewise linear function and velocity is a piecewise quadratic function.

Figure 3-6: Bang-bang and bang-off-bang control strategy for inactive and active state constraints. The resulting acceleration and velocity are piecewise linear and quadratic functions.

For clarity, the term *phases* is used to distinguish between the polynomial solutions.
for velocity and acceleration; adopting the standard that phase 2 corresponds to when the acceleration constraint is active. Thus, phase 2 is only present in the bang-off-bang case while there is always a corresponding solution for phase 1 and 3. The velocity and acceleration polynomial for phase \( k \) are represented as \( v^{(k)} \) and \( a^{(k)} \).

The closed form solution for the switching times is now derived. Define \( j := \text{sgn} \{v_f - v_0\} j_{x,\text{max}} \) so the derivation can be valid for arbitrary initial conditions. Let \( v_i \) and \( a_i \) be the velocity and acceleration at the \( i^{th} \) switch. Evaluating velocity and acceleration at the first switch time \( t_1 \) during phase 1:

\[
v^{(1)}(t_1) = v_1 = v_0 + a_0 t_1 + \frac{1}{2} j t_1^2 \tag{3.44}
\]

\[
a^{(1)}(t_1) = a_1 = a_0 + j t_1 \tag{3.45}
\]

If the acceleration constraint is assumed to not be violated then phase 2 does not exist. Using (3.44) and (3.45) to enforce continuity between phase 1 and phase 3:

\[
v^{(3)}(t_{f,x}) = v_f = v_1 + a_1 (t_{f,x} - t_1) - \frac{1}{2} j (t_{f,x} - t_1)^2 \tag{3.46}
\]

\[
a^{(3)}(t_{f,x}) = 0 = a_1 - j (t_{f,x} - t_1) \tag{3.47}
\]

Observe that \( t_{f,x} \) can be solved for in (3.47):

\[
t_{f,x} = t_1 + \frac{a_1}{j} \tag{3.48}
\]

Substituting (3.48) in (3.46),

\[
v_f = v_1 + \frac{a_1^2}{2j} \tag{3.49}
\]

Since \( v_1 \) and \( a_1 \) are given by (3.44) and (3.45), (3.49) can be re-written as:

\[
v_f = v_0 + a_0 t_1 + \frac{1}{2} j t_1^2 + \frac{(a_0 + j t_1)^2}{2j} \tag{3.50}
\]

\[
= \left( v_0 + \frac{a_0^2}{2j} \right) + 2a_0 t_1 + j t_1^2 \tag{3.51}
\]
which is a quadratic equation in \( t_1 \). It thus can be easily solved for by:

\[
t_1 = \max \left\{ \frac{-a_0 \pm \sqrt{\frac{1}{2} a_0^2 - j (v_0 - v_f)}}{j} \right\}
\]  \hspace{1cm} (3.52)

where \( \max \{\cdot\} \) is taken for the positive root.

It was assumed the acceleration constraint was not violated in the derivation of (3.52). Since \( t_1 \) is now known, the assumption can be checked by verifying \( a_1 \leq a_{x,\text{max}} \). If \( a_1 > a_{x,\text{max}} \) then enforcing continuity across phase 1 and phase 3 is not valid. \( t_1 \) is then given by:

\[
t_1 = \frac{a_{x,\text{max}} - a_0}{j}
\]  \hspace{1cm} (3.53)

Applying continuity across phase 2 and phase 3:

\[
a^{(2)}(t_2) = a^{(3)}(t_2) = a_{x,\text{max}}
\]  \hspace{1cm} (3.54)

so,

\[
a^{(3)}(t_{f,x}) = a^{(3)}(t_2) - j (t_{f,x} - t_2)
\]  \hspace{1cm} (3.55)

\[
= a_{x,\text{max}} - j (t_{f,x} - t_2)
\]  \hspace{1cm} (3.56)

\[
= 0
\]  \hspace{1cm} (3.57)

Solving for \( t_{f,x} \),

\[
t_{f,x} = t_2 + \frac{a_{x,\text{max}}}{j}
\]  \hspace{1cm} (3.58)

Continuity for velocity can now be enforced to find \( t_2 \). For instance,

\[
v^{(2)}(t_2) = v^{(3)}(t_2) = v_2
\]  \hspace{1cm} (3.59)
Also,

\[ v^{(3)}(t_{f,x}) = v_2 + a_{x,\text{max}} (t_{f,x} - t_2) - \frac{1}{2} j (t_{f,x} - t_2)^2 \]

\[ = v_f \]

Using (3.58) and solving for \( v_2 \),

\[ v_2 = v_f - a_{x,\text{max}} (t_{f,x} - t_2) + \frac{1}{2} j (t_{f,x} - t_2)^2 \]

\[ = v_f - \frac{a^2_{x,\text{max}}}{2j} \]

In addition, the velocity at \( t_1 \) is:

\[ v_1 = v^{(1)}(t_1) = v_0 + a_0 t_1 + \frac{1}{2} j t_1^2 \]

(3.63) and (3.64) can be used to solve for \( t_2 \). Evaluating \( v^{(2)}(t_2) \),

\[ v^{(2)}(t_2) = v_2 = v_1 + a_{x,\text{max}} (t_2 - t_1) \]

Solving for \( t_2 \),

\[ t_2 = t_1 + \frac{(v_2 - v_1)}{a_{x,\text{max}}} \]

which can be solved for using (3.63) and (3.64). Therefore, a closed-form solution for the switching times exists for minimum-time trajectory planning with a relaxed final position constraint. This enables extremely low-latency planning since solving for the switching times entails solving a quadratic equation and checking if the state constraint is satisfied. If it is violated, a set of linear equations must be solved. In either case, the computational complexity is insignificant, as shown in Section 3.6.

### 3.4.5 Collision Checking

Collision checking is generally the most computationally intensive component of trajectory planning. Collision checkers need to efficiently check if the planned trajectory
will pass through an obstacle. In addition, it should account for state and perception uncertainty. This subsection briefly reviews typical collision checking approaches and presents an adapted approach that addresses the limitation of previous work.

The naive way to check for collisions entails finely sampling the trajectory and determining if the resulting samples are close to or within an obstacle. There are a number of issues with this approach. First, it is easy to misclassify the trajectory as collision-free if the sampling interval is large. As a result, the sampling interval is made very small at the expense of computational complexity. Second, this approach scales linearly with the trajectory duration so planning longer trajectories is disadvantageous because it requires more computation time. Incorporating state uncertainty into this approach is possible but typically not done.

Another approach for collision checking is through convex segmentation of the free space \[23, 25, 26\]. This approach entails confining each trajectory segment to remain in a set convex polyhedrons that represent free space. The drawback of this approach is that it also entails sampling the trajectory to ensure confinement. It is also difficult to encode state uncertainty into the collision check.

The collision checking method adopted here is borrow from a sampling-based methods\[48\] that significantly reduces the complexity of collision checking. \[48\] defined a "safety certificate" that defines a region of space that is collision free. To this author's knowledge, this work is the first to adapt "safety certificates" to trajectory planning. The method of using safety certificates is now presented.

Consider again the scenario that was first presented in Section 3.4.2 where the vehicle is navigating through a set of obstacles. When a new plan is generated, the distance to the closest obstacle is calculated, as in Fig. 3-9a. Since this is the closest obstacle, the entire trajectory within the sphere of radius \(d_{\text{obst,min}}\) is guaranteed to be collision free. The next time-instance that a collision is possible is when \(||r_0'(t^*)|| = d_{\text{obst,min}}\). Since a desired top-speed \(v_{\text{top}}\) is known, \(t^*\) can be approximated to be \(t^* \approx \frac{d_{\text{obst,min}}}{v_{\text{top}}}\). Even if the vehicle is not travelling at top speed, it is guaranteed to be within the collision-free sphere. The process of finding the closest obstacle and evaluating the trajectory at the approximate boundary of the collision-free sphere is
repeated, Fig. 3-9b-Fig. 3-9d, until either the closest point is the intermediate goal or an obstacle is closer than a predefined safe distance known as the buffer distance.

High state uncertainty is very common in real systems so it should be included in the planning process. A natural way to incorporate state and obstacle uncertainty is in the collision check. The approach taken here is to have the safe buffer distance grow linearly with time, shown in Section 3.6. The initial buffer size and growth rate are both a function of the initial state estimate uncertainty. Both parameters require tuning and dictate how conservative the planner is.

3.5 Safety

An important role of the planner is ensure the vehicle never enters a state where no maneuver can prevent a collision, also known as an inevitable collision state (ICS). The approach commonly taken to guarantee safety is for the vehicle to always be able to execute a stop maneuver. For mapping and other low-speed applications this approach is completely valid. However, this can be very restrictive for applications that require high-speed flight. The primary reason is that state-of-the-art perception systems still can only perceive the world over a short distance. Thus, the vehicle's speed is limited so the stopping distance is less than the sensing distance.

One approach to increase speed is to ensure the vehicle can always swerve to avoid an obstacle. Although this requires some domain knowledge, like typical size of obstacles, a substantial increase in speed can be achieved. The stopping distance $d_{\text{stop}}$ and swerving distance $d_{\text{swerve}}$ are defined in Fig. 3-7. The stopping distance is always longer than the swerving distance since stopping requires complete dissipation of the vehicle's kinetic energy. Swerving on the other hand only requires a perturbation of the velocity vector to avoid an obstacle.

Fig. 3-8 shows the sensing range required to execute a stop or swerve maneuver as a function of initial velocity. These curves were generated by the trajectory generation method presented in Section 3.4.4 with $a_{xy,\text{max}} = 10 \text{m/s}^2$ and $j_{xy,\text{max}} = 40 \text{m/s}^3$ and for a 0.6m wide obstacle. There are three distinct regions in Fig. 3-8: safe, marginally
Figure 3-7: The stop and swerve distance definition. The stop distance is always longer than the swerving distance since stopping requires complete dissipation of the vehicle’s kinetic energy. A faster speed can be achieved if the vehicle is forced to swerve around obstacles.

safe, and inevitable collision state. The vehicle can swerve or stop when in the safe region while it can only swerve in the marginally safe region. If the vehicle is in the ICS region a collision is guaranteed to occur if an obstacle is present. It’s clear that the sensing range required to stop is always longer than that to swerve. As a result, for a given perception range, a faster speed is achievable if the vehicle swerves instead of stops to prevent collisions. Operating in the marginally safe regime places more emphasis on the planner’s ability to make quick and intelligent decisions. Though this work was not able to operate within the marginally safe regime due to flight volume limitations, future hardware and simulation experiments are planned to evaluate the planner’s performance during marginally safe flight.

3.6 Algorithm Benchmarking

The clustering, cluster sorting, trajectory generation, and collision checking were designed to have extremely low computation time and complexity. Each component was tested individually on an Intel Core i7 laptop. Table 3.1 shows the average computation for 100,000 evaluation times. It must be noted that the collision check
Figure 3-8: The stop and swerve curve for the planner where the critical sensing range is minimum range required to swerve or stop at a given speed. There are three distinct regions: safe, marginally safe, and inevitable collision state (ICS).

time is for one evaluation. This corresponds to when an obstacle is within the buffer distance after the first propagation, like that in Fig. 3-9b. It's difficult to find a representative number because the approach is highly dependent on the environment and planning parameters. Overall, the total run-time during experiments is between 0.5-1ms, almost two orders of magnitude faster than the current state-of-the-art.

Table 3.1: Planning Algorithm Benchmarking.

<table>
<thead>
<tr>
<th>Component</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering</td>
<td>$0.187 \pm 0.0081$</td>
</tr>
<tr>
<td>Sorting</td>
<td>$0.121 \pm 0.0059$</td>
</tr>
<tr>
<td>Traj. Gen.</td>
<td>$0.00738 \pm 0.00085$</td>
</tr>
<tr>
<td>Collision Check</td>
<td>$0.0487 \pm 0.0038$</td>
</tr>
</tbody>
</table>

3.7 Summary

This chapter presented the approach taken for perception and planning for high-speed navigation. The limitations of previous work were addressed by extracting key features from perception data and identifying potentially traversable regions via cluster-
ing. A closed form solution was then derived for a minimum-time, decoupled optimal control problem for a third order system by relaxing the final position constraint. An efficient collision checking method using safety certificates that also captured state and obstacle uncertainty was then presented. It was then shown that a substantial increase in speed is possible if the vehicle is allowed to only swerve, as opposed to stop, to prevent collisions; this requires some domain knowledge though. Lastly, the computation times for clustering, cluster sorting, trajectory generation, and collision checking were all shown to be less than 200\(\mu\)s, with a total run-time between 0.5-1ms. This is almost two orders of magnitude faster than state-of-the-art algorithms.
Figure 3-9: The collision checking approach using safety certificates. The closest obstacle is calculated when a new trajectory is generated. The trajectory is then re-evaluated at the next possible collision time, which occurs at the edge of the collision-free sphere. This process is repeated until the goal is reached or a collision is detected.
Figure 3-10: A linear growth rate for the safe buffer to account for state and obstacle uncertainty. The growth rate and initial size are function of initial uncertainty and dictate how conservative the planner is.
Chapter 4

Hardware and Software

4.1 Overview

This chapter presents the hardware and software used in this thesis. The hardware for a medium size vehicle equipped with a 2D laser for perception and a flight computer for on-board processing is first presented. The maximum allowable acceleration and jerk for desired trajectories is also identified. The software architecture, that entails using motion capture, a flight computer, and a custom autopilot for planning and control, is then presented. Lastly, the Measurable Augmented Reality (MAR) system that provides important context for robot experiments is introduced.

4.2 Hardware

The quadrotor used for flight experiments was built using a combination of commercial and custom components. The goal was to build a relatively small vehicle that could carry a moderate payload without sacrificing agility. The vehicle is shown in Fig. 4-1. The frame is composed of arms from a DJI F330 quadrotor and a custom base plate made out of Delrin. The custom base plate was designed to make mounting the 2D laser and flight computer easier. The motors, ESC’s, and propellers were bought commercially. RCMC 2212 1000Kv motors were used with 20A rated ESC’s and 8” propellers. With a 4S battery, the vehicle can hover at approximately 42%, leaving
Figure 4-1: Quadrotor used for flight experiments. A Hokuyo 2D laser is used for perception. An on-board flight computer runs the perception, planning, and control algorithms developed in the previous chapters.

enough control authority for aggressive flight experiments. The total vehicle weight is 1.2kg.

For the perception, planning, and control algorithms developed in the previous chapters to run on-board the vehicle, a light-weight flight computer is required. The Odroid XU4, shown in Fig. 4-2a, is a single-board computer with eight cores, weighs 60 grams, and has the footprint of a credit card. It is one the lightest and most powerful single-board computers on the market so it was used as the flight computer. To meet the real-time demands of fast attitude control, a custom autopilot was designed, developed, and tested. [14] originally designed the autopilot, shown in Fig. 4-2b. The autopilot communicates directly with flight computer via a serial connection. A more detailed discussion of the software architecture can be found in Section 4.3.

A Hokuyo URG-04LX 2D scanning laser range finder is used for perception. The Hokuyo, located on the top of the vehicle in Fig. 4-1, has a max range 4.1m, horizontal FOV of 240°, a scan time of 60ms, weighs 160 grams, and operates at 10Hz. The short sensing range is ideal for high-speed experiments in small flight volumes since obstacles are known only when the vehicle is in close proximity. This places additional importance on the planner’s ability to generate low-latency solutions.

Recall in Section 3.4 that the motor constraints are to be represented by acceleration and jerk constraints. These constraints can now be specified for the vehicle
Figure 4-2: The flight computer is a lightweight, small form factor computer that permits running perception, planning, and control algorithms on-board the vehicle. It sends attitude commanded to a custom autopilot that performs attitude control.

in Fig. 4-1. As previously stated, the vehicle hovers at approximately 42% throttle. From this, the maximum acceleration $a_{\text{max}}$ can be deduced. Accounting for the drop off in acceleration at high throttle values, the max acceleration is $a_{\text{max}} = 18\text{m/s}^2$. Assuming acceleration is equally allocated to the three axis, $a_{k,\text{max}} \approx 10\text{m/s}^2$ for axis $k$. Note that this leaves very little remaining acceleration in the vertical direction because of gravity. Since 3D flight is prohibited by the laser scanner this seemed acceptable. Downward acceleration is still required in-case the vehicle gains altitude during an aggressive maneuver. Flight tests revealed $\ddot{z}_{\text{min}} = -\frac{1}{2}g$ is enough to correct for any altitude gain.

The last remaining constraint is the maximum jerk $j_{k,\text{max}}$ for axis $k$. There are a number of factors that impact the choice of $j_{k,\text{max}}$, such as the physical limitations of the vehicle and how the perception data is influenced by large jerk maneuvers. Flight experiments showed that the vehicle is capable of achieving a max jerk of $60\text{m/s}^3$ before the motor constraints are violated. This corresponds to an angular rate of $700^\circ/s$. Unfortunately, this angular rate is fast enough to distort the laser scan and
is detrimental to the planner. \( j_{k,\text{max}} = 40 \text{m/s}^3 \) was the largest jerk value that did not corrupt the laser scan.

Section 3.4.4 showed that the solution to the minimum-time optimal control problem employed a bang-bang or bang-off-bang control strategy independent of how much of a change in velocity is desired. Hence, a jerk of \( 40 \text{m/s}^3 \) is applied for any change in desired velocity. In practice this works well for large changes in desired velocity but applying maximum jerk for minor velocity changes degrades tracking and induces oscillations. This was addressed by specifying a lower jerk value for minor velocity changes. A jerk of \( 5 \text{m/s}^3 \) for a change in velocity command less than 20\% of the max speed was found to improve tracking and eliminate oscillations.

4.3 Software

The overall software and data flow diagram is shown in Fig. 4-3. A Vicon motion capture system [49] broadcasts very precise position, velocity, attitude, and angular rate measurements over WIFI using the Robot Operating System (ROS) [50]. The flight computer accesses the Vicon measurements and performs feedback. The laser scan data from the Hokuyo is sent over USB to the flight computer. Bidirectional serial communication between the flight computer and the autopilot occurs over a hard wire. This approach is significantly more robust than that presented in [14] because data packets are less susceptible to being dropped over a hard wire than WIFI. The autopilot interfaces with the vehicle by sending PWM motor commands to ESC's.

4.3.1 Flight Computer

A key advantage of using a flight computer with an ARM processor is the ability to use the Robot Operating System (ROS) [50]. ROS is an open source software package that provides a flexible framework and tools for robotics development. ROS handles all the necessary message passing required for different software components. For instance, any computer can access the Vicon measurements by simply being on
The flight computer is first responsible for running the position and velocity controller presented in Chapter 2. The flight computer uses the position and velocity measurements from Vicon to generate the acceleration and jerk required to track a desired trajectory. Acceleration and jerk are transformed to attitude and attitude rate commands that are sent to the autopilot. The process of mapping acceleration and jerk to attitude and attitude rate commands was presented in Section 2.3. The controller is written in C++ and runs at 100Hz.
The flight computer is also responsible for processing laser scan data and generating a minimum-time trajectory. ROS provides a driver for Hokuyo lasers so accessing the laser scan data is easy. Before the scan data is clustered, as described in Section 3.3, the scan must be filtered to remove parts of the scan that strike the floor during large attitude maneuvers. This is required so the clustering algorithm does not misclassify the ground as an obstacle. This problem arises from the 2D nature of the laser scan. Filtering entails transforming the data from the body frame to the world frame and removing points that have a $z$ position less than a threshold value. For instance, points that have a $z$ position less than 0.1m are likely to be from the ground. The transformation from body to world frame involves applying the quaternion transformation in Section 2.2.1. The scan time for URG-04LX is long enough for the vehicle to have rotated a significant amount during the scan. Thus, linear interpolation between the quaternion at the beginning and end of the scan is used to improve ground plane rejection accuracy.

Clustering and minimum-time trajectory planning is then done on the filtered scan. The clustering, cluster sorting, and trajectory generation take less than 1ms total to generate a plan on a standard laptop. The computation time on the Odroid XU4 is slightly more at 5ms, a result of it being a single board computer. This is still an order of magnitude faster than the state-of-the-art perception and planning algorithms. All the aforementioned components occur when new laser data is available. The polynomial solution to the minimum-time optimal control problem is evaluated at 100Hz and sent to the controller. The code is written in C++ and uses the Eigen library extensively [51].

4.3.2 Autopilot

The role of the autopilot is to perform high-rate attitude control by performing feedback on attitude and angular rates. Recall that the attitude dynamics evolve much faster with time than does position or velocity. As a result, the attitude controller must operating at a much faster rate than position controller. The autopilot samples angular rates from an IMU and integrates them to obtain the vehicle’s attitude at
1kHz, ten times faster than the rate of the position controller. Motor commands are sent at 400Hz because of the bandwidth limitations of PWM signals.

The autopilot uses a 16-bit dsPIC33F microcontroller from Microchip Technologies Inc.[52] that reads angular rates from an MPU-6050 Invensense IMU [53]. The MPU-6050 is a single chip gyroscope and accelerometer that can measure angular rates up to 2000°/s and accelerations up to 16g.

The MPU-6050's gyroscope has very low noise characteristics for similarly priced IMU's. Even so, noise in the gyroscope causes the attitude estimate to drift over time. An attitude correction from an external source, such as Vicon, is required to correct the drift. A complimentary filter is used to blend the on-board and external estimate. A complimentary filter [54] has low computational complexity and a long history in attitude estimation. The attitude correction is sent to the autopilot at 100Hz from the flight computer. Future work will include using an adaptive complimentary filter [55] that intelligently blends the gyroscope estimate with the accelerometer estimate. This will remove the external estimate dependency and make the device more versatile.

### 4.3.3 Measurable Augmented Reality

A major challenge in conducting robotics experiments is displaying useful information in real-time. For instance, displaying the robot’s belief space is useful for understanding its decision making process. It gives the viewer context while observing the robot operate. The Measurable Augmented Reality (MAR) system [56] at MIT’s Aerospace Controls Lab can display real-time perception, planning, and control data using six downward facing projectors and a motion capture system. Two examples of this system are shown in Fig. 4-4. Fig. 4-4a displays data for a control experiment designed to show the limitations of pure feedback. Fig. 4-4b shows the vehicle’s perception range and resulting path through an obstacle course for a newly developed planner. In both cases, MAR provides context for both experiments that would otherwise be hard to extract from plots and tables. MAR is extensively used in Chapter 5.
Significant lag between actual and goal position

Large pitch angle to accelerate forward and try to catch goal position

Large throttle setting leaves very little remaining control authority

Figure 4-4: The Measurable Augmented Reality system provides useful context for robot experiments by displaying perception, planning, and control data in real-time.

(a) MAR for control experiment.

(b) MAR for perception and planning experiment.
4.4 Summary

This chapter presented the hardware and software used in this thesis. A combination of commercial and custom components allowed for on-board perception, planning, and position control. A lightweight, credit card sized single board computer was used to process laser scan data, plan trajectories, and perform position and velocity feedback. A 2D laser, which requires careful filtering when the scan strikes the ground plane, was used for perception. The maximum acceleration and jerk the vehicle is capable of were also identified. A custom autopilot that performs high-rate estimation and feedback on the attitude and angular rate commands from the outer-loop control was presented. Lastly, the Measurable Augmented Reality system that provides useful context for experiments was also introduced.
Chapter 5

Results

5.1 Overview

Flight experiments were performed to test the perception, planning, and control approach taken in this thesis. The vehicle was instructed to fly toward a known goal without prior knowledge of obstacles. Experiment parameters for three flight tests are summarized in Table 5.1. Flight 1 and 2 have the same experiment parameters and obstacle configuration but feedforward was not used in Flight 1. The speed and maximum jerk were increased for Flight 3. The obstacle configuration for Flight 3 was also modified to increase level of difficulty. Note that all flights are conducted in the safe region of Fig. 3-8 because of flight volume limitations. The rest of the chapter discusses the results for each flight test.

Table 5.1: Flight Experiments For Minimum-Time Trajectory Planning with Clustering.

<table>
<thead>
<tr>
<th>Flight Number</th>
<th>$v_{\text{max}}$ (m/s)</th>
<th>$a_{\text{max}}$ (m/s$^2$)</th>
<th>$j_{\text{max}}$ (m/s$^3$)</th>
<th>$N_{\text{obst}}$</th>
<th>Sensing Range</th>
<th>FF</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>6</td>
<td>3.5</td>
<td>N</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>6</td>
<td>3.5</td>
<td>Y</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>10</td>
<td>40</td>
<td>5</td>
<td>3.5</td>
<td>Y</td>
<td>✓</td>
</tr>
</tbody>
</table>
5.2 Flight 1

Flights 1 was designed to test the perception, planning, and control system at a moderate speed without feedforward. The obstacle course for Flight 1 is shown in Fig. 5-1. The vehicle is located at the top of the image and the goal is located at the bottom (not shown). The red outline projected on the floor is representative of the vehicle's perception range. As is the case for all the flight experiments, the vehicle cannot see the obstacles from its starting location.

The vehicle's commanded and actual path for Flight 1 is shown in Fig. 5-2. Note that each obstacle has been inflated to account for the vehicle's buffer size. The vehicle actually clips the first obstacle. The collision was not catastrophic but clipping is unacceptable. Flight 1 was deemed unsuccessful.

The resulting path for Flight 1 shows the vehicle gets very close to the first obstacle, a consequence of the laser pointing at the ground during the acceleration phase. This places extra emphasis on the vehicle's ability to quickly respond to new commands. Lateral position and velocity tracking for Flight 1 is shown in Fig. 5-3. Fig. 5-3a and Fig. 5-3b show poor lateral position and velocity tracking with a peak position and velocity error of 66.8cm and 2.39m/s, respectively. Without feedforward,
the vehicle's response to commands is delayed causing large overshoot. This is the primary cause of the collision.

5.3 Flight 2

Flights 2 was designed to test the system in the same obstacle course as Flight 1 but with feedforward. The commanded and actual path of the vehicle is shown in Fig. 5-4. The first observation is that the vehicle takes a different path through the identical obstacle fields. This is common when planning with raw perception data because of the stochastic nature of sensor measurements and is thus not important. What is important is that the vehicle successfully reached the goal without a collision.

Recall Flight 1 had a peak position and velocity error of 66.8cm and 2.39m/s, respectively. Conversely, the lateral position and velocity tracking for Flight 2, Fig. 5-
Figure 5-3: Lateral position and velocity tracking for Flight 1. The absence of feed-forward in Flight 1 lead to poor tracking thus causing the vehicle to clip an obstacle.

5a and Fig. 5-5b, is significantly better with a peak position and velocity error of 12.9cm and 0.57 m/s, respectively. It is clear that the absence of feedforward in Flight 1 caused the failure. This again shows the important role of feedforward in aggressive, high-speed maneuvers.

Fig. 5-5 showed that feedforward significantly improved position and velocity tracking. To what degree feedforward helped can be quantified by comparing the feedforward command to the total acceleration and jerk command. The total command in this context is the sum of the feedforward and feedback commands. Fig. 5-6 shows how the feedforward and total command compare. Qualitatively, the feedforward acceleration and jerk in Fig. 5-6a and Fig. 5-6b, respectively, are in good agreement with the total command. The difference between the two commands, normalized by the max total command, is shown in Fig. 5-6c for acceleration and Fig. 5-6d for jerk. The acceleration error is less than 34%, which is equivalent to 34% of the command is from feedback. Interestingly, the peaks in Fig. 5-6c occur at the discontinuities in the jerk command. This is also true for the jerk error, which is less than 35% for the flight. The peak errors occurring at the corners of the jerk command is not surprising because instantaneous jerk is not physically possible. With that said, the error is not large enough to outweigh the benefits of using this model.

Fig. 5-11 shows a series of camera and RVIZ images as the vehicle dodges obstacles in Flight 2. Since the obstacles are outside the sensing range from the start location,
the vehicle plans a straight trajectory toward the goal, Fig. 5-11a and Fig. 5-11c. The first obstacle is seen when the vehicle is approximately 2m away. The acceleration phase causes the vehicle to pitch forward leaving the laser useless until top speed is reached. The vehicle subsequently swerves once the first obstacle is seen, Fig. 5-11b and Fig. 5-11d. The vehicle continues to avoid the obstacles until it reaches the goal in Fig. 5-11j and Fig. 5-11l.

5.4 Flight 3

Flight 1 and 2 showed that the perception, planning, and control system is capable of navigating though an obstacle field at a moderate speed. Flight 3 was designed
Figure 5-5: Lateral position and velocity tracking for Flight 2. Feedforward significantly improves tracking performance and is thus responsible for the successful flight test.

to test the system’s performance at a higher speed in a more complicated obstacle configuration. The maximum allowable jerk was increased to allow a higher degree of agility. The obstacle course for Flight 3 is shown in Fig. 5-7. Again the vehicle is unaware of the obstacles from the starting location.

The vehicle’s commanded and actual path for Flight 3 is shown in Fig. 5-8. The vehicle is again in close proximity to the first obstacle because the laser only sees the ground during the acceleration phase. While the vehicle successfully avoids all obstacles, the actual path deviates from the commanded path more than previous flight tests.

To better understand the vehicle’s behavior in the latter stage of the trajectory, the acceleration Fig. 5-10a and jerk Fig. 5-10b profiles are shown in Fig. 5-10. The normalized acceleration error in Fig. 5-9c shows that a 31% peak error occurs in the middle of the trajectory, which is roughly the same as Flight 2. However, the peak error at the end of the trajectory is over 50%. There are a number of things that could have caused the acceleration error. The most likely source of the error is a damaged motor or a motor that needs to be recalibrated. Strenuous testing can eventually degrade motor performance. The vehicle’s takeoff for Flight 3 was less graceful than previous flights leading the author to believe a bad motor is the most likely cause.

Another metric for aggressiveness is the peak roll angle and roll rate achieved during flight. The roll angle, Fig. 5-10a, and roll rate, Fig. 5-10b, for Flight 3 are
Figure 5-6: Comparison of the feedforward model to the total commanded acceleration and jerk for Flight 2. Acceleration has a normalized error less than 34%. Jerk has a normalized error of 35%. The peak errors in acceleration and jerk coincide with the corners of the jerk command. This is because instantaneous jerk is not physically possible.

shown in Fig. 5-10. The vehicle reached a maximum roll angle of 49° and roll rate of 395°/s. Both of these are consistent with the acceleration and jerk limits specified for the flight test. The peak values are larger than work by other researchers, indicating these flight experiments are more challenging. There is good tracking throughout most of the flight until the very end of the trajectory. This is most likely caused by a damaged motor.

Fig. 5-12 shows a series of camera and RVIZ images as the vehicle dodges obstacles in Flight 3. Since the obstacles are outside the sensing range from the start location, the vehicle plans a straight trajectory toward the goal, Fig. 5-12a and Fig. 5-12c. The first obstacle is seen when the vehicle is approximately 2m away. The vehicle subsequently swerves once the first obstacle is seen, Fig. 5-12b and Fig. 5-12d. The
vehicle continues to avoid the obstacles until it reaches the goal.

5.5 Summary

This chapter presented three flight test results that demonstrated the perception, planning, and control system's ability to navigate through an unknown, cluttered environment. Flight 1, without feedforward, and Flight 2, with feedforward, served the purpose of evaluating the importance of feedforward. Though the planner was successful in identifying safe traversable regions that were collision free under prefect tracking, Flight 1 was deemed a failure because the vehicle clipped the first obstacle. The poor tracking performance, in the form of a delayed response, was the cause of the collision. Flight 2 had a tracking error nearly an order of magnitude less than that in Flight 1. The improved performance lead to a successful flight in Flight 2.

Flight 3 was designed to evaluate the system's performance at a higher speed in a more complicated obstacle course. The vehicle successfully navigated the obstacle field while reaching large attitudes and fast angular rates. More error was accumulated at the end of the flight than was observed in Flight 2. This was attributed to a damaged motor. Nonetheless, the vehicle still reached the goal without clipping

Figure 5-7: Obstacle configuration for Flight 3. The vehicle starts at the top of the image and flies towards a goal at the bottom. The vehicle cannot see the obstacles from its starting location nor are the obstacle locations known a priori.
Figure 5-8: Commanded and actual path for Flight 3. The vehicle was able to successfully navigate through the obstacle course. Note each obstacle has been inflated to account for the vehicle’s buffer size.

obstacles.
Figure 5-9: Comparison of the feedforward model to the total commanded acceleration and jerk for Flight 3. Acceleration has a normalized error of 31% during the collision avoidance maneuvers. Jerk has a normalized error of 28%. The peak errors in acceleration and jerk coincide with the corners of the jerk command. This is because instantaneous jerk is not physically possible. The large acceleration error at the end of the trajectory is attributed to a damaged motor.

Figure 5-10: Roll and roll rate tracking for Flight 3. Roll and roll rate are indicators of the aggressiveness of lateral maneuvers. The peak roll and roll rate values are consistent with the acceleration and jerk limits, respectively. The values are larger than reported by other researchers performing similar experiments.
Figure 5-11: Flight 2 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles.
Figure 5-11: Flight 2 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles.
Figure 5-11: Flight 2 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle’s initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles.
Figure 5-12: Flight 3 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle's initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles.
Figure 5-12: Flight 3 camera and RVIZ still images of the vehicle navigating through an obstacle course. The vehicle’s initial path is a straight line to the goal. Once the first obstacle is seen, the vehicle swerves to avoid it and subsequent obstacles.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presented a perception, planning, and control approach for high-speed navigation in unknown, cluttered environments. This is still an open problem in the robotics community because of the difficulty in perceiving the world and safely planning within it at high-speeds.

The three main contributions of this work are (1) an in-depth analysis of using feedforward control for high-speed navigation, (2) a low-latency perception and planning algorithm that generates minimum-time, state and input constrained trajectories using instantaneous perception data, and (3) flight experiments that verify the algorithm's ability to navigate in unknown, cluttered environments at relatively high-speeds.

Comprehensive background information on planning in known and unknown environments was given in Chapter 1. The limitations of past work was discussed in detail and how this work improves the state-of-the-art was highlighted.

Chapter 2 presented the low-level position and attitude controller used in this thesis. The cascaded feedback control architecture that maps position and velocity error to attitude setpoints for a quaternion-based, nonlinear attitude controller was then presented. A closed-form relation between the vehicle's jerk and angular rates was then derived. Substantial improvement in the vehicle's transient response to
commands when feedforward is used was then demonstrated. Lastly, stability for the attitude and position controller was proved.

Chapter 3 presented the approach taken for perception and planning. The limitations of previous work were addressed by extracting key features from perception data and identifying potentially traversable regions via clustering. A closed form solution was then derived for a minimum-time, decoupled optimal control problem for a third order system by relaxing the final position constraint. An efficient collision checking method using safety certificates that also captured state and obstacle uncertainty was then presented. It was then shown that a substantial increase in speed is possible if the vehicle is allowed to only swerve, as opposed to stop, to prevent collisions; this requires some domain knowledge though. Lastly, the computation times for clustering, cluster sorting, trajectory generation, and collision checking were all shown to be less than 200\(\mu s\) with a total run-time between 0.5-1ms; almost two orders of magnitude faster than state-of-the-art algorithms.

The hardware and software used in this thesis was presented in Chapter 4. A combination of commercial and custom components allowed for on-board perception, planning, and position control. A lightweight, credit card sized single board computer was used to process laser scan data, plan trajectories, and perform position and velocity feedback. The maximum acceleration and jerk the vehicle is capable of were also identified. A custom autopilot that performs high-rate estimation and feedback on the attitude and angular rate commands from the outer-loop control was presented. Lastly, the Measurable Augmented Reality system that provides useful context to experiments was also introduced.

Chapter 5 presented three flight test results that demonstrated the perception, planning, and control system's ability to navigate through a unknown, cluttered environment. Flight 1, without feedforward, and Flight 2, with feedforward, served the purpose of evaluating the importance of feedforward. Flight 2 had a tracking error nearly an order of magnitude less than that in Flight 1. Flight 3 was designed to evaluate the system's performance at a higher speed in a more complicated obstacle course. The vehicle successfully navigated the obstacle field while reaching large
6.2 Future Work

There are a number of directions this work can go. The first of which is quantifying the planner’s performance with different sensing ranges and at different speeds. Section 3.5 showed there are three distinct flight regimes: safe, marginally safe, and inevitable collision state. Marginally safe is the regime where the vehicle cannot stop to avoid an obstacle, it can only swerve. Flight experiments in this regime would be useful in understanding how the planner and vehicle perform and whether operating in this regime is even viable. Another interesting exercise would be to apply machine learning to generate a speed profile through randomly generated obstacle fields. An interesting result would be in which flight regime the machine learning algorithm places the vehicle.

Another interest is expanding the planner to 3D. This entails switching from the Hokuyo laser scanner to a 3D sensor. One sensor that has been recently released is the Intel RealSense R200 [57]. The R200 is a light-weight, small form factor RGB-D camera. The planning approach taken in this thesis would have to be modified to accommodate 3D perception data. For instance, the clustering approach in Section 3.3 does not scale to 3D because of the increased size of the perception data. An idea worth exploring is directly sampling the pointcloud to generate points that are used in trajectory generation. These points would serve the same role as the clusters in Section 3.3. The sampling approach could be random or biased. The samples would be assigned a cost and iterated through until a collision-free trajectory is found. Collision checking using safety certificates could still be used if the pointcloud is transformed to a $k$-$d$ tree.

Dynamic jerk allocation for the minimum-time optimal control problem in Section 3.4 is also worth exploring. In this work, jerk was allocated equally to all three $x$, $y$, and $z$ axes. As pointed out by [16], this makes the solution suboptimal since the same jerk is applied independent of the change in desired velocity. It may be
desirable to allocate less jerk for small changes in desired velocity. It's possible that
dynamic allocation will increase the computation time and complexity. The potential
increase in vehicle performance will have to outweigh the increased computation time
to justify its use.
Bibliography


