Enhancing Quadcopter Capabilities via Design and Control

by

Nathan Leo Bucki

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Assistant Professor Mark Mueller, Chair
Professor Kameshwar Poolla
Associate Professor Anil Aswani

Fall 2021

Enhancing Quadcopter Capabilities via Design and Control

Abstract

Enhancing Quadcopter Capabilities via Design and Control

by

Nathan Leo Bucki

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Assistant Professor Mark Mueller, Chair

In recent years, quadcopters have gained remarkable popularity in a wide range of industries. Given that the utility of quadcopters has already been extensively demonstrated, this dissertation explores changes to the design and control of conventional quadcopters which either enable operation in new environments, allow for novel tasks to be performed, or reduce computational hardware requirements.

Two novel designs are presented in this dissertation. First, we explore the use of angular momentum to reduce the sensitivity of a vehicle to torque disturbances. We show both theoretically and experimentally how torque disturbance sensitivity monotonically decreases with increasing net angular momentum of the vehicle when using an appropriately designed controller, and discuss how this effect scales with vehicle size. Second, we consider the use of passive (i.e. unactuated) mechanisms to expand the types of tasks a quadcopter can perform. Specifically, we demonstrate how a vehicle with freely rotating arms can change shape mid-flight, allowing for traversal of narrow passageways, simple manipulation, and perching behaviors.

Next, two algorithms are presented which enable autonomous flight in known and unknown environments. Both algorithms are focused on improving the time required to check whether a given trajectory collides with the environment, reducing the computational power requirements of onboard computers used the fly the vehicle. The first method is used to quickly determine whether a given trajectory collides with a convex obstacle, enabling the avoidance of both static and dynamic obstacles. The second method allows for operation in previously unseen environments by using depth images from an onboard sensor to represent the environment. Rectangular pyramids are used to partition the free-space of the depth image, which enable highly efficient collision checking between trajectories and the environment.

Each proposed design and algorithm is demonstrated experimentally on custom hardware, and relevant code has been made publicly available where appropriate.

Contents

Acknowledgments

I would like to first and foremost thank my parents for their unwavering support not only during my time at UC Berkeley, but throughout my entire academic career. Your selfless encouragement to pursue my own path has been an invaluable foundation for growth in all parts of my life, and for this I will be forever grateful.

I am also particularly grateful for the fantastic mentorship given by my advisor Mark Mueller. I appreciate your guidance in areas both related to my research as well as my own personal goals, and I hope to continue to uphold the same level of excellence you instilled in me throughout the rest of my career.

To all of my labmates and friends at UC Berkeley, thank you for making my time here so memorable and fun. Your help and friendship has been a constant source of encouragement that I will not forget.

Finally, this work would not have been possible without funding from the following sources: the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1752814, the Berkeley Fellowship for Graduate Study, the Berkeley DeepDrive project 'Autonomous Aerial Robots in Dense Urban Environments', China High-Speed Railway Technology Co., Ltd, and the Powley Fund. The experimental testbed at the HiPeRLab is the result of contributions of many people, a full list of which can be found at `hiperlab.berkeley.edu/members/`.

Chapter 1

Introduction

Quadcopters (also know as quadrotors, quadrocopters, or sometimes colloquially as drones) are a class of aerial vehicle which use four fixed-pitch propellers to perform various aerial maneuvers, typically including vertical takeoff and landing, hovering, and agile flight. An example of a typical quadcopter is shown in Figure 1.1. These vehicles have been shown to be capable of performing a number of useful tasks, including search and rescue, package delivery, cinematography, inspection, as well as many additional functions. Although quadcopters have been proven to be useful machines, their capabilities are inherently limited (like all robotic systems) both by their physical design (e.g. their dynamic properties and size/shape) as well as the algorithms used to control them (e.g. due to limited available onboard computational power).

This work seeks to expand the capabilities of quadcopters on both fronts. First, we

Figure 1.1: Standard quadcopter equipped with various sensors.

explore how relatively minor design changes to the physical structure of a quadcopter can allow the vehicle to operate in environments and perform tasks not previously achievable using conventional designs. Then, several novel algorithms are presented which enable vehicles with severely limited onboard computational power to operate autonomously in cluttered environments. The utility of the proposed design changes and control algorithms are compared to those of a conventional quadcopter, allowing for a clear assessment of their usefulness. For example, if a proposed design requires e.g. a substantially larger vehicle mass than normal, it will be shown that the improvement in the effectiveness of the vehicle outweighs any adverse effects of the change (e.g. reduced flight time). However, as shown the the following chapters, a majority of the techniques proposed in this dissertation either improve vehicle capabilities (e.g. by enabling vehicles to perform novel tasks) or reduce hardware requirements (e.g. by reducing onboard computation) in exchange for relatively minor and often negligible trade-offs in other aspects of the vehicle design. Finally, because this dissertation is primarily concerned with techniques that tangibly impact the utility of quadcopters, each concept proposed in this dissertation is demonstrated using flight experiments.

The remainder of this chapter seeks to (1) provide examples of tasks which quadcopters (and multirotors in general) are particularly well-suited to accomplish, (2) discuss the fundamental design choices that enable quadcopters to perform these tasks, and (3) examine the shortcomings of quadcopters due to these fundamental design choices. Finally, an outline of this dissertation is presented at the end of the chapter, highlighting how the presented shortcomings can be addressed.

1.1 Quadcopter Applications

Recently quadcopters have become the focus of a significant amount of research effort, leading to a large number of different commercial applications. Perhaps the most widely used application of the quadcopter is as a mobile platform for maneuvering various sensors into difficult to reach areas. For example, quadcopters are favorable platforms for performing search and rescue operations in environments that may be difficult for humans or ground-based robots to survey in a timely manner [1] [2]. Similarly, quadcopters are ideal for monitoring dynamically changing disasters such as wildfires, as they can be rapidly deployed from the ground in order to provide information to first responders regarding the extent and location of the disaster in real-time [3].

A more routine use of quadcopters is to perform aerial imaging [4], which can be used to quickly obtain e.g. geographic/topological data of a large section of land [5]. Furthermore, quadcopters are well-suited to perform surveillance operations such as monitoring crowds [6] or international borders [7]. Commercially, quadcopters have also been shown to be exceptionally useful in performing inspections of buildings and infrastructure that would otherwise be difficult and/or hazardous to inspect manually [8].

The ability for quadcopters to be flown autonomously through cluttered environments has additionally become a source of great interest in recent years. For example, the ability to

fly in close proximity to obstacles allows for e.g. higher resolution images to be taken during inspection tasks or for the vehicle to fly though complex environments such as tunnels or other indoor spaces [9]. Such flight often requires traversal of tight spaces, and the agility of the quadcopter has been show to enable the traversal of narrow gaps as shown in [10] and [11].

1.2 Quadcopter Design

Although there are many reasons that quadcopters have become commonly used tools to accomplish the tasks described in the previous section, here we focus on the following two key aspects of their design:

- No complex mechanisms are required for basic operation
- All components required for basic operation are relatively low-cost

In other words, when compared to other methods of accomplishing the same task, quadcopters have the advantage of generally being simple and cheap.

The simplicity of the quadcopter is evident when comparing it to another vehicle capable of vertical takeoffs and landings: the helicopter. Unlike the quadcopter, a helicopter requires the use of a swashplate mechanism to control the pitch of the main rotor of the vehicle, leading to increase manufacturing and maintenance requirements for the vehicle. In contrast, typically the only moving parts of a quadcopter are the four motors that drive its fixed-pitch propellers. Similarly, although a fully actuated aerial vehicle design (i.e. a vehicle capable of producing forces and torques in three dimensions independently from one another) can be achieved by using additional propellers and/or mechanisms, the quadcopter design has persisted as the dominant multi-rotor vehicle design at least in part due to its simplicity. By using only four fixed-pitch propellers, the conventional quadcopter design minimizes mechanical complexity while still allowing the vehicle to perform vertical takeoffs and landings. This lack of complex mechanisms not only reduces the manufacturing cost of the vehicle (as the same motor design can be used to drive all four propellers), but improves reliability due to the lack of extraneous moving parts.

The low cost of quadcopters has also been cited as a reason for their current ubiquity [12]. Because most components (e.g. motors, propellers, batteries, fight controllers) can be manufactured at scale, quadcopters can be built for relatively low costs when compared to other robotic systems that can achieve similar tasks [13]. This has allowed for the widespread adoption of quadcopters as not only industrial tools, but as platforms for various forms of research and entertainment.

Thus, in this dissertation, methods of improving the usefulness of quadcopters are developed which do not significantly reduce the simplicity of the vehicle or increase its cost, ensuring that any proposed changes to the conventional quadcopter design are aligned with the original reasons for their widespread adoption.

1.3 Quadcopter Limitations

Partially due to their relatively simple design and low cost, quadcopters have several inherent limitations which we seek to improve in this work. First, we note that their ability to reject disturbances is limited by the maximum forces and torques that can be produced by the propellers of the vehicle. This fundamental limitation prevents quadcopters from operating in certain harsh environments that other robotic systems may be able to work in. For example, because quadcopters are generally designed to be lightweight, they are very susceptible to wind disturbances that can push them away from a desired position.

Another basic operational limitation relates to the size of the vehicle. Because it is often desirable to operate quadcopters in cluttered environments with many nearby obstacles, their size must not be so large as to prevent them from traversing areas that require them to fly in close proximity to obstacles. Thus, there are spaces which can be inaccessible to the vehicle due to its size, which is typically governed by the size and mass of sensors and other actuators that the vehicle is carrying.

Finally, quadcopters commonly require significant onboard computational power in order to operate autonomously in cluttered environments. Because it is favorable to use lightweight, low-power computers onboard the vehicle in order to minimize energy usage, the maneuverability of the vehicle can be limited by how fast such computers can analyze incoming sensor data and compute the appropriate control actions needed to avoid obstacles. Thus, quadcopters capable of high-speed flight through cluttered environments typically require heavier, higher-power computers in order to be able to sense and act quickly enough to avoid collisions.

In the following chapters we propose several changes to the design and control of quadcopters which address these limitations, and discuss the trade-offs associated with each proposed change.

1.4 Dissertation Outline

The dissertation is organized as follows.

Chapter 2: Quadcopter Dynamics and Control. First, the basic quadcopter dynamics model (and associated notation) is introduced for use in subsequent chapters. The model is derived using Newtonian mechanics, and is used as a basis for deriving the dynamics models of the modified quadcopter designs presented in Chapters 3 and 4. This basic model is also used to motivate the trajectory generation techniques described in Chapters 5 and 6. Finally, a standard method for controlling quadcopters is presented, which will be built upon in subsequent chapters.

Chapter 3: Enhanced Disturbance Rejection via Angular Momentum. In this chapter a multicopter augmentation capable of improving the vehicle's torque disturbance rejection capabilities is described, analyzed, and demonstrated. A momentum wheel is used to increase the total angular momentum of the vehicle, which we show strictly improves the

torque disturbance rejection capabilities of the vehicle. A video of the proposed vehicle can be viewed at `https://youtu.be/C2fj2D_2pI8`.

Chapter 4: Improved Operational Capabilities via Aerial Morphing. Next, we describe another quadcopter design modification that enables the vehicle to change shape mid-flight. In this case, the normally rigid connections where the four arms meet the central body of the vehicle are replaced by unactuated hinges which have an approximately 90° range of motion. Using a specially developed controller, these additional unactuated degrees of freedom are shown to enable the vehicle to perform a number of tasks that a conventional quadcopter cannot perform. This ability of the vehicle to change shape is shown to enable the traversal of narrow passages, simple grasping, and perching. A video demonstrating an experimental vehicle performing each of these abilities can be found at `https://youtu.be/xEg8GXlb82g`.

Chapter 5: Computationally Efficient Trajectory Generation in Known Environments. In the remainder of the dissertation, computationally efficient trajectory generation methods are presented which enable low-power computers to be used to perform obstacle avoidance. In this chapter a computationally efficient method for checking whether a given quadcopter trajectory (modeled as a fifth order polynomial in time) collides with a convex obstacle. Due to the efficiency of the proposed collision checking method, many candidate trajectories can be generated and evaluated for collisions with the environment in a very short period of time, allowing for new trajectories to be generated in real-time. A video demonstrating this method can be found at `https://youtu.be/cpskvQPhpoY`.

Chapter 6: Computationally Efficient Trajectory Generation in Unknown Environments. Next, a similar method to that presented in Chapter 6 is presented which is capable of quickly finding a collision-free trajectory given a single depth image taken by an onboard camera. The proposed method relies upon on an efficient representation of the free space of the depth image, namely rectangular pyramids. These pyramids allow for efficient trajectory collision checking to be performed, allowing for a quadcopter to fly autonomously through a previously unseen cluttered environment while replanning at high rates using a low-power onboard computer. A video demonstrating the method can be found at `https://youtu.be/Pp-HIT9S6ao`.

Chapter 7: Conclusions. Finally, the overarching findings of this dissertation are summarized.

Chapter 2

Quadcopter Dynamics and Control

In this section a derivation of the nonlinear dynamics of a quadcopter and a standard quadcopter controller are presented. A rigid body model is used in deriving the dynamics, and it is assumed that the only forces and torques acting on the vehicle are those due to gravity and the thrusts and torques produced by each propeller. This dynamics model is used as a basis for comparison to the proposed vehicle designs presented in Chapters 3 and 4, and is used to motivate the trajectory generation algorithms presented in Chapters 5 and 6. The controller presented in this chapter is a commonly used method for controlling quadcopters, and is used as a basis for developing the modified quadcopter controllers presented in Chapters 3 and 4.

2.1 Notation

Non-bold symbols such as m represent scalars, lowercase bold symbols such as $\boldsymbol{g}$ represent first order tensors (vectors), and uppercase bold symbols such as $\boldsymbol{J}$ represent second order tensors (matrices). Subscripts such as m_B represent the body to which the symbol refers, and superscripts such as $\boldsymbol{g}^E$ represent the frame in which the tensor is expressed. A second subscript or superscript such as $\boldsymbol{\omega}_{BE}$ or $\boldsymbol{R}^{BE}$ represents what the quantity is defined with respect to. The symbol $\boldsymbol{d}$ represents a displacement, $\boldsymbol{\omega}$ represents an angular velocity, and $\boldsymbol{R}$ represents a rotation matrix. The skew-symmetric matrix form of the cross product is written as $\boldsymbol{S}(\boldsymbol{a})$ such that $\boldsymbol{S}(\boldsymbol{a})\,\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b}$.

We define $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ to be unit vectors such that e.g. $\boldsymbol{x}_B$, $\boldsymbol{y}_B$, and $\boldsymbol{z}_B$ are unit vectors that point in the x, y, and z directions of frame B respectively.

2.2 Model

Let E be the inertial frame, and B be a frame fixed to the body of the quadcopter as shown in Figure 2.1. The rotation matrix of frame B with respect to frame E is defined as $\boldsymbol{R}^{BE}$

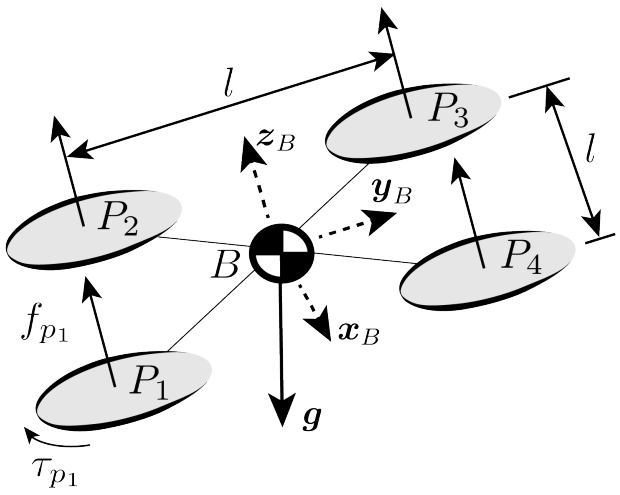

Figure 2.1: Model of a conventional quadcopter. Each propeller produces force f_{p_i} and torque τ_{p_i} at point P_i. The center of mass of the vehicle is equidistant from each propeller at point C, and the distance between adjacent propellers is l.

such that the quantity $\boldsymbol{v}^B$ expressed in the B frame is equal to $\boldsymbol{R}^{BE}\boldsymbol{v}^E$ where $\boldsymbol{v}^E$ is the same quantity expressed in frame E.

When used in a subscript of a displacement tensor or its time derivatives, E is defined as a fixed point in the inertial frame and B is defined as the center of mass of the vehicle. For example, $\boldsymbol{d}^E_{BE}$ represents the displacement of the center of mass of the vehicle with respect to a fixed point in the inertial frame, and is expressed in the inertial frame E.

The mass and mass moment of inertia of the vehicle taken at the center of mass of the vehicle are denoted m_B and $\boldsymbol{J}_B$ respectively. Additionally, the vehicle has four propellers labeled $i \in \{1, 2, 3, 4\}$, each of which produces a scalar thrust force f_{p_i} and aerodynamic reaction torque τ_{p_i} in the $\boldsymbol{z}_B$ direction. We assume that the torque produced by each propeller is piecewise linearly related to the propeller thrust force [14] with positive proportionality constants $\kappa^+$and κ^-such that:

$$\tau_{p_i} = \begin{cases} (-1)^i \kappa^+ f_{p_i} & f_{p_i} \geq 0 \\ (-1)^i \kappa^- f_{p_i} & f_{p_i} < 0 \end{cases} \tag{2.1}$$

where $(-1)^i$ models the handedness of the propellers, $f_{p_i} < 0$ when the propellers are spun in reverse, and $\kappa^+ \neq \kappa^-$ when asymmetric propellers are used, as is common with quadcopters. Note that for conventional quadcopters f_{p_i} is typically constrained to be positive due to hardware limitations, though in special cases (such as the vehicle described in Chapter 4) negative thrust forces are allowed. Finally, let P_i be a point along the thrust axis of propeller i such that $\boldsymbol{d}_{P_iB}$ describes the displacement of a fixed point along thrust axis of propeller i with respect to the center of mass of the vehicle.

2.3 Dynamics

The translational and rotational dynamics of the vehicle are derived using Newton's second law and Euler's law respectively [15]. The time derivative of a vector is taken in the reference frame in which that vector is expressed. Let $\boldsymbol{g}$ be the acceleration due to gravity.

The translational dynamics of the vehicle (written in the inertial frame E) are then

$$m_B \ddot{\boldsymbol{d}}_{BE}^E = m_B \boldsymbol{g}^E + \boldsymbol{R}^{EB} \sum_{i=1}^{4} \boldsymbol{z}_B^B f_{p_i} \tag{2.2}$$

and the rotational dynamics (written in the body-fixed frame B) are then

$$\boldsymbol{J}_B^B \dot{\boldsymbol{\omega}}_{BE}^B + \boldsymbol{S}\big(\boldsymbol{\omega}_{BE}^B\big)\, \boldsymbol{J}_B^B \boldsymbol{\omega}_{BE}^B = \sum_{i=1}^{4} \big(\boldsymbol{S}\big(\boldsymbol{d}_{P_iB}^B\big)\, \boldsymbol{z}_B^B f_{p_i} + \boldsymbol{z}_B^B \tau_{p_i}\big) \tag{2.3}$$

Individual thrust force computation

In order to simplify the controller synthesis presented in the following subsection, we next rewrite (2.2) and (2.3) to be functions of the intermediate control inputs f_Σ and $\boldsymbol{\tau}^B$. The individual propeller thrust forces $u = (f_{p_1}, f_{p_2}, f_{p_3}, f_{p_4})$ are related to the desired total thrust in the $\boldsymbol{z}_B$ direction f_Σ and the desired torques about the axes of the body-fixed B frame, $\boldsymbol{\tau}^B = (\tau_x, \tau_y, \tau_z)$ as follows:

$$\begin{bmatrix} f_\Sigma \\ \boldsymbol{\tau}^B \end{bmatrix} = \begin{bmatrix} M_{f_\Sigma} \\ M_{\boldsymbol{\tau}^B} \end{bmatrix} u = Mu \tag{2.4}$$

where $M_{f_\Sigma} \in \mathbb{R}^{1\times 4}$ is the mapping from u to f_Σ, $M_{\boldsymbol{\tau}^B} \in \mathbb{R}^{3\times 4}$ is the mapping from u to $\boldsymbol{\tau}^B$, and $M \in \mathbb{R}^{4\times 4}$ is the combined mapping.

The mapping M is computed using the geometry of the vehicle and the torque produced by each propeller as a function of the thrust it produces. Let $\boldsymbol{d}_{P_iB}$ be the position of propeller P_i relative to the center of mass of the entire vehicle B, and $\kappa_{p_i} = (-1)^i \kappa^+$ or $\kappa_{p_i} = (-1)^i \kappa^-$ depending on the thrust direction of propeller i as defined in (2.1). Then, the i-th columns of M_{f_Σ} and $M_{\boldsymbol{\tau}^B}$ are

$$M_{f_\Sigma}[i] = 1, \quad M_{\boldsymbol{\tau}^B}[i] = \boldsymbol{S}\big(\boldsymbol{d}_{P_iB}^B\big)\, \boldsymbol{z}_B^B + \kappa_{p_i} \boldsymbol{z}_B^B \tag{2.5}$$

where we recall that each propeller produces thrust in the $\boldsymbol{z}_B^B$ direction.

For a conventional quadcopter with l defined as shown in Figure 2.1, this mapping is defined as M_u:

$$M_u = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -l/2 & -l/2 & l/2 & l/2 \\ -l/2 & l/2 & l/2 & -l/2 \\ -\kappa^+ & \kappa^+ & -\kappa^+ & \kappa^+ \end{bmatrix} \tag{2.6}$$

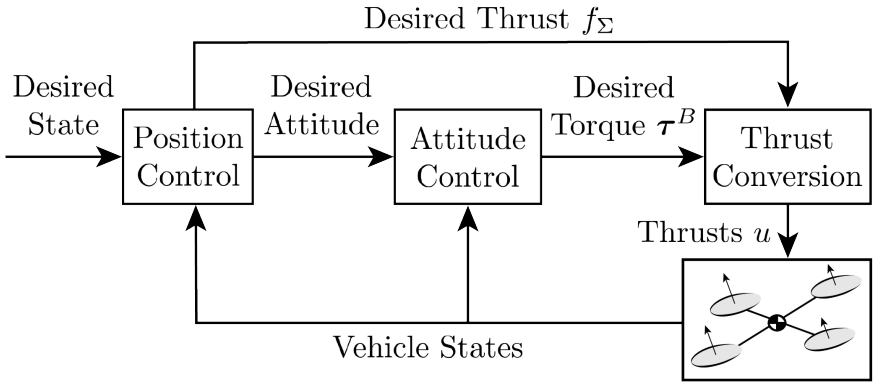

Figure 2.2: Cascaded controller used to control the vehicle.

where we note that $|\kappa_{p_i}| = \kappa^+$ for all propellers, as the propellers of a conventional quadcopter are typically restricted to only spin in one direction.

By employing this mapping, the dynamics of the vehicle can be rewritten such that the only control inputs affecting the translational and rotational dynamics are f_Σ and $\boldsymbol{\tau}^B$ respectively:

$$m_B \ddot{\boldsymbol{d}}^E_{BE} = m_B \boldsymbol{g}^E + \boldsymbol{R}^{EB} \boldsymbol{z}^B_B f_\Sigma \tag{2.7}$$

$$\boldsymbol{J}^B_B \dot{\boldsymbol{\omega}}^B_{BE} + \boldsymbol{S}\left(\boldsymbol{\omega}^B_{BE}\right) \boldsymbol{J}^B_B \boldsymbol{\omega}^B_{BE} = \boldsymbol{\tau}^B \tag{2.8}$$

2.4 Control

A cascaded control structure, shown in Figure 2.2, is used to control the vehicles presented in this dissertation (any exceptions in the following chapters are clearly noted). A position controller first computes a desired acceleration based on position and velocity errors, allowing for the computation of the desired total thrust in the $\boldsymbol{z}_B$ direction, f_Σ. Then, an attitude controller computes the desired torque required to align the thrust direction $\boldsymbol{z}_B$ with the desired acceleration direction and achieve the desired yaw angle. Finally, the individual propeller thrust forces necessary to generate the desired total thrust and desired body torque are computed. For each propeller, the desired thrust is converted to a desired angular velocity, which an electronic speed controller is used to track.

Position control

Let $\boldsymbol{d}^E_{BB_d}$ represent the difference between the desired and current position, $\dot{\boldsymbol{d}}^E_{BB_d}$ represent the difference between the desired and current velocity, and $\ddot{\boldsymbol{d}}^E_{BB_d}$ represent the desired

acceleration of the vehicle. The linear system $\dot{s}_p = A_p s_p + B_p u_p$ is then defined as follows

$$s_p = (\boldsymbol{d}^E_{BB_d}, \dot{\boldsymbol{d}}^E_{BB_d}), \qquad u_p = \ddot{\boldsymbol{d}}^E_{BB_d} \tag{2.9}$$

$$A_p = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \qquad B_p = \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{2.10}$$

An infinite-horizon LQR controller [16] is used to compute the desired acceleration $\ddot{\boldsymbol{d}}^E_{BB_d}$ using this linear system. The desired thrust f_Σ and thrust direction are then computed from the desired acceleration as follows.

$$f_\Sigma = m_B ||\ddot{\boldsymbol{d}}^E_{BB_d} - \boldsymbol{g}^E||_2, \qquad \boldsymbol{z}^E_{B_d} = \frac{\ddot{\boldsymbol{d}}^E_{BB_d} - \boldsymbol{g}^E}{||\ddot{\boldsymbol{d}}^E_{BB_d} - \boldsymbol{g}^E||_2} \tag{2.11}$$

Using the the desired acceleration direction $\boldsymbol{z}^E_{B_d}$ and a desired yaw angle, the desired attitude of the vehicle is defined as the attitude at which $\boldsymbol{z}_B$ becomes aligned with $\boldsymbol{z}^E_{B_d}$ and the desired yaw angle is achieved.

Attitude control

The attitude controller is designed using desired first-order behavior, described here by the rotation vector $\boldsymbol{r} = (\phi_e, \theta_e, \psi_e)$ that represents the rotation between the current and desired attitude (i.e. a rotation about the axis defined in the direction of $\boldsymbol{r}$ by angle $||\boldsymbol{r}||$). Note that, to first order, ϕ_e, θ_e, and ψ_e represent roll, pitch, and yaw respectively. The desired attitude is defined as that attitude at which the yaw angle of the vehicle matches the desired yaw angle and at which the thrust direction of the vehicle matches the desired thrust direction, which is given by the position controller (see Figure 2.2).

The linearized attitude dynamics of the vehicle are then

$$\begin{bmatrix} \dot{\boldsymbol{r}} \\ \ddot{\boldsymbol{r}} \end{bmatrix} = A \begin{bmatrix} \boldsymbol{r} \\ \dot{\boldsymbol{r}} \end{bmatrix} + B\boldsymbol{\tau}^B \tag{2.12}$$

where

$$A = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}, \quad B = \begin{bmatrix} \boldsymbol{0} \\ (\boldsymbol{J}^B_B)^{-1} \end{bmatrix} \tag{2.13}$$

and where we recall that $\boldsymbol{J}^B_B$ is the moment of inertia of the entire vehicle written at its center of mass.

A second infinite-horizon LQR controller with state cost matrix $Q \in \mathbb{R}^{6\times 6}$ and input cost matrix $R_{\boldsymbol{\tau}^B} \in \mathbb{R}^{3\times 3}$ is used to minimize the attitude error. For each configuration of the vehicle, we weight the cost of each state error independently such that Q is a diagonal matrix. The values of the diagonal of Q are chosen such that the costs associated with ϕ_e and θ_e (i.e. elements 1 and 2) are equal and such that the costs associated with the roll rate and pitch rate (i.e. elements 4 and 5) are equal. However, we choose to define the input

cost matrix $R_{\boldsymbol{\tau}^B}$ using the mapping $M_{\boldsymbol{\tau}^B}$ from the individual thrust forces u to the desired torque $\boldsymbol{\tau}^B$ as defined in (2.4) (i.e. the lower three rows of M_u).

$$R_{\boldsymbol{\tau}^B} = (M^+_{\boldsymbol{\tau}^B})^T R_u M^+_{\boldsymbol{\tau}^B} \tag{2.14}$$

where $M^+_{\boldsymbol{\tau}^B}$ is the pseudoinverse of the mapping matrix $M_{\boldsymbol{\tau}^B}$, and $R_u \in \mathbb{R}^{4\times4}$ is a diagonal matrix that encodes the cost associated with the thrust force produced by each propeller. For a conventional quadcopter, R_u is typically chosen to be a diagonal matrix with equal costs along the diagonal as each propeller typically has nearly identical aerodynamic properties.

By defining the input cost matrix $R_{\boldsymbol{\tau}^B}$ as a function of the mapping matrix $M_{\boldsymbol{\tau}^B}$, we can straightforwardly synthesize different infinite-horizon LQR attitude controllers for e.g. vehicles of different sizes. Furthermore, the torque cost matrix $R_{\boldsymbol{\tau}^B}$ can be used to analyze the ability of the vehicle to control its attitude in different configurations, as it describes the cost of producing an arbitrary torque on the vehicle while implicitly accounting for the geometry of the vehicle due to its dependence on $M_{\boldsymbol{\tau}^B}$.

Chapter 3

Enhanced Disturbance Rejection via Angular Momentum

In this chapter we present a novel quadcopter design that utilizes a momentum wheel to reduce the sensitivity of the vehicle to torque disturbances. As described in Chapter 1, the proposed vehicle is designed to require minimum modifications to the design of a conventional quadcopter such that new capabilities can be realized (i.e. improved torque disturbance rejection). The mechanical design, coupled with intelligent feedback control, allows for operation of autonomous aerial systems in challenging environments where conventional designs may fail. We show that sensitivity to torque disturbances monotonically decreases with increasing angular momentum, and the effect scales such that a greater improvement in torque disturbance sensitivity is experienced by smaller vehicles. For a fixed vehicle size, a trade-off exists between the added torque disturbance rejection capability, the power required to carry the wheel's added mass, and the kinetic energy of the rotating wheel. A cascaded controller structure is proposed that accounts for the additional angular momentum and that accelerates or decelerates the momentum wheel to gain additional control authority in yaw. Theoretical results are validated experimentally using two vehicles of different scales. The proposed vehicle design is likely to be of value in situations where precision control is required in the face of large disturbances.

Note that the material presented in this chapter is based on the following previously published work. This chapter is primarily based upon the latter paper (which is an extension of the former), and differs primarily in the notation used to describe the dynamics of the vehicle.

- Nathan Bucki and Mark W Mueller. "Improved Quadcopter Disturbance Rejection Using Added Angular Momentum". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018
- Nathan Bucki and Mark W Mueller. "A novel multicopter with improved torque disturbance rejection through added angular momentum". In: *International Journal of Intelligent Robotics and Applications* 3.2 (2019), pp. 131–143

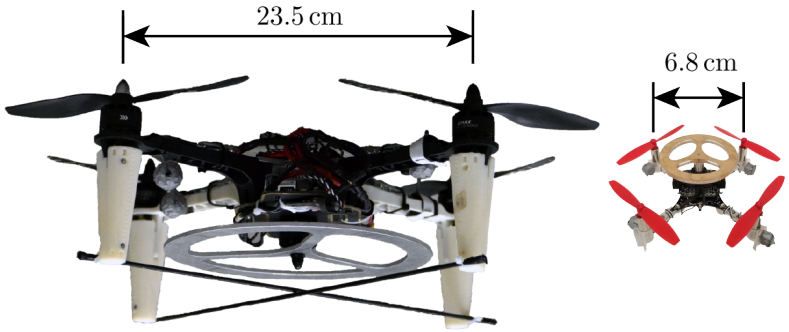

Figure 3.1: Quadcopters of different sizes with added momentum wheels. Each momentum wheel is driven by a dedicated motor and spins about the thrust direction of the vehicle.

3.1 Introduction

As discussed in Chapter 1, multicopters are used to perform a variety of tasks such as aerial imaging, environmental monitoring, building inspection, and search and rescue. However, in challenging conditions multicopters may be unable to perform adequately, due to (e.g.) the danger posed by poor tracking performance. For example, multicopters may struggle in high wind shear environments, or environments with flying debris (e.g. hail storms).

Several controllers have been developed that improve the disturbance rejection capabilities of multicopters. A method for estimating and compensating for wind disturbances acting on quadcopters was presented by [19], and a sliding mode controller used in conjunction with a sliding mode disturbance observer was presented by [20] in order to improve robustness to unknown disturbances. [21] use a nonlinear adaptive state feedback controller to track trajectories in the presence of constant force disturbances, and [22] develop an attitude controller and disturbance observer to compensate for time varying disturbances.

Although existing disturbance-observer-based controllers improve the disturbance rejection capabilities of the system, the performance of these controllers is inherently limited by the system dynamics, sensor noise, and by the available range of control inputs. To improve disturbance rejection beyond what is possible by changing the controller, it is necessary to adapt the system's design. One such change is to increase the angular momentum in the thrust direction of the multicopter by attaching a momentum wheel that spins about the thrust axis of the multicopter (e.g. a momentum wheel as shown in Figure 3.1). The additional angular momentum aids in the rejection of torque disturbances, enhancing the ability of the vehicle to fly in environments with torque disturbances.

The idea of using angular momentum to improve attitude control was first rigorously studied in the context of dual-spin spacecraft, which are vehicles that consist of two bodies rotating about a shared spin axis in order to maintain a desired attitude. Attitude stability criteria for dual-spin spacecraft is presented in [23], and the effects of energy dissipation on dual-spin spacecraft is presented in [24]. However, the above consider the stabilizing effect of

angular momentum on the orientation of spacecraft and do not focus on the effect of angular momentum on the translational dynamics of the vehicle.

Unlike a satellite in free-fall, the translation and orientation of a multicopter are strongly coupled, and thus the effect of angular momentum on the translational dynamics of multicopters must be considered in order to perform stable flight. Several multicopter-based vehicles have been proposed that exhibit stable flight with a significant amount of angular momentum. In [25], stability criteria are developed for a class of vehicles with a single propeller and passive stabilizing mechanisms, and the contribution of the vehicle's angular momentum to its stability is discussed. A method for controlling a quadcopter despite the loss of one, two, or three of its propellers is presented in [26] that involves the vehicle gaining significant angular momentum in order to hover, and this idea has been further investigated in [27], which presents an aerial vehicle that rotates parallel to the direction of gravity using only a single propeller. Furthermore, in [28] a novel aerial vehicle design is presented that uses one large propeller and three smaller propellers to enable more energy efficient flight compared to similar sized quadcopters. Due to the relative size and rotation directions of the propellers, the vehicle has a nonzero net angular momentum.

In this chapter we focus on how a large source of angular momentum can be used to enhance the torque disturbance rejection capabilities of a multicopter rather than treating any angular momentum as an unfortunate secondary effect of the vehicle design. In addition to these considerations, we emphasize the fact that for our proposed vehicle the effect of the momentum wheel can be changed mid-flight by changing the speed at which the wheel spins. Mid-flight changes to the dynamics of multicopters have also been explored, for example, in [29] and [30]. In [29] a quadcopter with tilting propellers is presented that is able to change the thrust direction of the vehicle without changing its attitude, and in [30] a quadcopter capable of changing the length and orientation of its arms is presented. Although these vehicles are not specifically designed to improve the disturbance rejection capabilities of a multicopter, they allow the vehicle to perform maneuvers that a standard multicopter cannot.

Regarding the proposed novel vehicle, we present the following in this chapter:

- an analysis of the disturbance rejection capabilities of the vehicle that includes position and velocity,
- a comparison between the force and torque disturbance rejection capabilities of the system,
- an analysis of how disturbance rejection scales with vehicle size,
- an analysis of the robustness of the control law to errors in the estimate of the total angular momentum of the vehicle,
- a new control law that leverages the source of angular momentum to improve the yaw control authority of the vehicle,

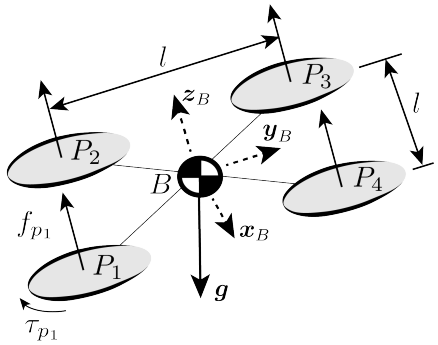

Figure 3.2: Model of a quadcopter with an added momentum wheel. The vehicle is nearly identical to a conventional quadcopter (described in Chapter 2), but has a momentum wheel that rotates about the $\boldsymbol{z}_B$ axis with angular velocity ω_W.

- additional experiments with a small scale quadcopter that verify how the disturbance rejection capabilities scale with vehicle size

3.2 Dynamics

The dynamics of the vehicle largely match the dynamics of a conventional quadcopter (presented in Chapter 2), but include additional terms related to the momentum wheel. Figure 3.2 shows a model of the proposed vehicle including the momentum wheel.

The momentum wheel is assumed to rotate about its center of mass, and the bodies are assumed rigid except for their relative rotation. Because the momentum wheel does not produce any external force on the vehicle while spinning, the translational dynamics of the vehicle are unchanged from those of a conventional quadcopter. That is, the translational dynamics of the proposed vehicle are identical to those presented in (2.2):

$$m_B \ddot{\boldsymbol{d}}^E_{BE} = m_B \boldsymbol{g}^E + \boldsymbol{R}^{EB} \sum_{i=1}^{4} \boldsymbol{z}^B_B f_{p_i} \tag{3.1}$$

where m_B is the mass of the entire vehicle including the momentum wheel.

The total mass moment of inertia of the vehicle (including the momentum wheel) is denoted $\boldsymbol{J}^B_B$, and the mass and mass moment of inertia of only the momentum wheel are denoted m_W and $\boldsymbol{J}^B_W$. We assume that the momentum wheel rotates about the vehicle-fixed thrust axis $\boldsymbol{z}_B$, and is symmetric about this axis of rotation, so that $\boldsymbol{J}^B_W$ is constant when expressed in the vehicle-fixed frame. The attitude dynamics may then be derived using Euler's law for clustered bodies with a fixed center of mass [15].

The vehicle rotates with respect to the earth at angular velocity $\boldsymbol{\omega}^B_{BE}$, written in the vehicle-fixed frame, and the wheel rotates with respect to the body at a speed ω_W so that

$\boldsymbol{\omega}_{WB}^{B} = \boldsymbol{x}_{B}^{B}\omega_{W}$. The vehicle uses its propellers to produce an external torque $\boldsymbol{\tau}^{B}$, and an internal torque is produced by the motor driving the momentum wheel, which acts between the wheel and the body and is parallel to $\boldsymbol{z}_{B}$. Taking derivatives with respect to the vehicle-fixed frame, and manipulating, gives:

$$\begin{aligned} \boldsymbol{J}_{B}^{B}\dot{\boldsymbol{\omega}}_{BE}^{B} + \boldsymbol{J}_{W}^{B}\dot{\boldsymbol{\omega}}_{WB}^{B} = \\ -\boldsymbol{S}\left(\boldsymbol{\omega}_{BE}^{B}\right)\left(\boldsymbol{J}_{B}^{B}\boldsymbol{\omega}_{BE}^{B} + \boldsymbol{J}_{W}^{B}\boldsymbol{\omega}_{WB}^{B}\right) + \boldsymbol{\tau}^{B} \end{aligned} \tag{3.2}$$

Note that this neglects any effect due to the angular momentum of the propellers: Typical multicopters have an even number of propellers, identical up to a mirror symmetry, that rotate with alternating handedness, thus having near zero net angular momentum on average and having a negligible effect on the system dynamics. Moreover, the mass moment of inertia of the propellers is likely to be negligible compared to that of the body and momentum wheel. An example application where the propellers' angular momentum was considered significant is given in [31].

Compared to a traditional multicopter, the translational dynamics (3.1) are unchanged by the addition of the momentum wheel. However, the attitude dynamics include two additional terms: the coupling effect of the momentum wheel's angular momentum $\boldsymbol{S}\left(\boldsymbol{\omega}_{BE}^{B}\right)\boldsymbol{J}_{W}^{B}\boldsymbol{\omega}_{WB}^{B}$, and the acceleration/deceleration of the momentum wheel relative to the vehicle-fixed frame $\boldsymbol{J}_{W}^{B}\dot{\boldsymbol{\omega}}_{WB}^{B}$. Notably, the wheel coupling term is linear in the vehicle's angular velocity $\boldsymbol{\omega}_{BE}^{B}$, so that it has a much larger effect near hover than the other cross-coupling term $\boldsymbol{S}\left(\boldsymbol{\omega}_{BE}^{B}\right)\boldsymbol{J}_{B}^{B}\boldsymbol{\omega}_{BE}^{B}$ (which is quadratic with respect to $\boldsymbol{\omega}_{BE}^{B}$). The acceleration term $\dot{\boldsymbol{\omega}}_{WB}^{B}$ serves as additional control input to the system, and is actuated by the torque τ_{W} produced by the motor driving the momentum wheel. Note that because we neglect aerodynamic effects, the location of the momentum wheel in the $\boldsymbol{z}_{B}$ direction affects only the location of the center of mass and the total mass moment of inertia of the vehicle, and thus does not appear explicitly in the vehicle dynamics. The location of the center of mass relative to large aerodynamic surfaces may also affect a vehicle's stability, as is discussed in, e.g., [32].

Linearized dynamics

Here we present the linearized system dynamics as background for the system analysis in the following section. When linearizing the system we assume that $\boldsymbol{x}_{B}$, $\boldsymbol{y}_{B}$, and $\boldsymbol{z}_{B}$ are the principal axes of inertia of both the vehicle and the momentum wheel. For simplicity of expression, we make the assumption that the vehicle has a 90° symmetry about its thrust axis, so that the moments of inertia about $\boldsymbol{x}_{B}$ and $\boldsymbol{y}_{B}$ are identical, and that the momentum wheel is symmetric about its axis of rotation. The principal mass moments of inertia about $\boldsymbol{x}_{B}$ and $\boldsymbol{z}_{B}$ are then denoted $J_{B,xx}^{B}$ and $J_{B,zz}^{B}$ respectively for the vehicle and $J_{W,xx}^{B}$, and $J_{W,zz}^{B}$ for the momentum wheel.

The position of the vehicle relative to a fixed point in the inertial frame is written as $\boldsymbol{d}_{BE}^{E} = (x, y, z)$, and the velocity is written as $\dot{\boldsymbol{d}}_{BE}^{E} = (\dot{x}, \dot{y}, \dot{z})$. The attitude of the vehicle relative to the inertial frame is written as roll, pitch, and yaw (notated ϕ, θ, ψ), and the

angular velocity of the vehicle is written as $\boldsymbol{\omega}_{BE}^{B} = (p, q, r)$ where p, q and r are the body rates of the vehicle about the $\boldsymbol{x}_B$, $\boldsymbol{y}_B$, and $\boldsymbol{z}_B$ axes respectively. The dynamics are linearized about a desired angular velocity of the momentum wheel $\bar{\omega}_W$ such that $\Delta\omega_W = \omega_W - \bar{\omega}_W$ is the deviation of the angular velocity of the rotating body from the desired value.

The control inputs to the system are the deviation from the total thrust required to hover $\Delta f_\Sigma = f_\Sigma - m_B||\boldsymbol{g}||$, torques produced by the propellers $\boldsymbol{\tau}^B = (\tau_x, \tau_y, \tau_z)$, and the internal torque produced by the motor driving the momentum wheel τ_W. The linearization of (3.1) and (3.2) yields three decoupled subsystems with the following state and input vectors, where $\dot{s}_{xy} = A_{xy}s_{xy} + B_{xy}u_{xy}$, $\dot{s}_z = A_z s_z + B_z u_z$, and $\dot{s}_\psi = A_\psi s_\psi + B_\psi u_\psi$, and

$$s_{xy} = (x, y, \dot{x}, \dot{y}, \phi, \theta, p, q) \qquad u_{xy} = (\tau_x, \tau_y) \tag{3.3}$$

$$s_z = (z, \dot{z}) \qquad u_z = \Delta f_\Sigma \tag{3.4}$$

$$s_\psi = (\psi, r, \Delta\omega_W) \qquad u_\psi = (\tau_z, \tau_W) \tag{3.5}$$

The system matrices for these three linear subsystems are then

$$A_{xy} = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & A_1 & 0 \\ 0 & 0 & 0 & I \\ 0 & 0 & 0 & A_2 \end{bmatrix} \qquad B_{xy} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \left(J_{B,xx}^B\right)^{-1} I_{2\times 2} \end{bmatrix} \tag{3.6}$$

$$A_z = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad B_z = \begin{bmatrix} 0 \\ m_B^{-1} \end{bmatrix} \tag{3.7}$$

$$A_\psi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B_\psi = \begin{bmatrix} 0 & 0 \\ \left(J_{B,zz}^B\right)^{-1} & -\left(J_{W,zz}^B\right)^{-1} \\ 0 & \left(J_{W,zz}^B\right)^{-1} \end{bmatrix} \tag{3.8}$$

where

$$A_1 = ||\boldsymbol{g}|| \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad A_2 = \frac{J_{W,zz}^B}{J_{B,xx}^B}\bar{\omega}_W \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{3.9}$$

The linearized dynamics of the proposed system differ from the linearized dynamics of a normal multicopter due to the cross-coupling effect of matrix A_2, the additional state related to the wheel speed $\Delta\omega_W$, and the additional control input τ_W. When the nominal angular velocity of the momentum wheel $\bar{\omega}_W$ is zero, A_{xy} further decouples into a subsystem containing states $(x, \dot{x}, \theta, q)$ and a subsystem containing $(y, \dot{y}, \phi, p)$. However, as $\bar{\omega}_W$ increases in magnitude, the roll and pitch states becoming increasingly coupled, introducing an oscillatory mode into the dynamics that results in gyroscopic precession of the vehicle when external roll and pitch torques are applied. We show in the following section how this coupling improves the torque disturbance rejection capabilities of the vehicle as $\bar{\omega}_W$ increases when the additional angular momentum of the wheel is taken into account in the controller of the vehicle.

The additional control input τ_W is significant because it improves the yaw control authority of the vehicle. However its use causes deviation from the desired wheel speed, creating a trade-off between the improvement of the yaw control of the vehicle and the change in the magnitude of the coupling terms in A_2.

3.3 System Analysis and Design

In this section we first analyze how the sensitivity of the vehicle's horizontal motion to disturbances changes with respect to the scale of the vehicle and speed of the wheel. We also examine the trade-off in the wheel design, comparing the power required to carry the wheel, the rotational energy that must be stored in the wheel, and the size of the wheel.

Scaling analysis

We compare the sensitivity of a system to external disturbances as the system size is varied, and reason about how all parameters scale as a function of the vehicle's size. Through this, we argue that smaller vehicles are likely to see a lower sensitivity to torque disturbances through the addition of a momentum wheel.

Noting that the system dynamics, to first order, decouple into horizontal, vertical, and yaw subsystems, we restrict our analysis to the horizontal subsystem where the momentum wheel affects the system dynamics. The scale of the vehicle is captured by a linear scaling factor λ, so that all lengths of the vehicle scale proportionally to λ.

Scaling of dynamics

Considering the linearized horizontal dynamics of (3.6), we note that the dynamics matrix A_{xy} is a function of the vehicle parameters only through the term $\frac{J^B_{W,zz}}{J^B_{B,xx}}\bar{\omega}_W$, while the input matrix B_{xy} contains only the inertia $\left(J^B_{B,xx}\right)^{-1}$.

Assuming materials of constant density are used, the mass of the vehicle will scale proportional to its volume, or $\sim \lambda^3$, and the mass moment of inertia of the vehicle as λ^5 (being composed of mass multiplied by distance squared). We assume that the added momentum wheel scales in the same manner as the rest of the vehicle's mass moment of inertia, so that the ratio $J^B_{W,zz}/J^B_{B,xx}$ is independent of λ.

The final parameter in the dynamics is the speed of the wheel, $\bar{\omega}_W$, which is here assumed to scale proportionally to the speed of the propellers. This assumption is motivated by the fact that the momentum wheel is likely to be powered by a motor similar to that powering the propellers, and that the energy stored in the momentum wheel during operation then scales proportionally to the energy stored in the propellers (so that the momentum wheel never represents a disproportionate amount of energy in the system). Inspired by [33] we apply Mach scaling to the propellers, assuming that the propeller tip speed remains constant at different scales, so that $\bar{\omega}_W \sim \lambda^{-1}$.

Scaling of controller parameters

We reason about the system's performance when rejecting disturbances by computing the system's closed-loop sensitivity to both torque and force disturbances when applying the $\mathcal{H}_2$ optimal controller [34]. The controller is parametrized through the costs applied to the error signal $z_{xy} \in \mathbb{R}^4$ defined as:

$$z_{xy} = c(\lambda) \begin{bmatrix} I_{2\times 2} & 0_{2\times 6} \\ 0_{2\times 2} & 0_{2\times 6} \end{bmatrix} s_{xy} + d(\lambda) \begin{bmatrix} 0_{2\times 2} \\ I_{2\times 2} \end{bmatrix} u_{xy} \tag{3.10}$$

where $c(\lambda)$ is the cost of position errors, and $d(\lambda)$ is the cost of applying inputs. Under the assumption that position errors are best measured in body-lengths, the cost factor will scale as $c(\lambda) \sim \lambda^{-1}$.

The cost applied to the input is assumed to scale inversely proportionally to the maximum torque that the vehicle can apply, which will scale as the maximum force multiplied by the linear scale. We assume that the maximum force scales as the vehicle's mass, so that the maximum torque scales as λ^4 and thus the cost as $d(\lambda) \sim \lambda^{-4}$.

Sensitivity to scaling

Under the preceding assumptions, the effect of adding a momentum wheel for disturbance rejection can be compared for vehicles of different sizes. We consider the effect of force as well as torque disturbances on the vehicle, but consider these effects separately: The relative magnitude of these disturbances will depend on the nature of the disturbances, the vehicle geometry, and other factors that are difficult to capture in a straight-forward scaling law. Torque disturbances enter as added to the torque inputs, and force disturbances act directly on the vehicle velocity state. As we will normalize performance at each scale, the dependence of these disturbances on vehicle scale is immaterial.

Specifically, we consider a vehicle of nominal parameters corresponding to the larger quadcopter shown in Figure 3.1, where for $\lambda = 1$ we have $m_B = 922\,\mathrm{g}$, $J^B_{W,zz}/J^B_{B,xx} = 0.11$, $\bar{\omega}_W(\lambda) = 468\lambda^{-1}\mathrm{rad\,s^{-1}}$, and we use the costs $c(\lambda) = \lambda^{-1}\mathrm{m^{-1}}$, and $d(\lambda) = \lambda^{-4}\mathrm{N^{-1}\,m^{-1}}$. Though these costs are chosen to illustrate the effect of scaling, the resulting feedback gain matrix is similar to that which is used in the experimental evaluation of Section 6.5, thus yielding meaningful insights.

The sensitivity to torque and force disturbances as a function of the momentum wheel speed at three different vehicle scales is shown in Figure 3.3. The figure compares the nominal vehicle to vehicles that are of the scale $\lambda \in \left\{\frac{1}{2}, 1, 2\right\}$, i.e. with masses ranging from 115g to 7.4kg. The sensitivity to disturbances is defined as the normalized state feedback $\mathcal{H}_2$ norm of the system, which can be interpreted as the signal energy of z_{xy} (defined in (3.10)) after Dirac impulse disturbances occur, written as

$$||z_{xy}||_2 = \left(\int_0^\infty z_{xy}(t)^T z_{xy}(t) dt \right)^{1/2} \tag{3.11}$$

Notable is that, for all scales, the vehicle's sensitivity to torque decreases monotonically with increasing momentum wheel speed. The sensitivity to force disturbances initially decreases weakly with increasing angular momentum, before increasing, so that vehicles with large added angular momentum may be more sensitive to force disturbances. Thus, a substantial decrease in sensitivity to torque disturbances may be achieved without a great change to sensitivity to force disturbances, and the exact trade-off will depend on the relative magnitudes of force disturbances and torque disturbances.

The smaller scale vehicle, moreover, is capable of storing more angular momentum in the wheel relative to the vehicle inertia, and thus has a lower overall sensitivity to torque disturbances at the expense of a higher sensitivity to force disturbances when compared to larger scale vehicles. Note, however, that such a comparison between vehicle scales relies on strong assumptions about how the dynamics and control parameters scale as described in the previous subsections.

An optimal choice for the trade-off between sensitivity to force and torque disturbances will require additional information about the nature of the expected disturbances, which are likely application-specific. Any practical design must also weigh the potential increase in system robustness to the additional cost of carrying the added mass of the momentum wheel. This is touched upon next.

Momentum wheel design

For a fixed sized vehicle, we now investigate the design of the momentum wheel itself. Specifically, a designer must choose a wheel size, mass, and angular velocity; these will be shown to relate to the vehicle's efficiency, safety, and disturbance sensitivity.

The benefit to the dynamics follows from the angular momentum of the wheel, $J^B_{W,zz}\omega_W$, so that the effect is increased with increasing momentum. Increasing the wheel mass increases its mass moment of inertia, but this requires additional power to be carried. The increase in power consumption at hover due to the added weight can be estimated through momentum theory, which holds that the total mechanical power produced by the propellers P_Σ is related to the mass of the vehicle as $P_\Sigma = \mu m_B^{3/2}$, where μ is an experimentally measured constant that depends on the propeller geometry [35]. The maximum moment of inertia for a given mass is provided by a thin ring, whose outer radius will typically be constrained by mechanical considerations of the vehicle (e.g. so that the wheel does not protrude beyond the vehicle arms). For maximum efficiency, thus, a wheel of low mass but large radius is desired.

The momentum may also be increased by increasing the wheel's speed, but the added kinetic energy may represent a substantial safety concern e.g. in the event of a crash. For a fixed angular momentum and wheel size, the kinetic energy stored in the rotating wheel $\frac{1}{2}J^B_{W,zz}\omega^2_W$ increases inversely proportionally to the mass of the wheel. If, instead, the wheel mass and angular momentum are fixed, but its radius and speed are allowed to vary, the energy stored scales inversely proportionally to the radius squared. A practical design must

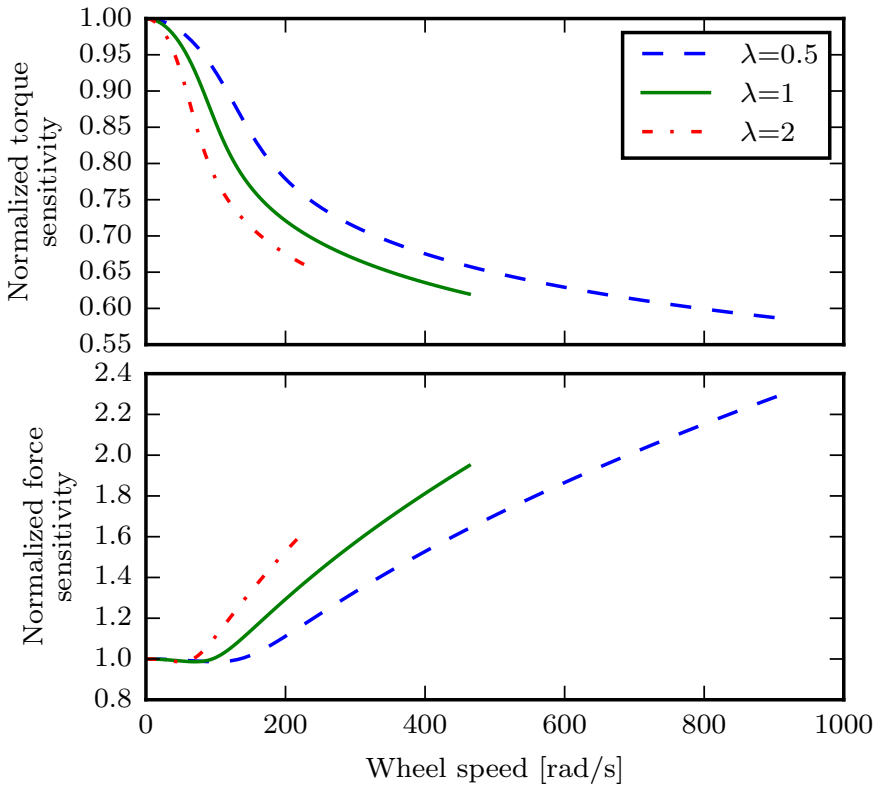

Figure 3.3: Normalized ability to reject disturbances at three different vehicle scales λ. Sensitivities are computed under the assumptions of Sect. 3.3. As the scale of the vehicle decreases, the normalized torque disturbance sensitivity of the vehicle decreases and the normalized force sensitivity of the vehicle increases.

trade off the additional power required to lift the wheel, the energy stored in the wheel, the radius of the wheel, and the vehicle's disturbance sensitivity.

3.4 Control

A controller similar to the cascaded controller used for a conventional quadcopter as described in Section 2.4 is used to control the vehicle. Unlike a controller that relies on linearizing the full dynamics of the system at hover, the proposed cascaded controller is straightforwardly able to cope with large changes of the vehicle's attitude. Although the proposed cascaded controller has a slightly larger $\mathcal{H}_2$ cost than a naïve application of the full-state linearized controller, it is shown to have much improved robustness to error in the estimated momentum wheel speed.

For the controller, an outer (position) controller computes a desired total thrust f_Σ and

thrust direction to track position reference commands, and an inner (attitude) controller tracks this desired thrust direction and a yaw command by commanding desired torques $\boldsymbol{\tau}^B$ and a desired internal torque on the wheel τ_W. The total force and torques are then converted to individual motor speed commands. Note that because the linear dynamics of the proposed vehicle are identical to those of a conventional quadcopter, the same position controller is used as was presented in Section **??**.

We proceed by defining a state feedback $\mathcal{H}_2$ optimal attitude controllers based on the linearized dynamics derived in Section 3.2. An analytic expression for the state feedback $\mathcal{H}_2$ optimal attitude controller based on the estimated speed of the momentum wheel is given, allowing for the $\mathcal{H}_2$ optimal attitude controller to be computed on-the-fly at low computational cost during quasi-static wheel speed changes.

Attitude control

For the inner (attitude) controller, we propose to apply a nonlinear controller based on [36]. The controller gains are chosen by the desired first-order behavior, described here in terms of the Euler angles that define the rotation from the desired attitude to the current attitude $(\phi_e, \theta_e, \psi_e)$. The desired attitude is defined as that attitude at which the yaw angle of the vehicle matches the desired yaw angle and at which the thrust direction of the vehicle matches the desired thrust direction $\boldsymbol{z}_{B_d}^E$. The angular velocity error $\boldsymbol{\omega}_e^B = (p_e, q_e, r_e)$ is defined as the difference between the desired and true angular velocity of the vehicle. Recall that $\Delta\omega_W = \omega_W - \bar{\omega}_W$ represents the difference between the true and desired angular velocity of the momentum wheel.

As derived in Section 3.2, the rotational dynamics decouple into two independent subsystems: one related to the roll and pitch of the vehicle, and another related to the yaw and angular velocity of the momentum wheel. The states relating to the roll and pitch of the vehicle are a subset of the states s_{xy} as given in (3.3) and (3.6), and form the rotational subsystem $\dot{s}_{\phi,\theta} = A_{\phi,\theta} s_{\phi,\theta} + B_{\phi,\theta} u_{\phi,\theta}$. The states are $s_{\phi,\theta} = (\phi_e, \theta_e, p_e, q_e)$, inputs $u_{\phi,\theta} = (\tau_x, \tau_y)$, and system matrices defined as follows, where A_2 is given in (3.9):

$$A_{\phi,\theta} = \begin{bmatrix} 0 & I \\ 0 & A_2 \end{bmatrix}, \qquad B_{\phi,\theta} = \begin{bmatrix} 0 \\ \left(J_{B,xx}^B\right)^{-1} I_{2\times 2} \end{bmatrix} \tag{3.12}$$

Due to vehicle symmetry, the state costs are chosen such that roll and pitch are penalized equally, as are the input torques about $\boldsymbol{x}_B$ and $\boldsymbol{y}_B$. We choose not to explicitly penalize the angular velocity, and thus have the output error $z_{\phi,\theta} = C_{\phi,\theta} s_{\phi,\theta} + D_{\phi,\theta} u_{\phi,\theta}$, where

$$C_{\phi,\theta} = c_{\phi,\theta}\,\mathrm{diag}\,(1, 1, 0, 0) \tag{3.13}$$

$$D_{\phi,\theta} = d_{\phi,\theta}\mathrm{diag}\,(0, 0, 1, 1) \tag{3.14}$$

The state feedback $\mathcal{H}_2$ optimal controller $u_{\phi,\theta} = -K_{\phi,\theta} s_{\phi,\theta}$ is defined by $K_{\phi,\theta} = (D_{\phi,\theta}^T D_{\phi,\theta})^{-1} B_{\phi,\theta}^T P$, where P is the solution to the relevant continuous time algebraic Riccati

equation [16]. The solution for $K_{\phi,\theta}$ can be tediously computed by solving the Riccati equation symbolically in terms of the system parameters (e.g. the mass moments of inertia of the system) and the cost weights. Solving for $K_{\phi,\theta}$ in this way allows for the state feedback $\mathcal{H}_2$ optimal attitude controller to be computed on-the-fly for any given momentum wheel speed, system parameters, and state and input costs. Note that this is applicable only if the wheel speed changes quasi-statically: otherwise, a time-varying controller may be required at the expense of increased computational cost.

$$K_{\phi,\theta} = \begin{bmatrix} \alpha & \beta & \gamma & 0 \\ -\beta & \alpha & 0 & \gamma \end{bmatrix} \tag{3.15}$$

where

$$L = J^B_{W,zz}\bar{\omega}_W \ , \quad H = \frac{c_{\phi,\theta}}{d_{\phi,\theta}}\sqrt{16\left(J^B_{B,xx}\right)^2 + L^4} - L^2 \tag{3.16}$$

$$\alpha = \frac{H}{4J^B_{B,xx}} \ , \quad \beta = \frac{L\sqrt{2H}}{4J^B_{B,xx}} \ , \quad \gamma = \frac{\sqrt{2H}}{2} \tag{3.17}$$

The controller for the yaw subsystem s_ψ given by (3.5) and (3.8) is computed using on the state and input cost matrices C_ψ and D_ψ. As this subsystem is not affected by the wheel speed $\bar{\omega}_W$, the gain matrix K_ψ may be computed offline.

After the desired total thrust f_Σ and input torques $\boldsymbol{\tau}^B = (\tau_x, \tau_y, \tau_z)$ have been computed, the propeller thrusts (and thus speeds) required to achieve these forces and torques are computed. This mapping is dependent on the structure of the multicopter and will change depending on the number and position of propellers. For example, the desired individual propeller thrusts for a quadcopter are computed by inverting (2.6).

Sensitivity to estimated wheel speed error

Errors between the estimated wheel speed and true wheel speed can result in a greater sensitivity of the system to disturbances or even outright instability. Figure 3.4 shows how the torque disturbance sensitivity of the system defined by (3.3) and (3.6) changes as a function of the difference between the estimated wheel speed and a true wheel speed of $\omega_W = 468\,\mathrm{rad\,s^{-1}}$ using the parameters of the larger vehicle shown in Figure 3.1. The $\mathcal{H}_2$ cost is normalized by the sensitivity of the system with $\omega_W = 0$, and is evaluated for both the $\mathcal{H}_2$ optimal state feedback controller considered in Section 3.3 and the cascaded controller presented in this section.

The naïve full-state $\mathcal{H}_2$ optimal controller results, of course, in the lowest system sensitivity to torque disturbances. However, this controller shows extreme sensitivity to errors in the belief of the estimated wheel speed, and the closed-loop system becomes unstable even for a very small over-estimation of the wheel speed. The cascaded controller, on the other hand, does not show this sensitivity to estimation errors, but has a greater closed-loop sensitivity to the disturbances, although it should be noted that this is not necessarily due

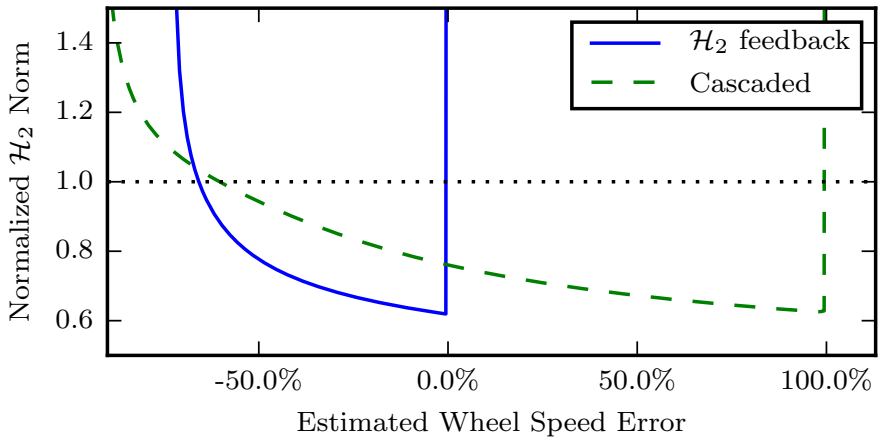

Figure 3.4: Sensitivity of the vehicle to roll/pitch torques as a function of the error of the estimated wheel speed for the large quadcopter with a true wheel speed of $\omega_W = 468\,\mathrm{rad\,s^{-1}}$. The $\mathcal{H}_2$ cost is normalized such that any value less than 1 indicates the vehicle is less sensitive to roll/pitch torque disturbances than a vehicle with $\omega_W = 0$.

to the cascaded nature of the controller. In fact, the sensitivity of the naïve full-state controller to estimation errors can be changed by modifying the costs associated with each state (including adding cost to the angular velocity states), but it is unclear how these costs would be chosen such that acceptable sensitivities to both estimation errors and disturbances are simultaneously achieved. Furthermore, although the cascaded controller presented in this section is not the optimal controller in terms of improving the disturbance sensitivity of the system, it is still useful due to relative insensitivity to errors in the estimated wheel speed, ease of implementation, and its ability to straight-forwardly cope with large attitude errors.

3.5 Experimental Validation

In this section we present experimental results to validate the models using two vehicles of different scales subjected to torque impulse disturbances. An additional test case is presented where the torque impulse disturbance is large enough to cause saturation of the motor forces. The improvement of the yaw control authority of the vehicle is verified by commanding a step change in yaw.

Platform

Two custom quadcopters of different sizes were constructed for testing as shown in Figure 3.1, with masses differing by a factor of 18. The smaller vehicle uses CL-0720-14 brushed motors while the larger vehicle uses EMAX MT2208 brushless motors with DYS SN30A electronic

Table 3.1: Physical parameters of experimental vehicles

Parameter	Large Vehicle	Small Vehicle
m_B	$922\,\mathrm{g}$	$50\,\mathrm{g}$
$J^B_{B,xx}$	$5.8 \times 10^{-3}\,\mathrm{kg\,m^2}$	$3.5 \times 10^{-5}\,\mathrm{kg\,m^2}$
$J^B_{B,zz}$	$10.7 \times 10^{-3}\,\mathrm{kg\,m^2}$	$6.0 \times 10^{-5}\,\mathrm{kg\,m^2}$
$J^B_{W,zz}$	$6.4 \times 10^{-4}\,\mathrm{kg\,m^2}$	$1.7 \times 10^{-6}\,\mathrm{kg\,m^2}$
$f_{\max}$	$6.86\,\mathrm{N}$	$0.24\,\mathrm{N}$
l	$0.235\,\mathrm{m}$	$0.068\,\mathrm{m}$
κ^+	$0.014\,\mathrm{m}$	$0.001\,\mathrm{m}$

speed controllers that control the rotational velocity of the motors in closed-loop. The same motors and speed controllers are used to spin the momentum wheel on each vehicle. The physical parameters of each vehicle are listed in Table 3.1.

We compare the responses of each vehicle both with and without their momentum wheels spinning. For the large vehicle, the nominal wheel speed is set to $\bar{\omega}_W = 468\,\mathrm{rad\,s^{-1}}$, determined based upon an energy argument: At this speed, the rotational energy stored in the momentum wheel is twice the maximum rotational energy stored in the propellers of the vehicle, meaning that the addition of the momentum wheel does not radically change the danger posed by the vehicle's rotating parts. For the smaller vehicle, the nominal wheel speed is set to $\bar{\omega}_W = 1000\,\mathrm{rad\,s^{-1}}$. At this scale, the rotational energy is not a concern, and this speed was chosen to be slightly below the maximum that can be achieved by the driving motor.

For the shown experiments, the position and attitude of the quadcopter are measured directly by an external motion capture system, and the angular velocity of the quadcopter is measured using an onboard rate gyroscope. The position controller runs on an offboard computer and sends commands and attitude measurements to the quadcopter via radio at 50Hz. The attitude controller is ran onboard the quadcopter at 500Hz. The use of other sensing technologies (e.g. GPS) are expected to yield similar results, while even more pronounced improvements may result if onboard vision is used, where estimation performance is degraded through motion blur.

Control

The $\mathcal{H}_2$ error weights for the cascaded controller of Section 3.4 were chosen using Bryson's rule [37], which is a heuristic based on the maximum acceptable values of the states and inputs, so that the output errors are normalized to their maximum acceptable values. For example, for a maximum roll error $\phi_{e,max}$, a cost weighting $c_{\phi,\theta} = \phi^{-1}_{e,max}$ is used. The cost weighting are then used to compose state the cost matrices C_p, $C_{\phi,\theta}$, C_ψ and input cost matrices D_p, $D_{\phi,\theta}$, D_ψ, which are used to compute corresponding the state feedback $\mathcal{H}_2$ optimal controllers as described in Section 3.4. We use d_{max} as the maximum acceptable position error, a_{max} is the maximum acceptable acceleration, and again weight the two

Table 3.2: Parameters used to compute control cost matrices

Parameter	Value
$\lvert d_{max}\rvert$	$2.5\,\mathrm{m}$
$\lvert a_{max}\rvert$	$10\,\mathrm{m\,s^{-2}}$
$\lvert\phi_{e,max}\rvert, \lvert\theta_{e,max}\rvert, \lvert\psi_{e,max}\rvert$	30°
$\lvert\Delta\omega_{W,max}\rvert$	$50\,\mathrm{rad\,s^{-1}}$
$\lvert\tau_{x,max}\rvert, \lvert\tau_{y,max}\rvert$	$f_{\max} l/2$
$\lvert\tau_{z,max}\rvert$	$\kappa^+ f_{\max}$
$\lvert\tau_{W,max}\rvert$	$5\lvert\tau_{z,max}\rvert$

horizontal axes similarly due to the symmetry of the vehicle. The maximum acceptable values used to compute each cost matrix are given in Table 3.2.

In addition to the cascaded feedback controller, a small feedforward term is added to τ_z to compensate for the drag torque exerted by the momentum wheel about its axis of rotation $\boldsymbol{z}_B$ as it spins. This drag torque is not included in the system model (3.8), and thus must be determined experimentally.

Power Consumption

Using the relationship described in Section 3.3 to calculate the mechanical power required to lift a given mass, the larger vehicle requires 58 W of mechanical power to hover when the momentum wheel is not attached. However, when the momentum wheel is attached, an additional 100 g is added to the mass of the vehicle, requiring an additional 11 W of mechanical power to hover. Furthermore, 4.8 W of mechanical power is required to spin the wheel at the desired speed of $468\,\mathrm{rad\,s^{-1}}$, resulting in a 27% increase in power consumption when the wheel is attached and spinning compared to a vehicle without the wheel attached. The mechanical power required to spin the wheel is composed of the power consumed by the motor spinning the wheel (used to overcome aerodynamic drag acting on the spinning wheel), and the additional power consumed by the propellers (used to compensating for the torque produced by the wheel motor). The electrical power consumed will of course be larger, due to losses in the power train, but the relative increase should be approximately the same.

The increase in power consumption due to the momentum wheel is similar for the smaller vehicle. Without the wheel the vehicle produces 0.54 W of mechanical power during hover, and the attached 3.9 g momentum wheel requires and additional 0.07 W to lift and 0.05 W to spin at $1000\,\mathrm{rad\,s^{-1}}$, corresponding to an increase in power consumption of 23%. Optimized designs may be expected to perform substantially better than this.

Figure 3.5: Test vehicles with attached arm extensions. A falling mass collides with the arm extension in order to provide an torque impulse disturbance to the vehicle.

Impulse torque disturbance rejection

The torque disturbance rejection capabilities of each vehicle are tested by applying a repeatable torque impulse to the vehicle by dropping a mass on the vehicle from above. Arm extensions are added to each vehicle as shown in Figure 3.5 so that the falling mass does not collide with the propellers of the vehicle. For the larger vehicle, masses of 67 g and 135 g were dropped from a height of 1 m to apply torque impulses of $0.092\,\mathrm{N\,m\,s}$ and $0.186\,\mathrm{N\,m\,s}$ to the vehicle respectively. For the smaller vehicle, a mass of 5 g was dropped from a height of 5 cm to apply an estimated torque impulse of $4.7 \times 10^{-4}\,\mathrm{N\,m\,s}$ to the vehicle. Figure 3.6 shows the responses of each vehicle to these torque impulse disturbances, and Figure 3.7 shows a sequence of images corresponding to the response of the larger vehicle to the $0.186\,\mathrm{N\,m\,s}$ torque impulse disturbance. [1]

In each of the experiments, the improvement due to the momentum wheel is clear: A lower peak tilt error is recorded, smaller horizontal and vertical errors occur, and the required thrust forces are lower. The improvement of the response of the vehicle with the wheel spinning over the vehicle without the wheel spinning is particularly clear for the large impulse on the larger vehicle, shown in Figure 3.6b, being large enough to cause the thrust forces to saturate only when the wheel is not spinning.

We compare the responses of the vehicles by computing the experimental state feedback $\mathcal{H}_2$ cost of the system trajectory for each test; that is we integrate (3.10) over $t \in [0, 2.5]$ using a position cost of $1\,\mathrm{m}^{-1}$ and roll/pitch torque cost of $1\,\mathrm{N}^{-1}\,\mathrm{m}^{-1}$. For the large vehicle's test cases shown in Figs. 3.6a and 3.6b, the experimental cost when using the wheel is 0.54 and 0.32 respectively when normalized to the cost when the wheel is not used. This experimentally observed normalized cost is comparable to that predicted by the analytic

[1] A video demonstrating how the experiments were performed can be viewed at: `https://youtu.be/C2fj2D_2pI8`

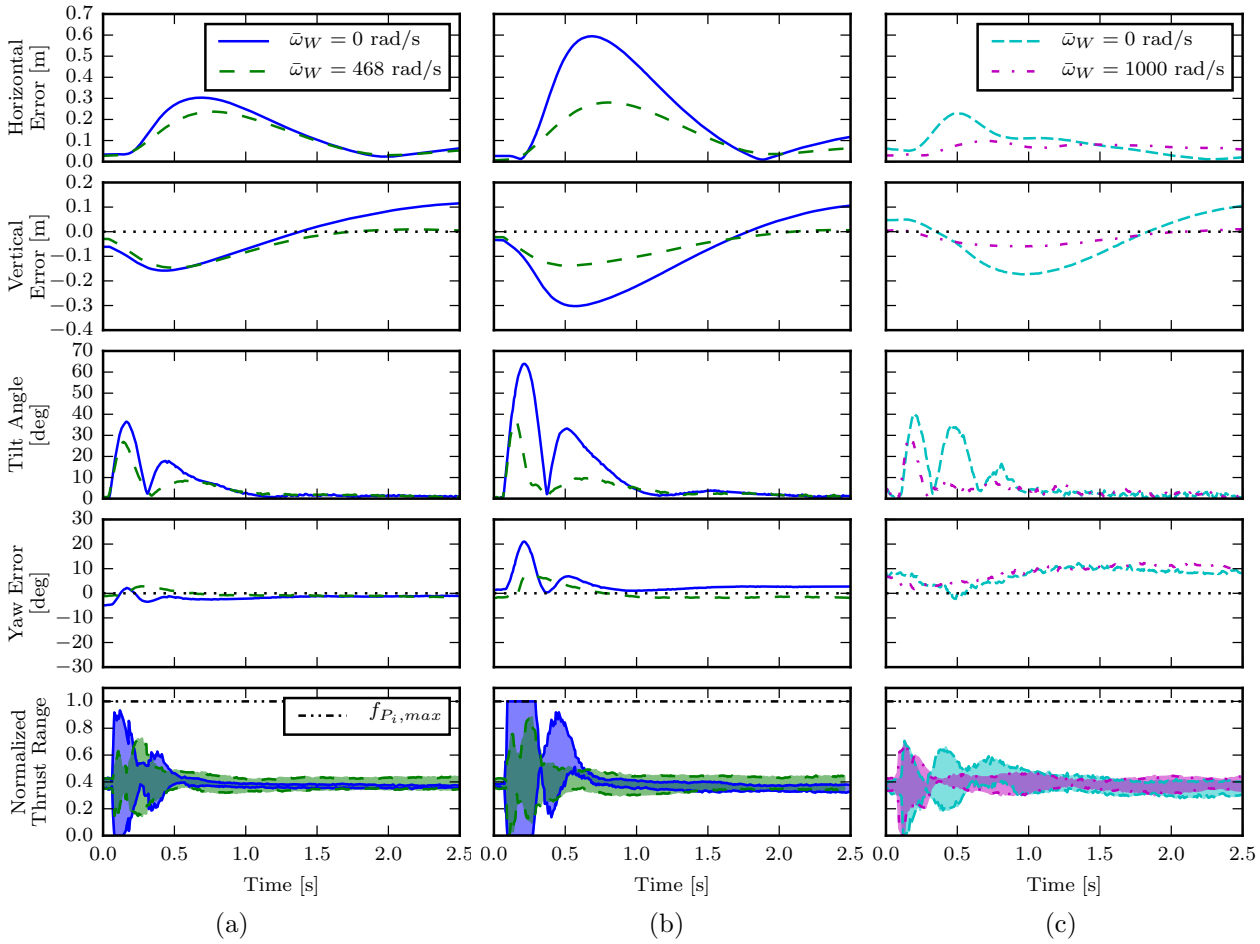

Figure 3.6: Responses to a torque impulse caused by a collision with a mass dropped from above. The tilt angle is defined as the angle between $\boldsymbol{z}_B^E$ and the vertical, and the normalized thrust range is defined as the minimum range that contains all four thrust forces, which are normalized to the maximum thrust $f_{\max}$. The response of the larger vehicle to a $0.092\,\mathrm{N\,m\,s}$ torque impulse is shown in (a), the response of the larger vehicle to a $0.186\,\mathrm{N\,m\,s}$ torque impulse is shown in (b), and the response of the smaller vehicle to a $4.7 \times 10^{-4}\,\mathrm{N\,m\,s}$ torque impulse is shown in (c). The performance of the vehicle is improved in all three cases when the wheel is spinning, and an even more significant improvement is observed when the motor forces saturate, as shown in (b).

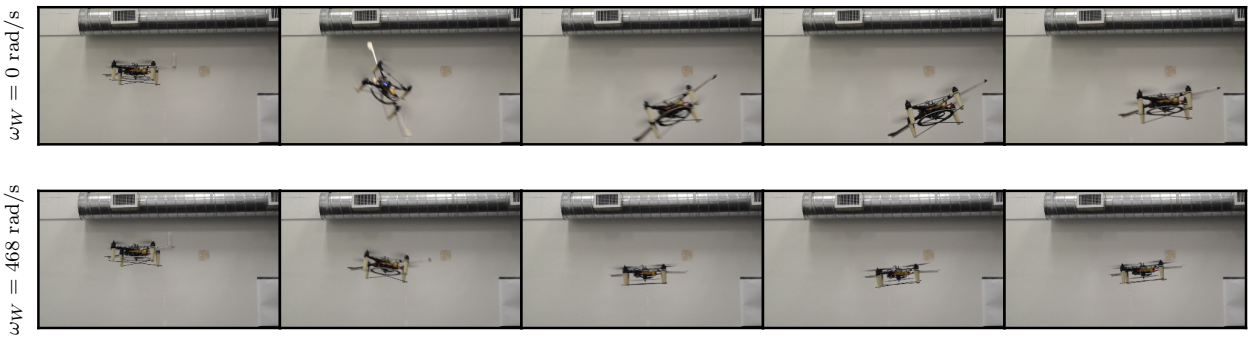

Figure 3.7: Response of the larger vehicle to a 0.186 N m s torque impulse as shown in Figure 3.6b. The top series of images shows the response without the momentum wheel spinning, and the bottom series of images shows the response when the momentum wheel is spinning at 468 rad s^{-1}. Images are spaced 0.2 seconds apart.

model of 0.62, with the discrepancy likely due to modeling errors and actuator saturation in the case of Figure 3.6b. The smaller vehicle's experimentally observed performance exceeds that predicted, most likely due to (comparatively) poor position tracking for the smaller vehicle. Moreover, the presence of sensor/environmental noise in the system (in addition to the initial impulse), means that the experimental cost is affected by the choice of integration length (with the theoretical result corresponding to a noise-free impulse response integrated over an infinite horizon). The analytic model also does not account for any aerodynamic effects that may be introduced by the spinning of the momentum wheel, and in general it is difficult to predict such effects.

Improved yaw authority

Although the magnitude of the angular momentum stored in the momentum wheel does not affect the sensitivity of the vehicle to torque disturbances about the yaw axis, the wheel can be used to improve the yaw authority of the vehicle as discussed in Section 3.4. Specifically, the motor driving the wheel provides an additional yaw torque that results in the wheel either accelerating or decelerating relative to the vehicle body. The improvement in yaw authority is experimentally validated by commanding 45° step changes in desired yaw in both the positive and negative directions while the vehicle is hovering. Results are compared to the response of the vehicle performing the same maneuver without allowing the momentum wheel to accelerate or decelerate, which is accomplished by setting $\Delta\omega_{W,max} = 0$ instead of $\Delta\omega_{W,max} = 50\,\text{rad}\,\text{s}^{-1}$ as given in Table 3.2 and recomputing the associated cost matrices C_ψ and D_ψ used to compute the gain matrix K_ψ as described in Section 3.5. A response to a

single trial for the large vehicle is shown in Figure 3.8. [2]

The magnitude of the improvement of the yaw authority is dependent on both the limits on the torque of the motor driving the momentum wheel, and on the tolerable change in the speed of the wheel. Note, however, that the range of τ_W can be asymmetric due to the actuators used to drive the wheel. For example, on the smaller vehicle the wheel is driven by a unidirectional brushed motor, meaning that the wheel can be accelerated by the motor, but must rely on friction and drag torque to decelerate the wheel.

Although the controller described in Section 3.4 uses the torque produced by the motor driving the wheel as a control input, electronic speed controllers (including those used in these experiments) commonly required speed commands. In order to compute the desired speed command $\bar{\omega}_W$ for the momentum wheel motor such that the desired torque τ_W is applied, we model the speed of the wheel ω_W as a first order system as follows, where c is the time constant of the wheel (we estimated $c \approx 1\,\mathrm{s}$ for the large vehicle).

$$\dot{\omega}_W = \frac{1}{c}(\bar{\omega}_W - \omega_W) \tag{3.18}$$

Given a desired torque, the desired speed command for the momentum wheel motor is then computed as

$$\bar{\omega}_W = \omega_W + \frac{c}{J^B_{W,zz}}\tau_W \tag{3.19}$$

Step change in position

In order to compare how additional angular momentum affects the maneuverability of the vehicle, the responses of the larger vehicle to a 1.5 m horizontal step change in desired position both with and without the momentum wheel spinning are compared in Figure 3.9. The sudden change in desired thrust direction also results in a sudden change in desired attitude of the vehicle.

Due to the angular momentum of the wheel, more time is required to change the attitude of the vehicle when the wheel is spinning. However, because the majority of the maneuver consists of translating rather than rotating, the responses of the vehicles with and without the wheel spinning are similar, where the vehicle with the wheel spinning requires slightly more time to reach the desired position.

3.6 Conclusion

In this chapter we have presented a novel design for a multicopter for use in challenging environments, exploiting the addition of a momentum wheel for increased robustness to disturbances. The vehicle dynamics were derived, of which the additional coupling between

[2] A video of the experiment can be found at `https://youtu.be/C2fj2D_2pI8`

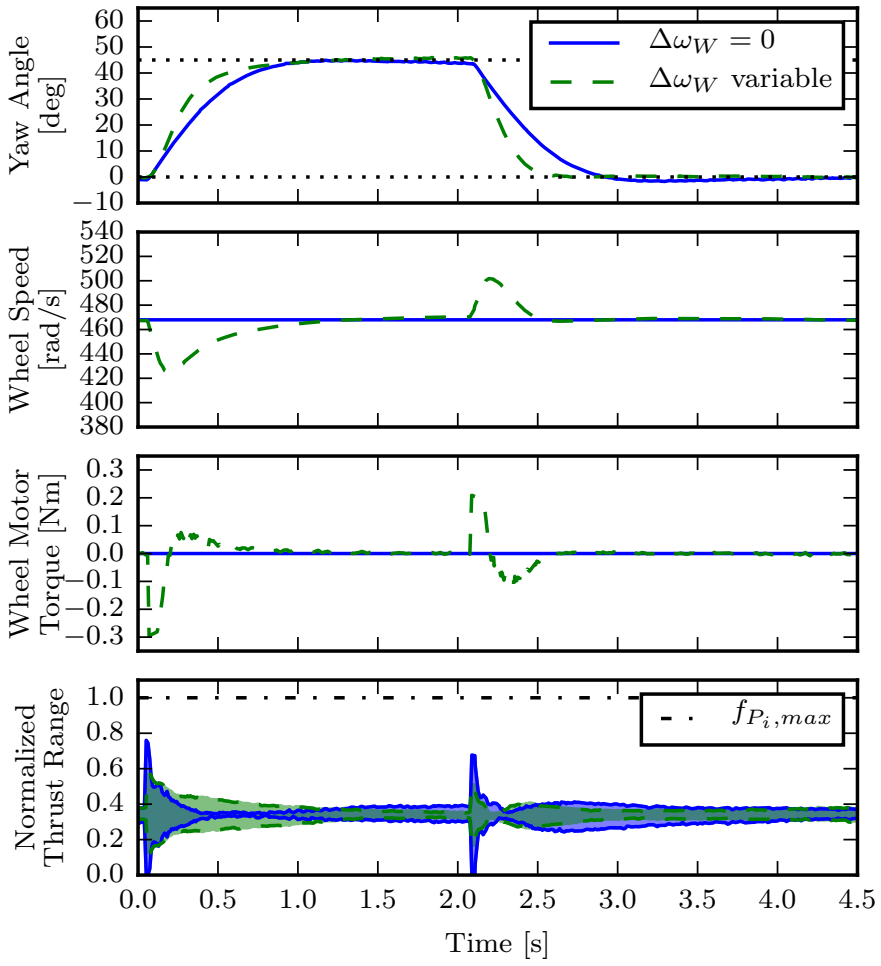

Figure 3.8: Response of the large vehicle to a step change in desired yaw of 45° followed by a second step change in desired yaw back to 0°. A controller that allows for some error in the desired momentum wheel speed outperforms a controller that does not allow the wheel to accelerate or decelerate.

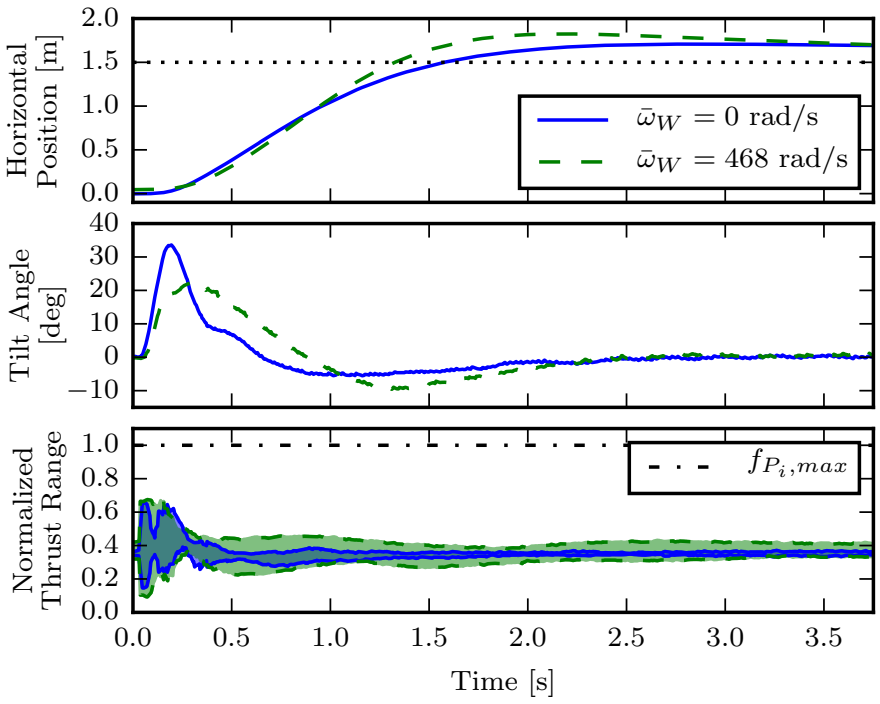

Figure 3.9: Responses of the large vehicle to a 1.5 m horizontal step change in desired position both with and without the momentum wheel spinning. The vehicle with the wheel spinning requires slightly more time to reach the desired position.

the vehicle's roll and pitch dynamics is key to the added robustness. A scaling analysis shows that greater benefit is expected for smaller vehicles. A simple cascaded control structure is proposed and implemented in experiment; the experimental results are shown to correspond closely to that predicted by the analysis of the system.

Specifically, it is shown that as the angular momentum of the wheel is increased, the vehicle's position tracking $\mathcal{H}_2$ cost monotonically decreases for torque disturbances. For force disturbances, the cost initially also decreases, but is shown to increase when the wheel has large angular momentum. As the momentum wheel speed can be varied dynamically in flight, the vehicle's flight characteristics can be adapted mid-mission, allowing e.g. for agile motion followed by steady station keeping.

This increase in robustness comes at additional energetic cost associated with increasing the vehicle mass, though the amount of added mass can be reduced at the cost of increasing the kinetic energy stored in the wheel. Furthermore, we show that the closed-loop system may be extremely sensitive to errors in belief of the wheel speed, but the proposed cascaded controller is shown to be less sensitive than a naïve full state linearized feedback controller.

Vehicles of the proposed design may be expected to be especially valuable when conducting missions in very sensitive or unpredictable environments, such as when operating

over crowds of people, or near critical infrastructure. Future implementations may consider replacing the momentum wheel with a large propeller, allowing the vehicle to increase the propellers' surface area and potentially increasing overall system efficiency; other designs may enclose the momentum wheel in a very robust cage, allowing the wheel to operate at very high velocities with low mass.

Chapter 4

Improved Operational Capabilities via Aerial Morphing

Similar to the previous chapter, this chapter presents a modified quadcopter design that enables the vehicle to perform tasks that a conventional quadcopter would be unable to perform. Specifically, the design and control of a novel quadcopter capable of changing shape mid-flight is presented, allowing for operation in four configurations with the capability of sustained hover in three. The normally rigid connections between the arms of the quadcopter and the central body are replaced by free-rotating hinges that allow the arms to fold downward; no additional actuators beyond the four motors that drive the propellers are used. Configuration transitions are accomplished by either reducing or reversing the thrust forces produced by specific propellers during flight. Constraints placed on the control inputs of the vehicle prevent the arms from folding or unfolding unexpectedly, allowing for the use of existing quadcopter controllers and trajectory generation algorithms. For our experimental vehicle at hover, we find that these constraints result in a 36% reduction of the maximum yaw torque the vehicle can produce, but do not result in a reduction of the maximum thrust or roll and pitch torques. Furthermore, the ability to change configurations is shown to enable the vehicle to traverse small passages, perch on hanging wires, and perform simple grasping tasks.

Note that the material presented in this chapter is based on the following works, with the latter paper currently under review at IEEE Transactions on Robotics (T-RO). As the latter paper is an extension of the former, this chapter is largely based upon the latter paper and only differs slightly in terms of the notation used in the paper and the discussion of the control of the vehicle, as this has already been partially described in Chapter 2.

- Nathan Bucki and Mark W Mueller. "Design and Control of a Passively Morphing Quadcopter". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9116–9122

- Nathan Bucki, Jerry Tang, and Mark W Mueller. "Design and Control of a Midair Reconfigurable Quadcopter using Unactuated Hinges". In: *arXiv preprint*

arXiv:2103.16632 (2021)

4.1 Introduction

In recent years, many extensions of the original quadcopter design have been proposed in order to allow for new tasks to be performed, improving their utility. However, this typically requires the vehicle to carry additional hardware, which not only can reduce flight time due to the increased weight of the system, but can also increase the complexity of the vehicle, making it more difficult to build and maintain, which can lead to a higher likelihood of system failures. In this chapter we present a design change to the quadcopter which allows the vehicle to change shape during flight, perch, and perform simple aerial manipulation, all without requiring significant hardware additions (e.g. motors or complex mechanisms).

Related Work

Several aerial vehicles capable of changing shape have been previously developed. For example, in [40] a vehicle capable of automatically unfolding after being launched from tube is presented, and in [41] a vehicle is presented which uses foldable origami-style arms to automatically increase its wingspan during takeoff. Although such designs excel in enabling the rapid deployment of aerial vehicles, they do not focus on repeated changes of shape, and thus require intervention to be returned to their compressed forms.

Vehicles capable of changing shape mid-flight have also been developed in order to enable the traversal of narrow passages. In [42] a vehicle that uses several servomotors to actuate a scissor-like structure that can shrink or expand the size of the vehicle is presented, and in [43] a single servomotor is used in conjunction with an origami structure to enable the arms of a quadcopter to shorten or lengthen during flight. Vehicles that use a central actuator to change the angle of their arms in an X-shape are presented in [44] and [45], and a vehicle that uses four servomotors to change each arm angle is presented in [46] and extended in [47]. In [48] and [49] a quadcopter design is presented that is capable of using one or more actuators to reposition the propellers of the vehicle to be above one another such that the horizontal dimension of the vehicle is reduced. Similarly, [50] uses a single actuator to reposition the propellers of the vehicle to be in a horizontal line, and demonstrates the vehicle being used to traverse a narrow gap.

A large body of work has also been produced regarding the use of quadcopters to perform aerial manipulation. Aerial vehicles with the capability to interact with the environment open the door to a wide range of potential applications, e.g. performing construction as shown in [51]. Typically such designs involve attaching one or more robot arms to a quadcopter, as shown in [52], [53], and [54] for example. Several designs involve changing the structure of the vehicle, such as [46] (described previously) and [55], which describes a ring-shaped multicopter-like vehicle capable of changing the relative position of each propeller such that the body of the vehicle can be used to grasp objects. Other designs, such as [56], use passive

elements to engage a gripper and a single actuator to disengage the gripper. However, such designs require the vehicle to carry one or more actuators beyond the four motors used to drive the propellers (e.g. servomotors used for opening/closing a gripper), increasing the weight of the vehicle and therefore decreasing flight time. Additional examples of vehicles used to perform aerial manipulation can be found in the aerial manipulation survey papers [57] and [58].

Finally, several designs have been proposed that enable aerial vehicles to perch on structures in the environment. Such vehicles are able to fly to a desired location, attach themselves to a feature in the environment, and then remain stationary without consuming significant amounts of energy (e.g. while monitoring the surrounding area). In [59] a passive adhesive mechanism is proposed for perching on smooth surfaces, and in [60] adhesive pads are used in conjunction with a servomotor to attach and detach the vehicle from vertical walls. In [61] and [62] grippers actuated using servomotors are used to enable perching on bars. Similarly, [63] describes a purely passive gripper that used the weight of the vehicle to close a gripper around a horizontal bar.

The work presented in this chapter additionally extends [38] in several ways, enabling the vehicle to perform several of the previously mentioned tasks while requiring only minor changes to the design and control of the vehicle compared to a conventional quadcopter.

Capabilities of the novel vehicle

In [38] a quadcopter design was presented that replaced the typically rigid connections between the arms of the quadcopter and the central body with sprung hinges that allow for the arms of the quadcopter to fold downward when low thrusts are produced by the propellers. This feature enabled the vehicle to reduce its largest dimension while in projectile motion, allowing the vehicle to fly towards a narrow gap, collapse its arms, and then unfold after traversing the gap. In this work we perform two significant design changes. First, we remove the springs used to fold the arms, and instead fold each arm by reversing the thrust direction of the attached propeller. Second, we change the geometry of the vehicle such that when two opposite arms are folded, the thrust vectors of the associated propellers are offset from one another, allowing for a yaw torque to be produced when the thrust direction of the propellers is reversed.

No actuators or complex mechanisms are added to the vehicle, keeping its mass low, and only standard off-the-shelf components (e.g. propellers, motors, and electronic speed controllers) are used in the design, with the exception of the custom 3D printed frame of the vehicle. Thus, the main difference between the vehicle described in this chapter and a conventional quadcopter is the fact that each arm is attached to the central body via a rotational joint rather than with a rigid connection.

These changes enable the vehicle to perform a number of tasks as shown in Figure 4.1, namely:

- Stable flight as a conventional quadcopter with all four arms unfolded

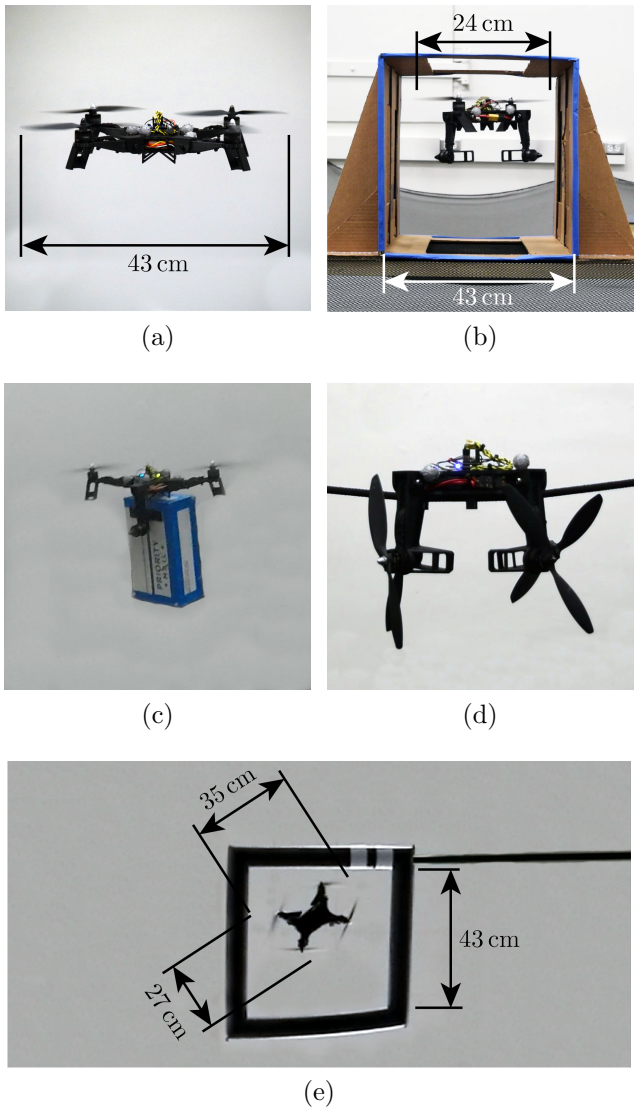

Figure 4.1: Images of the experimental vehicle performing a variety of different tasks. The vehicle is capable of flying like a conventional quadcopter when in the unfolded configuration (a), but when flying in the two-arms-folded configuration is able to, e.g., traverse narrow tunnels (b) and perform simple aerial manipulation tasks such as carrying a box (c). Additionally, by allowing all four arms to fold, the vehicle is able to perch on thin wires (d), and even traverse narrow gaps in projectile motion (e) (view from below).

- Stable flight with two arms folded, allowing for the traversal of narrow tunnels
- Grasping of and flight with objects of appropriate dimensions
- Perching on wires with all four arms folded
- Traversal of narrow gaps in projectile motion with all four arms folded

By avoiding the use of complex mechanisms or additional actuators beyond the four motors used to drive the propellers, the proposed vehicle is capable of flying in the unfolded configuration with an efficiency nearly identical to that of a similarly designed conventional quadcopter. The main drawback of our design is the fact that stricter bounds must be placed on the four thrust forces such that the vehicle remains in the desired configuration during flight. However, as we will show in Section 4.4, these bounds do not significantly reduce the agility of the vehicle except in terms of a decrease of the maximum yaw torque the vehicle can produce in the unfolded configuration.

4.2 System Model

In this section we follow [38] in defining a model of the system and deriving the dynamics of the vehicle. The dynamics of the vehicle are then used in Section 4.3 to derive bounds on the control inputs such that the vehicle remains in the desired configuration. Note that the notation and derivation of the vehicle dynamics also closely follows that of a conventional quadcopter as described in Chapter 2, but with additional consideration given to the unactuated degrees of freedom added to the vehicle.

The vehicle consists of four rigid arms connected to a central body via unactuated rotary joints (i.e. hinges) which are limited to a range of motion of 90°. Unlike [38], however, each hinge is positioned such that the vehicle is only 180° axis-symmetric rather than 90° axis-symmetric. Figure 4.2 shows a top-down view of the vehicle, including the orientation of each of the hinges and arms.

Model

The system is modeled as five coupled rigid bodies: the four arms and the central body of the vehicle. The inertial frame is notated as E, the frame fixed to the central body as C, and the frame fixed to arm $i \in \{1, 2, 3, 4\}$ as A_i. The rotation matrix of frame C with respect to frame E is defined as $\boldsymbol{R}^{CE}$ such that the quantity $\boldsymbol{v}^C$ expressed in the C frame is equal to $\boldsymbol{R}^{CE}\boldsymbol{v}^E$ where $\boldsymbol{v}^E$ is the same quantity expressed in frame E. The orientation of arm i with respect to the central body is defined through the single degree of freedom rotation matrix $\boldsymbol{R}^{A_iC}$. Furthermore, let P_i be a point along the thrust axis of propeller i, and let H_i be the point where the rotation axis of hinge i intersects with the plane swept by the thrust axis of propeller i as arm i rotates about its hinge.

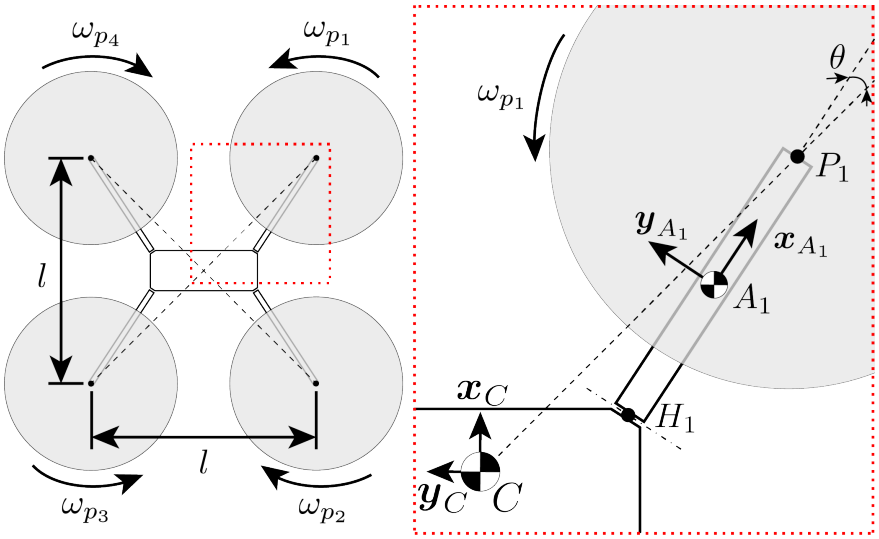

Figure 4.2: Top-down view of vehicle in the unfolded configuration (left) and arm A_1 (right). In the unfolded configuration, the thrust axis of each rotor is parallel and equidistant from its neighbor, as is typical for quadcopters. Each arm is connected to the central body by a hinge that rotates in the $\boldsymbol{y}_{A_i}$ direction, allowing the arms to independently rotate between the folded and unfolded configurations. The orientation of each hinge relative to the central body is determined by θ. Each propeller produces a thrust force f_{p_i} and torque τ_{p_i} at P_i in the direction of $\boldsymbol{z}_{A_i}$.

The internal reaction forces and torques acting at the hinge are defined as $\boldsymbol{f}_{r_i}$ and $\boldsymbol{\tau}_{r_i}$ respectively. The propeller attached to arm i produces scalar thrust force f_{p_i} and aerodynamic reaction torque τ_{p_i} in the $\boldsymbol{z}_{A_i}$ direction. We assume that the torque produced by each propeller is piecewise linearly related to the propeller thrust force as give in equation (2.1).

The mass and mass moment of inertia of the central body taken at the center of mass of the central body are denoted m_C and $\boldsymbol{J}_C$ respectively, and the mass and mass moment of inertia arm i taken at its center of mass are denoted m_{A_i} and $\boldsymbol{J}_{A_i}$ respectively.

Dynamics

The translational and rotational dynamics of the central body of the vehicle and the four arms are found using Newton's second law and Euler's law respectively [15]. We assume that the only external forces and torques acting on the vehicle are those due to gravity and the thrusts and torques produced by each propeller (for example, aerodynamic effects acting on the central body or arms are not considered). The time derivative of a vector is taken in the reference frame in which that vector is expressed.

We express the translational dynamics of the central body in the inertial frame E, and the

rotational dynamics of the central body in the body-fixed frame C. Let $\boldsymbol{g}$ be the acceleration due to gravity. The translational dynamics of the central body are then:

$$m_C \ddot{\boldsymbol{d}}_{CE}^{E} = m_C \boldsymbol{g}^{E} + \boldsymbol{R}^{EC} \sum_{i=1}^{4} \boldsymbol{f}_{r_i}^{C} \tag{4.1}$$

and the rotational dynamics of the central body are:

$$\begin{aligned} &\boldsymbol{J}_C^C \dot{\boldsymbol{\omega}}_{CE}^{C} + \boldsymbol{S}\left(\boldsymbol{\omega}_{CE}^{C}\right) \boldsymbol{J}_C^C \boldsymbol{\omega}_{CE}^{C} \\ &\quad = \sum_{i=1}^{4} \left(\boldsymbol{\tau}_{r_i}^{C} + \boldsymbol{S}\left(\boldsymbol{d}_{H_iC}^{C}\right) \boldsymbol{f}_{r_i}^{C}\right) \end{aligned} \tag{4.2}$$

We express the translational and rotational dynamics of arm i in frame A_i. The translational dynamics of arm i are (note $\boldsymbol{f}_{r_i}^{A_i} = \boldsymbol{R}^{A_iC} \boldsymbol{f}_{r_i}^{C}$):

$$m_{A_i} \left(\boldsymbol{R}^{A_iE} \ddot{\boldsymbol{d}}_{CE}^{E} + \boldsymbol{\alpha}\right) = m_{A_i} \boldsymbol{R}^{A_iE} \boldsymbol{g}^{E} + \boldsymbol{z}_{A_i}^{A_i} f_{p_i} - \boldsymbol{f}_{r_i}^{A_i} \tag{4.3}$$

where $\boldsymbol{\alpha}$ is

$$\begin{aligned} \boldsymbol{\alpha} =& \boldsymbol{R}^{A_iC} \left(\boldsymbol{S}\left(\boldsymbol{d}_{CH_i}^{C}\right) \dot{\boldsymbol{\omega}}_{CE}^{C} + \boldsymbol{S}\left(\boldsymbol{\omega}_{CE}^{C}\right) \boldsymbol{S}\left(\boldsymbol{d}_{CH_i}^{C}\right) \boldsymbol{\omega}_{CE}^{C}\right) \\ &+ \boldsymbol{S}\left(\boldsymbol{d}_{H_iA_i}^{A_i}\right) \dot{\boldsymbol{\omega}}_{A_iE}^{A_i} + \boldsymbol{S}\left(\boldsymbol{\omega}_{A_iE}^{A_i}\right) \boldsymbol{S}\left(\boldsymbol{d}_{H_iA_i}^{A_i}\right) \boldsymbol{\omega}_{A_iE}^{A_i} \end{aligned} \tag{4.4}$$

The rotational dynamics of arm i are (note $\boldsymbol{\tau}_{r_i}^{A_i} = \boldsymbol{R}^{A_iC} \boldsymbol{\tau}_{r_i}^{C}$):

$$\begin{aligned} &\boldsymbol{J}_{A_i}^{A_i} \dot{\boldsymbol{\omega}}_{A_iE}^{A_i} + \boldsymbol{S}\left(\boldsymbol{\omega}_{A_iE}^{A_i}\right) \boldsymbol{J}_{A_i}^{A_i} \boldsymbol{\omega}_{A_iE}^{A_i} = \boldsymbol{S}\left(\boldsymbol{d}_{P_iA_i}^{A_i}\right) \boldsymbol{z}_{A_i}^{A_i} f_{p_i} \\ &\quad + \boldsymbol{z}_{A_i}^{A_i} \tau_{p_i} - \boldsymbol{\tau}_{r_i}^{A_i} - \boldsymbol{S}\left(\boldsymbol{d}_{H_iA_i}^{A_i}\right) \boldsymbol{f}_{r_i}^{A_i} \end{aligned} \tag{4.5}$$

The equations of motion of the arm are written in terms of $\dot{\boldsymbol{\omega}}_{A_iE}^{A_i}$ and $\boldsymbol{\omega}_{A_iE}^{A_i}$ for convenience, which evaluate to:

$$\begin{aligned} \boldsymbol{\omega}_{A_iE}^{A_i} &= \boldsymbol{\omega}_{A_iC}^{A_i} + \boldsymbol{R}^{A_iC} \boldsymbol{\omega}_{CE}^{C} \\ \dot{\boldsymbol{\omega}}_{A_iE}^{A_i} &= \dot{\boldsymbol{\omega}}_{A_iC}^{A_i} + \boldsymbol{R}^{A_iC} \dot{\boldsymbol{\omega}}_{CE}^{C} - \boldsymbol{S}\left(\boldsymbol{\omega}_{A_iC}^{A_i}\right) \boldsymbol{R}^{A_iC} \boldsymbol{\omega}_{CE}^{C} \end{aligned} \tag{4.6}$$

Furthermore, note that the reaction torque acting in the rotation direction of hinge i is zero when arm i is rotating between the folded and unfolded configurations ($\boldsymbol{y}_{A_i}^{A_i} \cdot \boldsymbol{\tau}_{r_i}^{A_i} = 0$), positive when arm i is in the folded configuration ($\boldsymbol{y}_{A_i}^{A_i} \cdot \boldsymbol{\tau}_{r_i}^{A_i} \geq 0$), and negative when arm i is in the unfolded configuration ($\boldsymbol{y}_{A_i}^{A_i} \cdot \boldsymbol{\tau}_{r_i}^{A_i} \leq 0$). Thus, in order for arm i to remain in a desired position when starting in that position (i.e. folded or unfolded), the vehicle must be controlled such that $\boldsymbol{y}_{A_i}^{A_i} \cdot \boldsymbol{\tau}_{r_i}^{A_i}$ remains either positive (to remain folded) or negative (to remain unfolded). Such a method is presented in the following section.

4.3 Control

In this section we describe the controllers used to control the vehicle in each of its configurations. We focus on three distinct configurations: the unfolded configuration (shown in Figure 4.1a), the two-arms-folded configuration (shown in Figures 4.1b and 4.1c), and the four-arms-folded configuration (shown in Figure 4.1e).

The vehicle is capable of controlled hover in both the unfolded and two-arms-folded configurations. In the unfolded configuration, the vehicle acts as a conventional quadcopter; each of the four propellers produce positive thrust forces ($f_{p_i} > 0$) in the $\boldsymbol{z}_C$ direction. However, in the two-arms-folded configuration, only two propellers of the same handedness produce positive thrust forces in the $\boldsymbol{z}_C$ direction; the other two propellers spin in reverse, producing negative thrust forces that cause their associated arms to fold downward. In this configuration, the folded arms are positioned such that the thrust forces produced by their associated propellers create a yaw torque that counteracts the yaw torque produced by the other two propellers. Note that for the design considered in this chapter, the arms have a 90° range of motion such that the thrust produced by a folded arm has no component in the $\boldsymbol{z}_C$ direction.

In the four-arms-folded configuration each of the four propellers are spun in reverse ($f_{p_i} < 0$), resulting in all four arms folding. Although the vehicle is not capable of controlled hover in this configuration, the attitude of the vehicle can still be fully controlled, allowing for the vehicle to reorient itself while in projectile motion.

The controllers used to control the vehicle in the unfolded and two-arms-folded configuration follow the same structure described in Section 2.4, with the main difference between the controllers being the mapping matrix used to compute the individual thrust forces described in Section 2.3. Specifically, the same cascaded control structure typical of multicopter control (shown in Figure 2.2) is used in both configurations, including both the position and attitude controllers described in Section 2.4.

A similar control structure is used in the four-arms-folded configuration. However, because the four arms do not produce thrust in the $\boldsymbol{z}_B$ direction while folded, we omit the position controller and instead command desired attitudes to the attitude controller directly. The individual thrust forces that minimize the sum of each thrust force squared while producing the desired torque are then computed. Note these thrust forces can be efficiently computed using the Moore-Penrose pseudoinverse of the matrix relating the four thrust forces to the torque acting on the vehicle.

Individual thrust force computation

As described in Section 2.3, the individual propeller thrust forces $u = (f_{p_1}, f_{p_2}, f_{p_3}, f_{p_4})$ are related to the desired total thrust in the $\boldsymbol{z}_C$ direction f_Σ and the desired torques about the axes of the body-fixed C frame, $\boldsymbol{\tau}^B = (\tau_x, \tau_y, \tau_z)$ through a mapping M given in (2.4). Recall that this mapping is computed using the geometry of the vehicle and the torque produced by each propeller as a function of the thrust it produces.

Because the thrust direction of each propeller is constant in the arm-fixed frame A_i rather than in the central body-fixed frame C, we modify the computation of the i-th columns of M_{f_Σ} and $M_{\boldsymbol{\tau}^B}$ as given in equation (2.5) to be the following:

$$M_{f_\Sigma}[i] = \boldsymbol{z}_{A_i}^C \cdot \boldsymbol{z}_C^C, \quad M_{\boldsymbol{\tau}^B}[i] = \boldsymbol{S}\left(\boldsymbol{d}_{P_iB}^C\right) \boldsymbol{z}_{A_i}^C + \kappa_{p_i} \boldsymbol{z}_{A_i}^C \tag{4.7}$$

In the unfolded configuration, the mapping matrix M_u remains identical to that of a conventional quadcopter as given by (2.6). However, the mapping M_{2f} for the two-arms-folded configuration with arms A_2 and A_4 folded and θ defined as shown in Figure 4.2 is defined as follows:

$$\begin{aligned} M_{2f} &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ -l/2 & p_x & l/2 & -p_x \\ -l/2 & -p_y & l/2 & p_y \\ -\kappa^+ & -p_z & -\kappa^+ & -p_z \end{bmatrix} \\ p_x &= d_{P_2C,z}^C \cos(45^\circ + \theta) - \kappa^- \sin(45^\circ + \theta) \\ p_y &= d_{P_2C,z}^C \sin(45^\circ + \theta) + \kappa^- \cos(45^\circ + \theta) \\ p_z &= \frac{l\sqrt{2}}{2} \sin\theta \end{aligned} \tag{4.8}$$

where we note that the arms are of equal length, i.e. $d_{P_1C,z}^C = d_{P_2C,z}^C = d_{P_3C,z}^C = d_{P_4C,z}^C$.

Finally, the mapping M_{4f} for the four-arms folded configuration is defined as:

$$M_{4f} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ p_x & p_x & -p_x & -p_x \\ p_y & -p_y & -p_y & p_y \\ p_z & -p_z & p_z & -p_z \end{bmatrix} \tag{4.9}$$

The structure of these mappings can be analyzed to infer how different parameters of the vehicle affect how the vehicle can be controlled in each configuration. For example, we note that when the arms are not angled (i.e. when $\theta = 0$ as was done in our prior work [38]), the term p_z as defined in (4.8) equals zero. In this case, the vehicle would only be able to produce negative yaw torques due to the fact that the folded arms would be unable to offset the yaw torque produced by the two unfolded arms. Similarly, if $\theta = 0$, the vehicle would be unable to produce any yaw torque in the four-arms-folded configuration as the bottom row of M_{4f} would be zero. However, we observe that (2.6) does not depend on θ at all, showing that the thrust mapping matrix is unaffected by the choice of arm angle in the unfolded configuration.

Note that, unlike M_u, both M_{2f} and M_{4f} depend on the position of the center of mass of the vehicle in the $\boldsymbol{z}_C$ direction due to the fact that the thrust forces produced by the folded arms are perpendicular to $\boldsymbol{z}_C$. Thus, if the position of the center of mass of the vehicle in the $\boldsymbol{z}_C$ direction is changed (e.g. by adding a payload to the vehicle as shown in Figure 4.1c), the mappings M_{2f} and M_{4f} must reflect this change.

Furthermore, because the thrust forces of the folded arms are perpendicular to $\boldsymbol{z}_C$, there can exist a nonzero force in the $\boldsymbol{x}_C$ and $\boldsymbol{y}_C$ directions when flying in the two- or four-arms folded configurations. In the two-arms-folded configuration with arms A_2 and A_4 folded, for example, thrusts f_{p_2} and f_{p_4} act in opposite directions such that they produce a force of magnitude $|f_{p_2} - f_{p_4}|$. Because this force is zero at hover and remains small for small τ_x and τ_y (note that $|f_{p_2} - f_{p_4}|$ is not dependent on f_Σ or τ_z due to the structure of M_{2f}), we choose to treat such forces as disturbances in order to maintain the simplicity of the proposed controller.

Attitude control

The attitude controller used to control the vehicle is identical to that described in Section 2.4, but uses different input cost matrices depending on the configuration of the vehicle. That is, equation (2.14) will result in a different input cost matrix $R_{\boldsymbol{\tau}^B}$ depending on which thrust mapping matrix is used, thus changing the feedback matrix computed when synthesizing the infinite-horizon LQR attitude controller.

Furthermore, in this work we choose the diagonal entries of R_u based upon whether the associated propeller is spinning in the forward or reverse direction, as the propeller exhibits different characteristics in each mode of operation. For example, conventional propellers produce significantly less thrust when spinning in the reverse direction as they are typically optimized to spin in only the forward direction. Thus, we define $R_u = \text{diag}(r^+, r^+, r^+, r^+)$ for the unfolded configuration, $R_u = \text{diag}(r^+, r^-, r^+, r^-)$ for the two-arms-folded configuration, and $R_u = \text{diag}(r^-, r^-, r^-, r^-)$ for the four-arms-folded configuration, where r^+ is the cost associated with the propellers spinning in the forward direction, and r^- is the cost associated with the propellers spinning in the reverse direction. In general, $r^+ < r^-$ as conventional quadcopter propellers are optimized to spin in the forward direction.

By defining the input cost matrix $R_{\boldsymbol{\tau}^B}$ as a function of the mapping matrix $M_{\boldsymbol{\tau}^B}$, we can straightforwardly synthesize different infinite-horizon LQR attitude controllers for each configuration of the vehicle. Furthermore, the torque cost matrix $R_{\boldsymbol{\tau}^B}$ can be used to analyze the ability of the vehicle to control its attitude in different configurations, as it describes the cost of producing an arbitrary torque on the vehicle while implicitly accounting for the geometry of the vehicle due to its dependence on $M_{\boldsymbol{\tau}^B}$.

Thrust limits

Although the thrust produced by each propeller is already bounded by the performance limitations of the motor driving it, we impose additional bounds which ensure the vehicle remains in the desired configuration. Imposing these bounds ensures that none of the arms begin to fold or unfold unexpectedly, which means the mappings M_u, M_{2f}, and M_{4f} derived in Section 4.3 will remain valid during flight. Of course, the bounds are not imposed when changing between configurations.

Rather than bounding the individual thrust forces, we choose to instead bound f_Σ and $\boldsymbol{\tau}^B$ using the model derived in Section 4.2. Our approach is similar to that of [38], but differs in its inclusion of the arm angle θ, resulting in a modified expression for the bound.

Unfolded configuration bounds

We first note that by enforcing bounds that prevent each arm from folding or unfolding, the vehicle can be treated as one rigid body rather than five coupled rigid bodies. Thus, the acceleration of the center of mass of the vehicle expressed in the inertial frame $\ddot{\boldsymbol{d}}^E_{BE}$ is:

$$\ddot{\boldsymbol{d}}^E_{BE} = \boldsymbol{g}^E + \frac{1}{m_\Sigma}\boldsymbol{R}^{EC}\boldsymbol{z}^C_C f_\Sigma \tag{4.10}$$

where the total vehicle mass is $m_\Sigma = m_C + 4m_{A_i}$.

Similarly, the angular acceleration of the vehicle can be written as follows, where $\boldsymbol{J}^B_\Sigma$ represents the moment of inertia of the vehicle taken at its center of mass and expressed in the body-fixed frame C. We assume that the angular velocity of the vehicle $\boldsymbol{\omega}^C_{CE}$ is small such that second order terms with respect to $\boldsymbol{\omega}^C_{=CE}$ can be neglected (e.g. $\boldsymbol{S}\big(\boldsymbol{\omega}^C_{CE}\big)\boldsymbol{J}^B_\Sigma\boldsymbol{\omega}^C_{CE}$).

$$\dot{\boldsymbol{\omega}}^C_{CE} = \left(\boldsymbol{J}^B_\Sigma\right)^{-1}\boldsymbol{\tau}^B \tag{4.11}$$

Next, after some algebraic manipulation of (4.3) and (4.5) (omitted here for brevity), we find that the reaction torque about hinge i, i.e. $\boldsymbol{y}^{A_i}_{A_i}\cdot\boldsymbol{\tau}^{A_i}_{r_i}$, is linear with respect to $\ddot{\boldsymbol{d}}^E_{BE}$, $\dot{\boldsymbol{\omega}}^C_{CE}$, and propeller thrust f_{p_i}. Recall that f_{p_i} can be computed by inverting the mapping given in (2.4), meaning that $\ddot{\boldsymbol{d}}^E_{BE}$, $\dot{\boldsymbol{\omega}}^C_{CE}$, and f_{p_i} are all linear functions of f_Σ and $\boldsymbol{\tau}^B$. Thus, we find that the torque about hinge i is also a linear function of f_Σ and $\boldsymbol{\tau}^B$.

As discussed previously, arm i will remain in the unfolded configuration when $\boldsymbol{y}^{A_i}_{A_i}\cdot\boldsymbol{\tau}^{A_i}_{r_i} \leq 0$. Therefore, because $\boldsymbol{y}^{A_i}_{A_i}\cdot\boldsymbol{\tau}^{A_i}_{r_i}$ is linear with respect to f_Σ and $\boldsymbol{\tau}^B$, the following four bounds can be computed that ensure each of the four arms remain in the unfolded configuration:

$$c_{f_i}f_\Sigma + c_{x_i}\tau_x + c_{y_i}\tau_y + c_{z_i}\tau_z \geq 0, \quad i \in \{1,2,3,4\} \tag{4.12}$$

where c_{f_i}, c_{x_i}, c_{y_i}, and c_{z_i} are all constants that depend on the physical attributes of the vehicle.

For the unfolded configuration, the constants in (4.12) are as follows. Here we have included the assumption that $\boldsymbol{J}^B_\Sigma = \mathrm{diag}(J^B_{\Sigma,xx}, J^B_{\Sigma,yy}, J^B_{\Sigma,zz})$ in order to allow for clearer analysis of c_{x_i}, c_{y_i}, and c_{z_i}. We give the magnitudes of each term, noting that c_{x_i}, c_{y_i}, and c_{z_i} have different signs depending which arm they are associated with.

$$c_{f_i} = \frac{1}{4}d^{A_i}_{P_iH_i,x} - d^{A_i}_{A_iH_i,x}\frac{m_{A_i}}{m_\Sigma} \tag{4.13}$$

$$|c_{x_i}| = \frac{d^{A_i}_{P_iH_i,x}}{2l} - \frac{\widetilde{J}^{A_i}_{A_i,yy}\cos(45^\circ+\theta) + \tilde{m}_A\sin(45^\circ+\theta)}{J^B_{\Sigma,xx}} \tag{4.14}$$

$$|c_{y_i}| = -\frac{d^{A_i}_{P_iH_i,x}}{2l} + \frac{\widetilde{J}^{A_i}_{A_i,yy}\sin(45^\circ+\theta) + \tilde{m}_A\cos(45^\circ+\theta)}{J^B_{\Sigma,yy}} \tag{4.15}$$

$$|c_{z_i}| = \frac{d^{A_i}_{P_iH_i,x}}{4\kappa^+} - \frac{\tilde{m}_A}{J^B_{\Sigma,zz}} \tag{4.16}$$

where $\tilde{m}_A$ and $\tilde{J}^{A_i}_{A_i,yy}$ are

$$\tilde{J}^{A_i}_{A_i,yy} = \left(\boldsymbol{J}^{A_i}_{A_i} + m_{A_i}\boldsymbol{S}\left(\boldsymbol{d}^{A_i}_{A_iH_i}\right)\boldsymbol{S}\left(\boldsymbol{d}^{A_1}_{CA_1}\right)\right)_{yy} \tag{4.17}$$

$$\tilde{m}_A = m_{A_i}d^{A_i}_{A_iH_i,x}d^{A_1}_{CH_1,y} \tag{4.18}$$

Because of the equal magnitudes of the constants c_{f_i}, c_{x_i}, c_{y_i}, and c_{z_i} in the unfolded configuration, we can aggregate the four bounds given in (4.12) into a single bound:

$$c_{f_i}f_\Sigma - |c_{x_i}\tau_x| - |c_{y_i}\tau_y| - |c_{z_i}\tau_z| \geq 0 \tag{4.19}$$

Note that (4.19) can always be satisfied by increasing f_Σ, as this corresponds to requiring each propeller to produce more thrust (note that in general $c_{f_i} > 0$). By examining (4.13), we observe that the bound becomes less restrictive when, e.g., the ratio of the mass of an arm to the total mass of the vehicle decreases, as this results in a larger magnitude c_{f_i}. Similarly, because the magnitude of c_{z_i} decreases as κ^+ increases, the bound can be made less restrictive by, e.g., choosing propellers with a larger magnitude κ^+.

Finally, note that by writing this bound as a function of f_Σ and $\boldsymbol{\tau}^B$, we can apply a similar method to that presented in [64] to reduce these control inputs in the event that the bound is not satisfied. Specifically, if the controller presented in the previous subsections produces a f_Σ and $\boldsymbol{\tau}^B$ that does not satisfy (4.19), we first reduce the magnitude of the yaw torque τ_z until the bound is satisfied or $\tau_z = 0$. Next, if the bound is still not satisfied, we increase f_Σ until the bound is satisfied or it reaches the maximum total thrust the propellers can produce. If the maximum total thrust is reached, then the roll and/or pitch torques are reduced until the bound is satisfied. In practice, however, decreasing the roll and/or pitch torques in order to prevent the arms from folding is seldom necessary due to the magnitude of c_{x_i} and c_{y_i} relative to the other terms.

Two- and four-arms-folded configuration bounds

Similar expressions for c_{f_i}, c_{x_i}, c_{y_i}, and c_{z_i} can be found for the two- and four-arms-folded configurations, which we compute using a computer algebra system due to their algebraic complexity (and thus omit here for brevity).[1] Note that no aggregate bound such as (4.19) exists for the two- or four-arms-folded configuration, and thus it is necessary to enforce each

[1]We provide code for computing these bounds, as well as performing much of the other analyses and controller syntheses described in this chapter, at: `https://github.com/nlbucki/MidairReconfigurableQuadcopter`

bound given by (4.12) individually. However, the hierarchical modification of the control inputs f_Σ and $\boldsymbol{\tau}^B$ described previously can still be used to ensure the bounds are satisfied, guaranteeing that the vehicle remains in the desired configuration under the previously stated assumptions.

Numerical values for c_{f_i}, c_{x_i}, c_{y_i}, and c_{z_i} are given in Section 4.4 for the experimental vehicle in both the unfolded and two-arms-folded configurations. We do not provide such values for the four-arms-folded configuration, as in practice we have found it to be unnecessary to enforce such bounds. This is because the thrust forces required to transition into the four-arms-folded configuration are typically large enough to prevent the arms from unfolding without the need to enforce additional bounds.

Configuration transitions

Next we describe the method used to transition between configurations of the vehicle. We choose to focus on the transitions between the unfolded and two-arms-folded configurations as well as between the unfolded and four-arms-folded configurations, as these are the only transitions required to produce the behaviors of the vehicle demonstrated in this chapter. An example of the transition from the unfolded configuration to the two-arms-folded configuration and back is given in Section 4.5, and an example of the transition from the unfolded configuration to the four-arms-folded configuration and back is given in Section 4.5.

When transitioning between the unfolded and two-arms-folded configurations, we have found it sufficient to instantaneously change between the controller used in the unfolded configuration and the controller used in the two-arms-folded configuration. This discrete change in controllers is largely enabled by the fact that the vehicle possesses significant enough agility in either configuration to recover from small disturbances encountered during the transition. However, the transition is complicated by the fact that the vehicle experiences a significant yaw disturbance during the transition. This yaw disturbance occurs because it is necessary to reverse the rotation direction of two of the propellers of the same handedness during the transition. Specifically, the reversing propellers cannot offset the yaw torque produced by the propellers attached to the unfolded arms, which remain spinning in the forward direction. Additionally, the revering propellers experience a change in angular momentum that results in a corresponding change in angular velocity of the vehicle. Thus, after completing the maneuver, the vehicle will have a significantly different yaw angle and yaw rate than when the maneuver was initiated. In practice, we deal with this difference in yaw angle by choosing the post-transition desired yaw angle such that once the maneuver is completed the yaw error is small.

Unlike the transition to or from the two-arms-folded configuration, the transitions between the unfolded and four-arms-folded configurations are accomplished by commanding constant forward or reverse thrusts while the four arms are moving to the unfolded or folded configurations respectively. After all four arms have finished transitioning, we resume controlling the vehicle using either the unfolded or two-arms-folded controller as appropriate. The period of constant thrusts is required to ensure that all four arms fold or unfold simulta-

neously, and prevents any attitude errors that would otherwise be introduced by attempting to control the vehicle while the arms are transitioning (as M_u and M_{4f} would not be valid during the transition).

4.4 Experimental Vehicle Design

In this section we discuss the design of the experimental vehicle shown in Figure 4.1. We start by describing how the arm angle was chosen based upon other properties of the vehicle, then discuss how the properties of the chosen powertrain (i.e. the battery, speed controllers, motors, and propellers) affect the vehicle design, and finally discuss how the design of the vehicle influences several important properties of the proposed controllers for each configuration of the vehicle.

The properties of the experimental vehicle are given in Table 4.1. The overall dimensions of the experimental vehicle were chosen to be as similar as possible to a commonly used quadcopter design. Specifically, 8 in propellers spaced 24 cm apart are used, which correspond to the same spacing and size of the propellers that would be used with a DJI F330 frame (e.g. as used in [18]). We chose to use commonly available components in the vehicle design to demonstrate its similarity to a conventional quadcopter, and designed the vehicle to have a similar performance (in terms of power consumption and agility) as a conventional quadcopter when flying in the unfolded configuration.

Onboard the vehicle, a Crazyflie 2.0 flight controller is used to run the attitude controller and to transmit individual propeller angular velocity commands to four DYS SN30A electronic speed controllers (ESCs) at 500Hz. The vehicle is powered by a three cell, 40C, 1500 mA h LiPo battery, and four EMAX MT2208 brushless motors are used to drive four Gemfan 8038 propellers.

A motion capture system is used to localize the vehicle, although in principal any sufficiently accurate localization method (e.g. using onboard cameras) could be used. Note that we do not directly measure the position of any individual arm of the vehicle, and instead only measure the position and attitude of the central body of the vehicle. The position and attitude of the vehicle are measured by the motion capture system at 200Hz, and the angular velocity of the vehicle is measured at 500Hz using an onboard rate gyroscope. The position controller runs on an offboard laptop and sends commands to the vehicle via radio at 50Hz.

Choice of arm angle

We choose the angle that each arm makes with the diagonal of the vehicle θ, as shown in Figure 4.2, such that the vehicle is capable of hovering in the two-arms-folded configuration. That is, the vehicle should be capable of producing a total thrust f_Σ to offset gravity while producing zero torque on the vehicle.

Let the thrust each propeller can produce be bounded by $f_{p_i} \in [f_{\min}, f_{\max}]$, where $f_{\min}$ and $f_{\max}$ are determined by the physical limits of the powertrain of the vehicle. Note that

Table 4.1: Experimental Vehicle Parameters

Symbol	Parameter	Value
m_{A_i}	Arm mass	67 g
m_C	Central body mass	356 g
m_Σ	Total vehicle mass	624 g
κ^+	Propeller torque per unit positive thrust	0.0172 Nm/N
κ^-	Propeller torque per unit negative thrust	0.038 Nm/N
$f_{\min}$	Minimum thrust per propeller	−3.4 N
$f_{\max}$	Maximum thrust per propeller	7.8 N
θ	Arm angle	11.9°
l	Distance between adjacent propellers	24 cm
$\boldsymbol{d}^C_{CH_1}$	Position of central body center of mass relative to hinge 1 (written in B frame)	$\begin{pmatrix} -4.5\,\text{cm} \\ 7.1\,\text{cm} \\ -0.2\,\text{cm} \end{pmatrix}$
$\boldsymbol{d}^{A_i}_{H_iA_i}$	Position of hinge relative to arm center of mass (written in arm frame)	$\begin{pmatrix} -7.6\,\text{cm} \\ 0\,\text{cm} \\ -1.4\,\text{cm} \end{pmatrix}$
$d^{A_i}_{PA_i,x}$	Distance of propeller from arm center of mass	1.4 cm

in our case $f_{\min} < 0$ unlike conventional quadcopters which only allow propellers to spin in the forward direction.

We wish to find θ such that $M_{2f}u = (m_\Sigma g, 0, 0, 0)$ with M_{2f} as given in (4.8) while satisfying constraints on the thrusts each propeller can produce. As the constraints $\tau_x = \tau_y = 0$ can be trivially satisfied for any choice of θ when $f_{p_1} = f_{p_3}$ and $f_{p_2} = f_{p_4}$, we focus on the constraints on the total thrust f_Σ and yaw torque τ_z:

$$f_{p_1} + f_{p_3} \geq m_\Sigma g \tag{4.20}$$

$$-\kappa^+ (f_{p_1} + f_{p_3}) - \frac{l\sqrt{2}}{2} \sin\theta \, (f_{p_2} + f_{p_4}) = 0 \tag{4.21}$$

Thus, the following two inequalities must be satisfied in order for the vehicle to be able to hover with two arms folded:

$$\begin{gathered} \theta \geq \sin^{-1}\left(\frac{-\kappa^+ m_\Sigma g}{l\sqrt{2} f_{\min}}\right) \\ f_{\max} \geq \frac{m_\Sigma g}{2} \end{gathered} \tag{4.22}$$

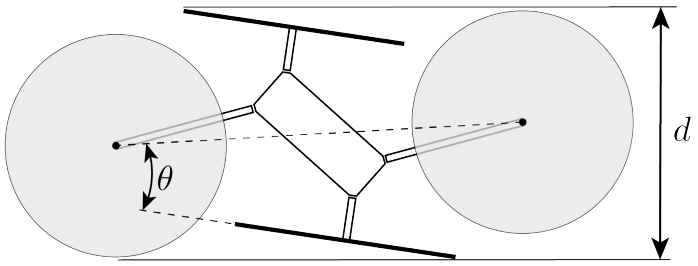

Figure 4.3: Top-down view of the vehicle in the two-arms-folded configuration. The minimum horizontal dimension of the vehicle d increases as the arm angle θ increases.

Because the geometry of the experimental vehicle is defined such that an increase in θ corresponds to an increase in the minimum dimension d of the vehicle in the two-arms-folded configuration as shown in Figure 4.3, we choose the smallest θ such that the vehicle has sufficient control authority to produce reasonable magnitude thrusts and torques with two arms folded. Specifically, we choose

$$\theta = \sin^{-1}\left(\frac{-\kappa^{+} m_{\Sigma} g}{l\sqrt{2} f_{\text{des}}}\right) \tag{4.23}$$

where $f_{\text{des}} > f_{\min}$ is the nominal thrust force produced by each of the two folded arms during hover. We choose f_{des} to be roughly half $f_{\min}$ (recall $f_{\min} < 0$) so that the vehicle is capable of producing roughly equal magnitude yaw torques in each direction.

Note that the bound presented in (4.22) is also dependent on several other parameters of the vehicle. For example, if a smaller θ is desired, it is advantageous to minimize both the mass of the vehicle m_{Σ} and the coefficient κ^{+} that relates the thrust produced by each propeller to the torque acting about its rotation axis. Coincidentally, minimizing these quantities is equivalent to minimizing the power consumption of the vehicle at hover, which is typically a preeminent concern when designing aerial vehicles. Thus, no significant trade-off exists between the power consumption of the vehicle and the choice of θ.

Powertrain selection

As discussed in the previous subsection, the arm angle θ is dependent on both the torque per unit positive thrust produced by each propeller κ^{+} as well as the maximum magnitude thrust each propeller can produce when spinning in reverse $f_{\min}$. Thus, in order to minimize θ, the ratio between κ^{+} and $f_{\min}$ must be minimized. To this end, the powertrain (i.e. battery, speed controllers, motors, and propellers) is chosen such that θ is minimized while simultaneously minimizing the power consumption of the vehicle while flying in the unfolded configuration, as this would likely be the primary mode of operation of the vehicle. In our

model, $f_{\min}$ and $f_{\max}$ are determined by the design of the powertrain, and κ^+ and κ^- are determined by the chosen propellers.

Although we spin several of the propellers in the reverse direction in the two- or four-arms-folded configurations, this does not imply that it would necessarily be advantageous to use symmetric propellers (sometimes referred to as "3D propellers") which are designed to spin in both directions. When compared to conventional propellers, symmetric propellers have the advantage of being able to produce much larger thrusts when spinning in reverse (i.e. $f_{\min}$ is larger in magnitude), but this comes at the cost of a smaller maximum forward thrust $f_{\max}$ and a larger torque per unit positive thrust κ^+. Thus, it is possible that the use of symmetric propellers may lead to a larger required θ if the ratio of κ^+ to $f_{\min}$ is larger than that of a conventional propeller. Additionally, $f_{\max}$ must still be large enough to satisfy the constraint given in (4.22), which may be difficult to achieve using symmetric propellers. Finally, the use of symmetric propellers would greatly increase the power consumption of the vehicle when hovering in the unfolded configuration, as symmetric propellers are not optimized to minimize power consumption compared to conventional propellers.

To this end, we choose to use conventional quadcopter propellers on the experimental vehicle. Figure 4.4 shows how the thrust and torque produced by a Gemfan 8038 propeller are related to the rotational speed of the propeller, demonstrating the difference in thrust produced by the propeller when spinning in the forward and reverse directions. We found that the powertrain of the experimental vehicle was capable of driving the propeller to produce up to 3.4 N of thrust in the reverse direction and 7.8 N of thrust in the forward direction with $\kappa^+ = 0.0172\,\text{Nm/N}$ and $\kappa^- = 0.038\,\text{Nm/N}$. This lead to a choice of $\theta = 11.9°$ according to (4.23) with $f_{\text{des}} = 1.5\,\text{N}$.

Finally, we note that although in theory f_{p_i} can achieve any value between $f_{\min}$ and $f_{\max}$, in practice we restrict f_{p_i} to not pass through zero unless the vehicle is performing a configuration transition that requires reversing the propeller. This is due to the fact that we use commonly available electronic speed controllers and brushless motors which use back-EMF to sense the speed of the motor. The use of back-EMF to sense motor speed results in significantly degraded performance when changing directions, meaning that such motors are typically restricted to spin in only one direction. Although this property can affect the performance of the proposed vehicle when changing between configurations, once the propellers have reversed direction they can continue to operate without any significant change in performance. Thus, in practice we restrict the thrust forces of propellers spinning in the forward direction and reverse direction to be in $[0, f_{\max}]$ and $[f_{\min}, 0]$ respectively, and only allow the propellers to change direction when changing between configurations.

Vehicle Agility

We now examine the effects of the bounds described in Section 4.3 on the experimental vehicle with thrust limits $f_{\min}$ and $f_{\max}$. For notational convenience, we define $W \in \mathbb{R}^{4\times4}$ as a matrix with each column defined by $c_{f_i}, c_{x_i}, c_{y_i}$, and c_{z_i} respectively. Then, the bounds

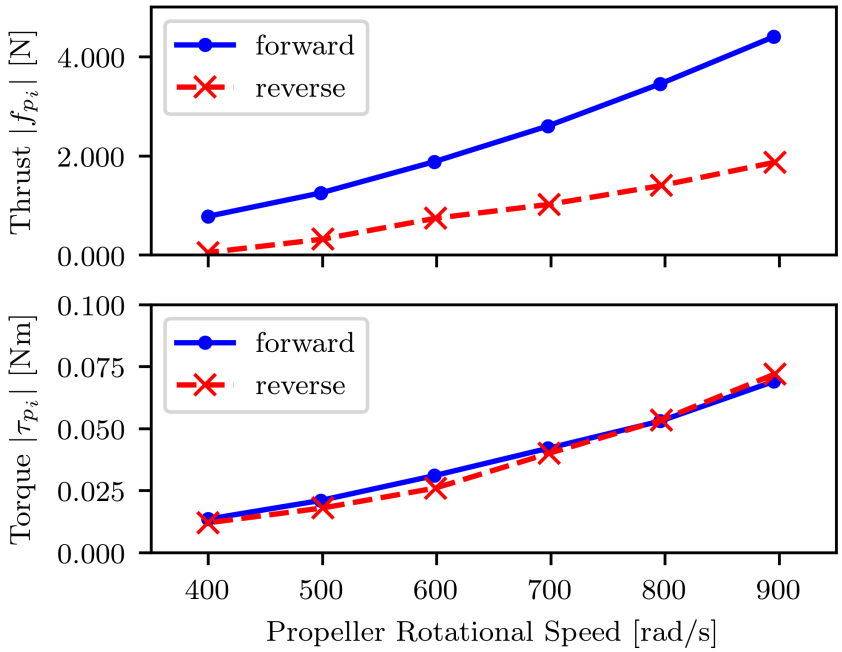

Figure 4.4: Magnitude of thrust and torque produced by an 8038 propeller spinning in both the forward and reverse directions. A load cell capable of measuring forces and torques was used in conjunction with an optical tachometer to collect the data. The propeller produces significantly more thrust but produces roughly the same magnitude torque when spinning in the forward direction compared to the reverse direction for a given speed.

defined in (4.12) can be rewritten as:

$$W \begin{bmatrix} f_\Sigma \\ \boldsymbol{\tau}^B \end{bmatrix} \succeq \mathbf{0} \tag{4.24}$$

where $\succeq$ denotes an element-wise inequality, and $\mathbf{0}$ denotes a vector of zeros.

Then, the matrix W_u for the experimental vehicle in the unfolded configuration is computed to be the following, where the first column has units of meters and the other columns are unitless.

$$W_u = \begin{bmatrix} 0.0144 & -0.0421 & -0.0252 & -1.304 \\ 0.0144 & -0.0421 & 0.0252 & 1.304 \\ 0.0144 & 0.0421 & 0.0252 & -1.304 \\ 0.0144 & 0.0421 & -0.0252 & 1.304 \end{bmatrix} \tag{4.25}$$

Similarly, the matrix W_{2f} for the experimental vehicle in the two-arms-folded configura-

tion is computed to be:

$$W_{2f} = \begin{bmatrix} 0.0369 & 0.08 & 0.0225 & 0.0059 \\ 0.0237 & 0.345 & -0.289 & 1.26 \\ 0.0369 & -0.08 & -0.0225 & 0.0059 \\ 0.0237 & -0.345 & 0.289 & 1.26 \end{bmatrix} \tag{4.26}$$

The individual thrust limits of each propeller can be written in terms of f_Σ and $\boldsymbol{\tau}^B$ by utilizing the inverse of the mapping matrix M introduced in (2.4):

$$\begin{bmatrix} \boldsymbol{I} \\ -\boldsymbol{I} \end{bmatrix} M^{-1} \begin{bmatrix} f_\Sigma \\ \boldsymbol{\tau}^B \end{bmatrix} \succeq \begin{bmatrix} \mathbf{1} f_{\min} \\ -\mathbf{1} f_{\max} \end{bmatrix} \tag{4.27}$$

where $\boldsymbol{I}$ the 4×4 identity matrix, and $\mathbf{1}$ is vector of ones of length four.

In order to compare the agility of the experimental vehicle to a conventional quadcopter, we examine how the set of feasible values of f_Σ and $\boldsymbol{\tau}^B$ is reduced when imposing the bounds given in (4.24) (i.e. those that prevent the arms from folding or unfolding). Note that both the experimental vehicle and a conventional quadcopter must satisfy the bounds on f_Σ and $\boldsymbol{\tau}^B$ given by (4.27) (i.e. those that ensure $f_{p_i} \in [f_{\min}, f_{\max}]$), but that the experimental vehicle must additionally satisfy the bounds that prevent the arms from folding or unfolding.

The reduction in agility of the experimental vehicle when $\tau_x = \tau_y = 0$ is shown in Figure 4.5, where we observe how the set of feasible yaw torques τ_z and total thrusts f_Σ is reduced in comparison to a conventional quadcopter. As shown in the figure, the bounds that prevent the arms from folding primarily result in a reduction in the range of feasible yaw torques. Specifically, the maximum yaw torque the experimental vehicle can produce at hover (i.e. when $f_\Sigma = m_\Sigma g$ and $\tau_x = \tau_y = 0$) is reduced by 36% when compared to a conventional quadcopter. We note that this is a significant improvement from our previous work [38], where the maximum yaw torque was reduced by roughly 75% when compared to a conventional quadcopter due to the use of springs to fold the arms rather than reverse thrust as we use in this work.

A similar analysis of the maximum magnitude roll and pitch torques the vehicle can produce at hover shows them to be no less than those of a conventional quadcopter, indicating that the bounds that prevent the arms from folding given in (4.24) are actually less restrictive than those on each of the individual thrust forces given in (4.27). Finally, we find that the minimum and maximum total thrust forces are also no less than those of a conventional quadcopter, which is also an improved result from our previous work [38] where we found that the minimum total thrust force was 70% of the thrust force required to hover (again due to the use of springs to fold the arms). Thus, this analysis implies that the only significant tradeoff between the proposed vehicle design and a conventional quadcopter (in terms of the control authority of the vehicle) is the reduction of the maximum yaw torque the vehicle can produce.

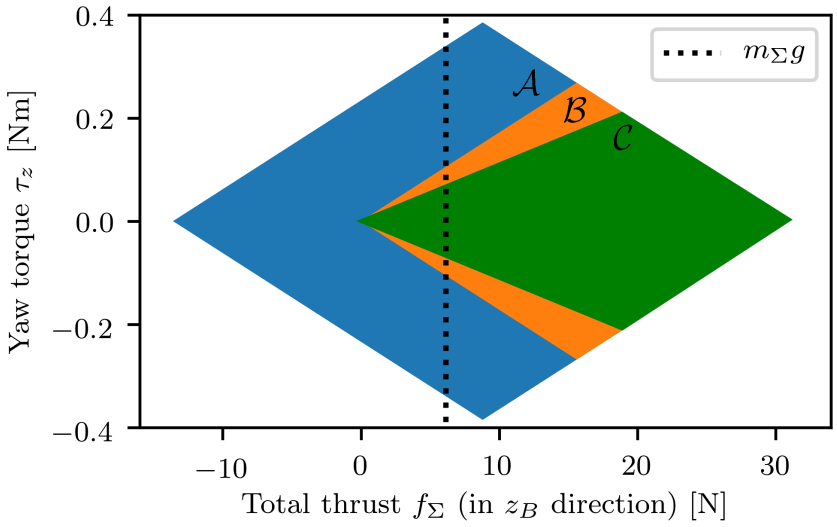

Figure 4.5: Range of feasible total thrusts f_Σ and yaw torques τ_z for the experimental vehicle in the unfolded configuration with zero roll and pitch torques $\tau_x = \tau_y = 0$. The dotted black line denotes the value of f_Σ at hover. The blue set $\mathcal{A}$ represents the feasible inputs when only the constraints on the minimum and maximum thrusts of each propeller $f_{\min}$ and $f_{\max}$ are considered. The orange set $\mathcal{B}$ represents the feasible inputs for a conventional quadcopter, i.e. with $f_{\min} = 0$ rather than $f_{\min} < 0$. Finally, the green set $\mathcal{C}$ represents the feasible inputs when the constraints that prevent the arms from folding are imposed, primarily reducing the range of feasible yaw torques. Note that $\mathcal{C} \subset \mathcal{B} \subset \mathcal{A}$.

4.5 Experimental Results

In this section, we demonstrate how the ability of the proposed vehicle to fold and unfold each arm enables it to perform a number of tasks which would be difficult or impossible to perform using a conventional quadcopter. We first show how the ability to fold two arms of the vehicle enables the vehicle to fly horizontally through narrow tunnels and perform simple aerial grasping, and then demonstrate how all four arms of the vehicle can be folded to perform perching and more aggressive vertical flight through narrow gaps.[2]

Horizontal flight through a narrow tunnel

We first demonstrate how the proposed vehicle can be used to fly in confined spaces which would normally be inaccessible to a conventional quadcopter of similar size. The vehicle

[2]Videos of each of the experiments discussed in this section can be viewed in the attached video or at `https://youtu.be/xEg8GXlb82g`

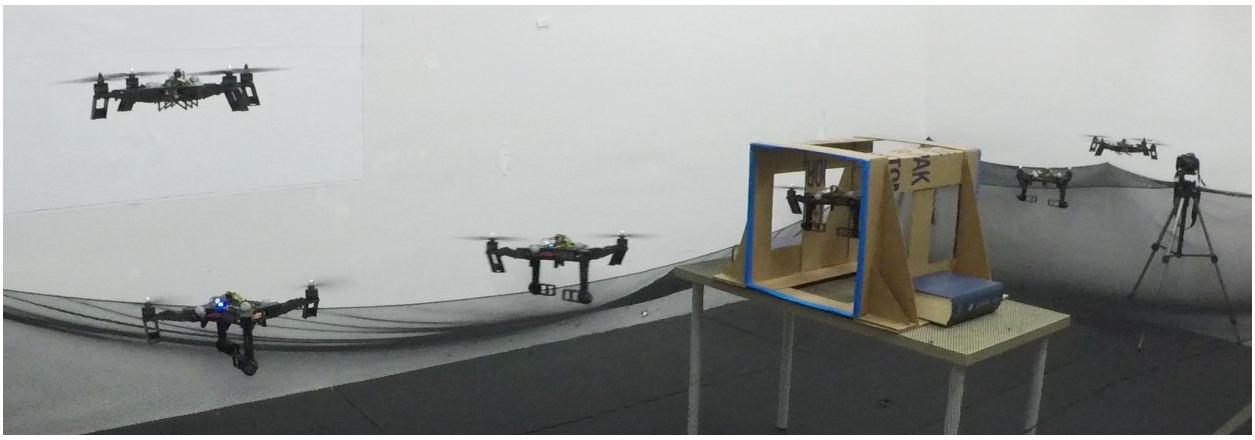

Figure 4.6: Composite image of the vehicle transitioning from the unfolded to two-arms-folded configuration (left), flying through a narrow tunnel, and transitioning back to the unfolded configuration (right).

was flown through a tunnel with a cross section that measures 43 cm by 43 cm, as shown in Figure 4.1b. These dimensions were chosen such that the vehicle could not traverse the tunnel in the unfolded configuration even with perfect trajectory tracking, as the minimum width of the vehicle in the unfolded configuration is 43 cm. However, the minimum width of the vehicle in the two-arms-folded configuration is 24 cm, allowing it to pass.

To perform the maneuver, the vehicle first transitions from the unfolded configuration to two-arms-folded configuration, then flies through the tunnel, and finally transitions back to the unfolded configuration as shown in Figure 4.6. The yaw angle of the vehicle was chosen to maximize the distance of the vehicle from the walls of the tunnel when flying through its center.

Grasping

Next, we show how the two-arms-folded configuration can be used to perform a simple grasping task, as shown in Figure 4.7. In this experiment a box with a mass of 83 g that measures 9 cm × 15 cm × 25 cm in height is used. The box was specifically chosen to be 9 cm in width in order to allow for the box to be grasped without significantly changing the geometry of the two-arms-folded configuration, as the distance between the legs of two opposing folded arms is approximately 9 cm. Note that because the total mass of the vehicle m_Σ increases when holding the box, each of the bounds given in (4.22) that govern the ability of the vehicle to hover in the two-arms-folded configuration become more restrictive, significantly limiting the maximum mass of a box that can be carried.

The experiment was conducted as follows: The vehicle was first commanded to land on

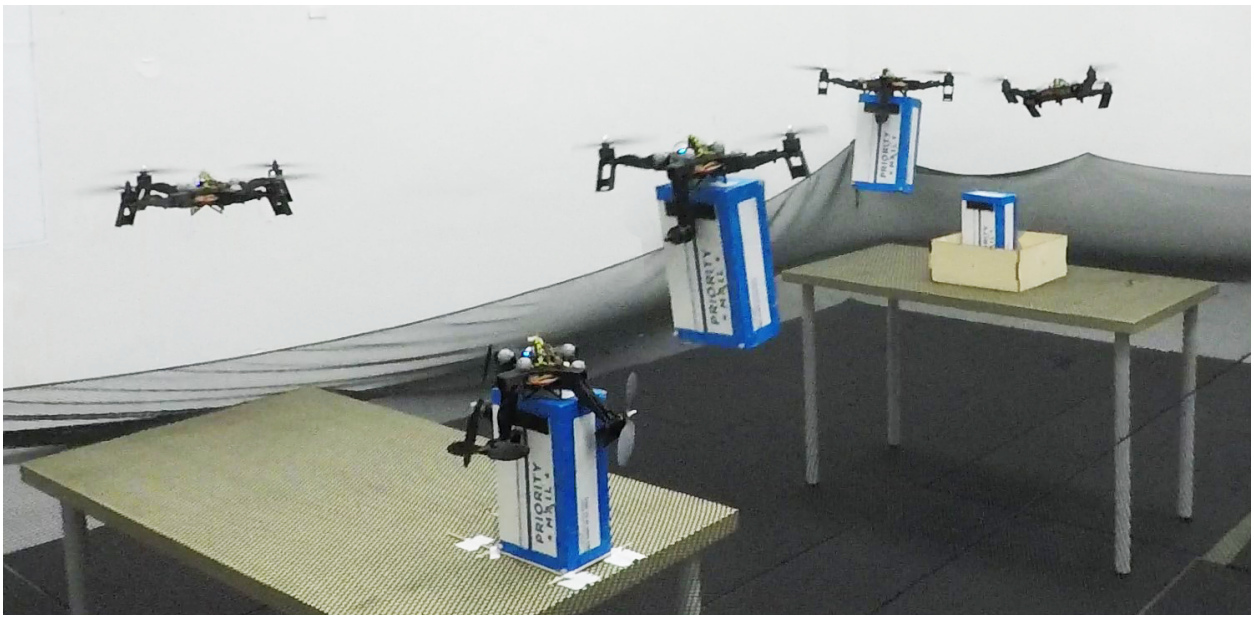

Figure 4.7: Composite image of the vehicle grasping a box (left), flying it to a new location, and dropping the box by returning to the unfolded configuration (right).

top of the box, which was constrained such that it could not rotate in the yaw direction. After landing, all four propellers were disabled, allowing two of the arms of the vehicle to fall into grasping position. Next, the two arms used to grasp the box were commanded to produce a thrust of $-2\,\mathrm{N}$ for one second to allow the arms to settle into a firm grasping position, after which time the two unfolded arms were commanded to produce a small thrust of $1\,\mathrm{N}$ for one second such that they fully unfolded before takeoff. After this grasping procedure was completed, the vehicle was commanded to takeoff and fly to the desired drop-off location using the two-arms-folded configuration controller, which was modified to account for the change in location of the center of mass of the vehicle as discussed in Section 4.3. After flying to the drop-off location, the vehicle was commanded to transition back to the unfolded configuration, resulting the the box being released at the desired location.

Wire perching

The vehicle is also capable of perching on wires in the four-arms-folded configuration, as shown in Figure 4.1d. To perform this maneuver, the vehicle simply aligns itself with the wire and lands on top of it, turning off all four motors when the maneuver is complete. The body of the experimental vehicle includes a notch that runs the length of the central body the vehicle, which helps align the vehicle with the wire when perching. Because only the

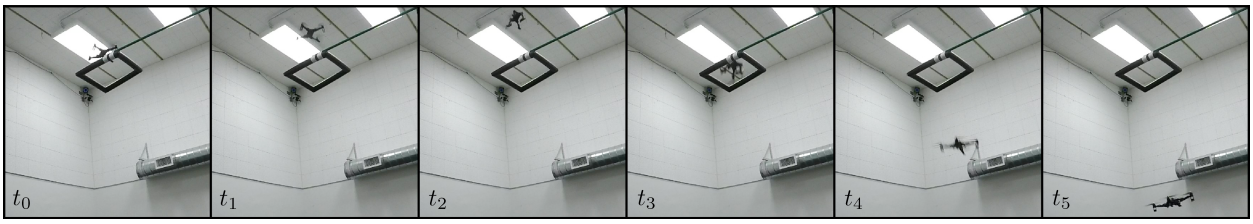

Figure 4.8: Image sequence of the vehicle transitioning from the unfolded to the four-arms-folded configuration and back in order to traverse a narrow gap. Data associated with this experiment is shown in Figure 4.9.

central body of the vehicle is supported by the wire, the four arms fold downward. This shifts the center of mass of the vehicle below the wire, which allows the vehicle to perch on the wire in a stable configuration. For the experimental vehicle, the center of mass is shifted 4 cm downward by folding the arms, resulting in the center of mass of the vehicle being 2 cm below where the wire contacts the vehicle.

Vertical flight through a narrow gap

Finally, we demonstrate capability of the vehicle to fold all four arms during flight, allowing for passage through narrow gaps in projectile motion. The maneuver is inspired in part by how birds fold their wings when passing through narrow gaps, as shown in [65], and mirrors our previous work [38], where we demonstrated a similar capability using springs to fold the arms rather than reverse thrust forces. Here we only show the vehicle traversing a gap vertically, as the traversal of gaps in the horizontal direction can be accomplished using the two-arms folded method demonstrated in Section 4.5. The gap measures 43 cm by 43 cm, and the experimental vehicle measures 27 cm by 35 cm in the four-arms-folded configuration.

Figure 4.8 shows images of the gap traversal maneuver, and Figure 4.9 graphs the trajectory of the vehicle during the maneuver, which consists of the following stages. First, the vehicle aligns itself with the gap while hovering above it. Once aligned, the vehicle begins to accelerate upward from time $t_0 = 0.2\,\text{s}$ to time $t_1 = 0.46\,\text{s}$. After completing this upward trajectory, a constant thrust command of $-1\,\text{N}$ is sent to each propeller at time t_1. At time $t_2 = 0.84\,\text{s}$ the arms finish the transition to the folded configuration, and the four-arms-folded attitude controller is used to stabilize the vehicle, where the desired attitude is chosen such that $\boldsymbol{z}_C$ is in the vertical direction. Next, at time $t_3 = 0.96\,\text{s}$, a constant thrust command of 1 N is sent to each propeller in order to unfold the arms. The vehicle traverses the gap (located at 3.3 m in this experiment) at approximately this time. Then, at time $t_4 = 1.21\,\text{s}$, the arms finish unfolding as evidenced by a sharp increase in the acceleration of the vehicle in the $\boldsymbol{z}_C$ direction. At this time the unfolded configuration controller is once again enabled,

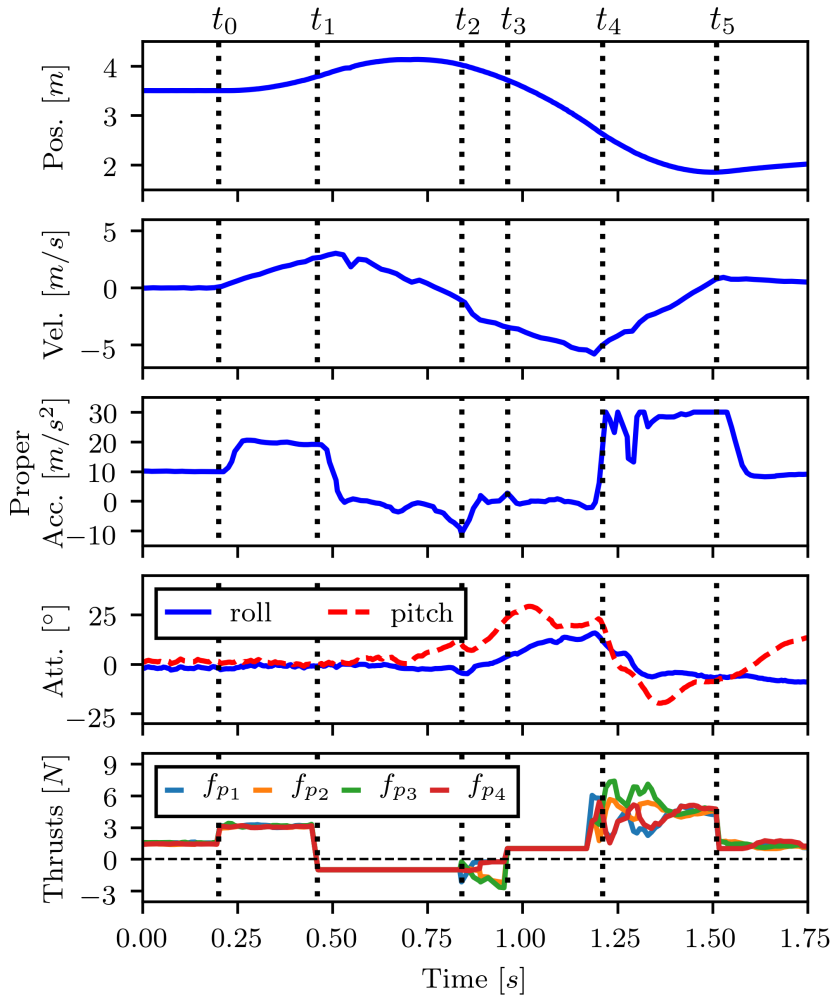

Figure 4.9: Trajectory of the vehicle while passing downward through a narrow gap in the four-arms-folded configuration. The position and velocity of the vehicle are given in the vertical $\boldsymbol{z}_E$ direction as measured by the motion capture system, and the proper acceleration is given in the $\boldsymbol{z}_C$ direction as measured by the onboard accelerometer. The vehicle starts accelerating upward at time t_0, and commands each propeller to produce a constant negative thrust at time t_1, initiating the transition to the four-arms-folded configuration. At time t_2 the arms finish folding, and the four-arms-folded controller is used to stabilize the attitude of the vehicle. Next, at time t_3, a constant positive thrust command is sent to each motor to initiate the transition back to the unfolded configuration, resulting in the vehicle returning to the unfolded configuration at time t_4. Finally, the vehicle is commanded to accelerate upward to reduce its downward velocity until the vehicle comes to rest at time t_5.

and the vehicle is commanded to produce a large vertical acceleration until the vertical speed of the vehicle is reduced to zero, which occurs at time $t_5 = 1.51\,\text{s}$.

Note that although using larger constant thrust commands than 1 N to fold and unfold the arms would result in the arms folding/unfolding more quickly, in practice we have found it preferable to command smaller constant thrust values. This is due to the fact that the arms may not fold at exactly the same time (e.g. due to friction), and thus large constant thrusts may result in large torques being exerted on the vehicle, leading to potentially large attitude errors once the transition is completed. The reduction of attitude errors in the four-arms-folded configuration is crucial because it ensures that the thrust direction of the vehicle will be in the opposite direction of its velocity after transitioning back to the unfolded configuration, allowing for the vehicle to quickly reduce its speed.

4.6 Conclusion

In this chapter we have presented a novel quadcopter design that differs from a conventional quadcopter in the use of passive hinges which allow each of the four arms to rotate freely between unfolded and folded configurations. The vehicle was designed to be nearly identical to a conventional quadcopter aside from the presence of the four passive hinges, which are lightweight and thus do not significantly affect the power consumption of the vehicle. Although these additional unactuated degrees of freedom require stricter bounds on the thrust forces produced by each propeller, these additional bounds were shown to not significantly affect the agility of the vehicle when flying in the unfolded configuration, aside from a reduced ability to produce yaw torques. Additionally, a method for easily synthesizing controllers for the different configurations of the vehicle was presented and used to control the attitude of the vehicle in both the two- and four-arms-folded configurations.

The design of the vehicle was also analyzed based upon the ability of the vehicle to hover in the two-arms-folded configuration. Specifically, it was shown that the angle of the arms relative to the central body is bounded from below by a function of the characteristics of the propellers and the mass and size of the vehicle. This lower bound, however, is structured such that it becomes less strict as the power consumption of the vehicle in the unfolded configuration is reduced, meaning that no tradeoff exists between vehicle power consumption and arm angle. A simple characterization of a conventional quadcopter propeller was also performed, showing that although significantly less thrust is produced by the propeller when spinning in reverse, such propellers can produce enough reverse thrust to enable the vehicle to be controlled in the two-arms-folded configuration with a reasonably small arm angle.

Finally, the viability of the design was demonstrated by constructing an experimental vehicle using commonly available quadcopter components (e.g. standard propellers, motors, etc.), which was shown to be capable of performing a number of tasks that a conventional quadcopter could not perform.

Chapter 5

Computationally Efficient Trajectory Generation in Known Environments

In the previous two chapters we have examined various novel quadcopter designs that enable a wider range of tasks to be achieved than what a conventional quadcopter could achieve. Similarly, in the following two chapters we investigate how novel path planning algorithms can be used to reduce the computational resources required to fly a quadcopter autonomously, and argue that this enables lower cost/power computational hardware to be used than what is used on conventional autonomous quadcopters.

Specifically, this chapter presents a continuous-time collision detection algorithm for quickly detecting whether certain polynomial trajectories in time intersect with convex obstacles. The algorithm is used in conjunction with an existing quadcopter trajectory generation method to achieve rapid, obstacle-aware motion planning in environments with both static convex obstacles and dynamic convex obstacles whose boundaries do not rotate. In general, this problem is difficult because the presence of convex obstacles makes the feasible space of trajectories nonconvex. The performance of the algorithm is benchmarked using Monte Carlo simulations, and experimental results are presented that demonstrate the use of the method to plan collision-free quadcopter trajectories in milliseconds in environments with both static and dynamic obstacles.

Note that the material presented in this chapter is based on the following previously published work, differing primarily in the notation used to describe the dynamics of the vehicle.

- Nathan Bucki and Mark W Mueller. "Rapid Collision Detection for Multicopter Trajectories". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019

5.1 Introduction

A key enabler of the use of autonomous systems in real-world situations is a fast method for generating state and input feasible trajectories between desired states. This problem is know as the motion planning problem, and is a well researched area that includes numerous methods for the generation of such trajectories. In particular, sampling-based methods such as rapidly exploring random trees (RRT) [67], probabilistic roadmaps (PRM) [68], and fast marching trees (FMT) [69], have been used with great success to construct collision-free paths between desired states. Such methods are typically performed by sampling the state space of the system and attempting to connect feasible sampled nodes with simple trajectories that do not collide with obstacles. Performing collision detection on both the sampled nodes and the trajectories that connect them is often considered the most computationally expensive step in the motion planning process, and will be the focus of this chapter.

Previous work focusing on the reduction of collision detection time includes [70], which presents an algorithm that involves using distance information from previously sampled nodes to avoid performing explicit node-obstacle collision detection when possible. In [71] this idea is adapted to reduce collision detection time for quadcopter trajectories by computing overlapping collision-free spheres around the trajectory based on the maximum velocity of the vehicle and distance to the nearest obstacle at each sample point.

Rather than generating a number of quadcopter trajectories and then checking each one for collisions, the authors of [72] first compute a series of overlapping, obstacle-free polyhedrons and then generate a series of trajectory segments that remain inside the polyhedrons. In [73], the authors take a similar approach by using an octree-based representation of the environment in order to enforce corridor constraints on each trajectory segment generated. A third approach to collision avoidance for aggressive flight is explored in [74], where a dense set of alternative trajectories to some desired trajectory are precomputed, allowing for one of the alternative trajectories to be chosen if a collision is predicted along the desired trajectory. In this case, a collision is determined by comparing the distance of each point along the discretized candidate trajectory to the points in a point cloud generated by a laser scanner.

In contrast to methods concerned only with planning in static environments, the authors of [75] leverage sequential convex programming to compute trajectories for multiple quadcopters that do not collide, allowing for dynamic formation changes. In [76] a method for dynamic obstacle collision avoidance is presented that models the obstacles as ellipsoids and incorporates them as nonconvex constraints in a model predictive controller.

In this chapter we are interested in reducing the computational time required to find feasible trajectories for quadcopters in order to enable high-speed flight in cluttered, unknown environments (e.g. when navigating a forest). Fast trajectory generation is a requirement in these scenarios due to the limited range of onboard sensors and limited onboard computational power. For example, obstacles can often be occluded or unexpectedly change position, requiring an immediate, agile response to avoid a collision if flying at high speeds. Furthermore, due to the constrained onboard computational power of aerial vehicles, efficient algorithms are often required in order to achieve acceptable performance.

To this end, we propose a computationally efficient method for quickly evaluating whether a candidate trajectory collides with obstacles in the environment. We limit ourselves to evaluating quadcopter trajectories similar to those described in [77], which describe the vehicle's position as a fifth order polynomial in time. These trajectories result in the minimum average jerk over the duration of the trajectory, and are particularly useful because they can be generated and checked for input feasibility with little computation. Unlike other collision detection methods that discretize the trajectory in time and perform a number of static collision checks at each sample point (e.g. as detailed in [78]), our method leverages a continuous-time representation of the trajectory to rapidly perform continuous-time collision detection.

5.2 System model

We follow [77] in modeling the quadcopter as a six degree of freedom rigid body with acceleration $\ddot{\boldsymbol{d}}_{BE}^{E} \in \mathbb{R}^3$ (written in the inertial coordinate frame) and orientation $\boldsymbol{R}^{EB}$, where $\boldsymbol{R}^{EB}$ represents the rotation matrix that rotates vectors in the body-fixed frame to the inertial frame. In contrast to the quadcopter dynamics presented in Chapter 2, we assume that the angular velocity of the vehicle $\boldsymbol{\omega}_{BE}^{B}$ is controlled by a high-bandwidth low-level controller such that the angular velocity converges very rapidly to a desired value, and may thus be treated as an input to the system for path planning purposes. Note that the controller presented in Section 2.4 can still be used to track such reference trajectories with only minor modifications needed to account for the desired feed-forward angular velocity associated with the reference trajectory.

The translational dynamics and simplified attitude dynamics of the quadcopter are then

$$\ddot{\boldsymbol{d}}_{BE}^{E} = \boldsymbol{R}^{EB}\boldsymbol{z}_{B}^{B} f + \boldsymbol{g}^{E}, \qquad \dot{\boldsymbol{R}}^{EB} = \boldsymbol{R}^{EB}\boldsymbol{S}\big(\boldsymbol{\omega}_{BE}^{B}\big) \tag{5.1}$$

where $\boldsymbol{S}\big(\boldsymbol{\omega}_{BE}^{B}\big)$ is the skew-symmetric form of the angular velocity vector $\boldsymbol{\omega}_{BE}^{B}$ and f is the mass-normalized thrust of the quadcopter.

Using this model, it can be shown that kinematically feasible polynomial trajectories in time can be generated using the differential flatness property of quadcopter dynamics [79]. Specifically, we plan trajectories by defining the components of the jerk $\dddot{\boldsymbol{d}}_{BE}^{E}(t)$ as second order polynomials in time between time $t = 0$ and $t = T$. As described in [77], this results in trajectories that minimize the average jerk over the trajectory. The thrust f and angular velocity $\boldsymbol{\omega}_{BE}^{B}$ are then written as a function of $\ddot{\boldsymbol{d}}_{BE}^{E}$ and $\dddot{\boldsymbol{d}}_{BE}^{E}$ as follows.

$$f = m_B||\ddot{\boldsymbol{d}}_{BE}^{E} - \boldsymbol{g}^{E}||_2, \quad \begin{bmatrix}\omega_2 \\ \omega_1 \\ 0\end{bmatrix} = \frac{1}{f}\begin{bmatrix}1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0\end{bmatrix}\boldsymbol{R}^{BE}\dddot{\boldsymbol{d}}_{BE}^{E} \tag{5.2}$$

where ω_1 and ω_2 are the components of angular velocity perpendicular to the thrust direction (i.e. the roll and pitch rates). Note that the angular velocity in the $\boldsymbol{z}_{B}^{B}$ direction does not

affect the translational motion of the vehicle, and is taken to be $\omega_3 = 0$ for the rest of the chapter.

The position and velocity of the quadcopter are defined as $\boldsymbol{d}_{BE}^E$ and $\dot{\boldsymbol{d}}_{BE}^E$ respectively, and are both in $\mathbb{R}^3$ and written in the inertial frame. Let $\boldsymbol{d}_{BE}^E(0)$, $\dot{\boldsymbol{d}}_{BE}^E(0)$, and $\ddot{\boldsymbol{d}}_{BE}^E(0)$ be the position, velocity, and acceleration of the vehicle at the start of the trajectory. Because the minimum average jerk trajectory is achieved when each component of the jerk is a second order polynomial in time, the trajectories of the states of the system follow as

$$\begin{bmatrix} \boldsymbol{d}_{BE}^E(t) \\ \dot{\boldsymbol{d}}_{BE}^E(t) \\ \ddot{\boldsymbol{d}}_{BE}^E(t) \end{bmatrix} = \begin{bmatrix} \frac{\boldsymbol{\alpha}}{120}t^5 + \frac{\boldsymbol{\beta}}{24}t^4 + \frac{\boldsymbol{\gamma}}{6}t^3 + \frac{\ddot{\boldsymbol{d}}_{BE}^E(0)}{2}t^2 + \dot{\boldsymbol{d}}_{BE}^E(0)t + \boldsymbol{d}_{BE}^E(0) \\ \frac{\boldsymbol{\alpha}}{24}t^4 + \frac{\boldsymbol{\beta}}{6}t^3 + \frac{\boldsymbol{\gamma}}{2}t^2 + \ddot{\boldsymbol{d}}_{BE}^E(0)t + \dot{\boldsymbol{d}}_{BE}^E(0) \\ \frac{\boldsymbol{\alpha}}{6}t^3 + \frac{\boldsymbol{\beta}}{2}t^2 + \boldsymbol{\gamma}t + \ddot{\boldsymbol{d}}_{BE}^E(0) \end{bmatrix} \tag{5.3}$$

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma} \in \mathbb{R}^3$ are linear functions of $\boldsymbol{d}_{BE}^E(T)$, $\dot{\boldsymbol{d}}_{BE}^E(T)$, and $\ddot{\boldsymbol{d}}_{BE}^E(T)$.

A method for quickly checking whether a given trajectory satisfies bounds on the minimum and maximum thrust f and the magnitude of the angular velocity $\boldsymbol{\omega}_{BE}^B$ is given in [77], to which we refer the reader for further discussion.

5.3 Algorithm for Static Obstacle Collision Detection

In this section we describe the collision detection algorithm. All obstacles are assumed to be convex; nonconvex obstacles may be approximated by defining them as a union of convex obstacles. In general, the presence of convex obstacles results in the feasible space being nonconvex, making the trajectory generation and collision detection problem difficult to perform using traditional optimization methods.

We first review a method used to check whether a polynomial trajectory lies on one side of a plane, which is defined by a point $\boldsymbol{p}$ and unit normal $\boldsymbol{n}$ (both written in the inertial frame). The distance of the trajectory from the plane can be computed as

$$d(t) = \boldsymbol{n}^T(\boldsymbol{d}_{BE}^E(t) - \boldsymbol{p}) \tag{5.4}$$

Furthermore, the critical points of $d(t)$ can be computed by differentiating with respect to t and finding the roots of the resulting equation:

$$\dot{d}(t) = \boldsymbol{n}^T\dot{\boldsymbol{d}}_{BE}^E(t) = c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0 \tag{5.5}$$

where

$$\begin{gathered} c_4 = \tfrac{1}{24}\boldsymbol{n}^T\boldsymbol{\alpha}, \qquad c_3 = \tfrac{1}{6}\boldsymbol{n}^T\boldsymbol{\beta}, \qquad c_2 = \tfrac{1}{2}\boldsymbol{n}^T\boldsymbol{\gamma} \\ c_1 = \boldsymbol{n}^T\ddot{\boldsymbol{d}}_{BE}^E(0), \qquad c_0 = \boldsymbol{n}^T\dot{\boldsymbol{d}}_{BE}^E(0) \end{gathered} \tag{5.6}$$

The trajectory $\boldsymbol{d}_{BE}^E(t)$ is defined only between $t = 0$ and $t = T$, so the critical points of $d(t)$ occur between and include the start and end of the candidate trajectory. The set of critical points $\mathcal{T}_{\text{crit}}$ is then defined as

$$\mathcal{T}_{\text{crit}} = \{t_i : t_i \in [0, T],\ \dot{d}(t_i) = 0\} \cup \{0, T\} \tag{5.7}$$

If the distance $d(t)$ at each critical point is positive, this indicates that $\boldsymbol{d}_{BE}^{E}(t)$ does not cross the plane. Because $\dot{d}(t)$ is a fourth order polynomial in time, its roots can be found in closed form, meaning that $\mathcal{T}_{\text{crit}}$ can be found with very little computation.

We now extend this method to detect collisions with convex obstacles. The given convex obstacle $\mathcal{O}$ and polynomial trajectory $\boldsymbol{d}_{BE}^{E}(t)$ are required to have the following two properties. First, it must be possible to check whether a specific point $\boldsymbol{d}_{BE}^{E}(t_0)$ is inside $\mathcal{O}$. Second, assuming $\boldsymbol{d}_{BE}^{E}(t_0) \notin \mathcal{O}$, it must be possible to define a separating plane between $\boldsymbol{d}_{BE}^{E}(t_0)$ and $\mathcal{O}$ (defined with point $\boldsymbol{p}$ unit normal $\boldsymbol{n}$). Thus, if $\boldsymbol{d}_{BE}^{E}(t)$ is found to not cross the separating plane, it is guaranteed to not collide with $\mathcal{O}$.

Algorithm 1 leverages this property to verify whether an individual segment of a given trajectory is in collision with a given obstacle. The algorithm begins by checking whether the end points of the trajectory $\boldsymbol{d}_{BE}^{E}(0)$ and $\boldsymbol{d}_{BE}^{E}(T)$ are inside the obstacle (lines 4-5), followed by a call to CheckSection, which returns whether the given section is feasible, infeasible, or whether the feasibility of the section cannot be determined (line 6). For each call to CheckSection(t_s, t_f), a time t_{split} between t_s and t_f is chosen which divides the trajectory in two. We choose t_{split} to be the average of t_s and t_f, as it will evenly divide the trajectory into two sub-trajectories in time (line 8).

For each section checked recursively by CheckSection, $\boldsymbol{d}_{BE}^{E}(t_{\text{split}})$ is first checked for feasibility (line 9), and then the minimum resolution of the section $t_{\min}$ is checked (line 11). The parameter $t_{\min}$ serves to terminate the algorithm early in order to prevent excessive time being spent checking any particular candidate trajectory, and limits the recursive depth of the algorithm. This end condition can be reached in the case where the candidate trajectory passes sufficiently close to the obstacle, requiring the trajectory to be split into a large number of sub-trajectories to be checked.

Next, the unit normal $\boldsymbol{n}$ and location $\boldsymbol{p}$ of a separating plane are found (line 13). Although there are many possible ways to find a separating plane, in our implementation we choose $\boldsymbol{p}$ such that it is the minimum distance point to $\boldsymbol{d}_{BE}^{E}(t_{\text{split}})$ located in $\mathcal{O}$. The unit normal of the plane $\boldsymbol{n}$ is then chosen such that the resulting plane lies on the obstacle boundary at $\boldsymbol{p}$ and points from $\boldsymbol{p}$ to $\boldsymbol{d}_{BE}^{E}(t_{\text{split}})$. The times $\mathcal{T}_{\text{crit}}$ at which the critical points of the distance of the trajectory from the resulting separating plane occur are then computed by solving the corresponding fourth order polynomial given by (5.5) (line 15). Once $\mathcal{T}_{\text{crit}}$ is computed, the two sections of the trajectory occurring before and after t_{split} are each checked for feasibility. Let $\mathcal{T}_{\text{crit}}^{(\downarrow)}$ be the elements of $\mathcal{T}_{\text{crit}}$ in (t_s, t_{split}) and $\mathcal{T}_{\text{crit}}^{(\uparrow)}$ be the elements of $\mathcal{T}_{\text{crit}}$ in (t_{split}, t_f).

The section of the trajectory between t_{split} and t_f is first checked for feasibility by iterating forward in time over $\mathcal{T}_{\text{crit}}^{(\uparrow)}$ and checking whether each critical point of $d(t)$ lies on the feasible side of the separating plane (lines 17-18). If a critical point is found to lie on the obstacle side of the plane, the section between the previous critical point (already determined to be on the feasible side of the plane) and t_f cannot be guaranteed to be feasible and is recursively checked with CheckSection (line 19). Finally, the section of the trajectory between t_s and t_{split} is checked for feasibility in a similar manner by iterating backwards in time over $\mathcal{T}_{\text{crit}}^{(\downarrow)}$ (lines 26-28). A graphical representation of Algorithm 1 is shown in Figure 6.2.

Note that Algorithm 1 treats $\boldsymbol{d}_{BE}^{E}(t)$ as the trajectory of a point. In order to detect collisions between $\mathcal{O}$ and a real quadcopter, we define a sphere of radius r_q that contains the vehicle, and enlarge $\mathcal{O}$ by r_q in each direction. Additionally, because polynomials of order greater than four do not have closed form solutions except in special cases, greater computation time would be required to find the critical points of any higher order position trajectories (e.g. as used in [9]).

Algorithm 1 Trajectory Collision Detection

input: Candidate trajectory parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, initial conditions $\boldsymbol{d}_{BE}^{E}(0)$, $\dot{\boldsymbol{d}}_{BE}^{E}(0)$ $\ddot{\boldsymbol{d}}_{BE}^{E}(0)$, minimum checking time t_{min}, convex obstacle $\mathcal{O}$
output: feasible, infeasible, or indeterminable
function CollisionCheck
    if $\boldsymbol{d}_{BE}^{E}(0)$ or $\boldsymbol{d}_{BE}^{E}(T)$ inside obstacle **then**
        return infeasible
    return CheckSection(0, T)
function CheckSection(t_s, t_f)
    $t_{\text{split}} \leftarrow \frac{t_s+t_f}{2}$
    if $\boldsymbol{d}_{BE}^{E}(t_{\text{split}})$ inside obstacle **then**
        return infeasible
    else if $t_f - t_s < t_{\min}$ **then**
        return indeterminable
    Find plane separating $\boldsymbol{d}_{BE}^{E}(t_{\text{split}})$ and obstacle
    $d(t) \leftarrow$ distance of $\boldsymbol{d}_{BE}^{E}(t)$ from separating plane
    $\mathcal{T}_{\text{crit}}^{(\uparrow)} \leftarrow$ critical points of $d(t)$ from t_{split} to t_f
    Sort $\mathcal{T}_{\text{crit}}^{(\uparrow)}$ ascending
    for t_i in $\mathcal{T}_{\text{crit}}^{(\uparrow)}$, skipping t_{split} **do**
        if $\boldsymbol{d}_{BE}^{E}(t_i)$ is on obstacle side of plane **then**
            result $\leftarrow$ CheckSection(t_{i-1}, t_f)
            if result is feasible **then**
                break
            else
                return result
    $\mathcal{T}_{\text{crit}}^{(\downarrow)} \leftarrow$ critical points of $d(t)$ from t_{split} to t_s
    Sort $\mathcal{T}_{\text{crit}}^{(\downarrow)}$ descending
    for t_i in $\mathcal{T}_{\text{crit}}^{(\downarrow)}$, skipping t_{split} **do**
        if $\boldsymbol{d}_{BE}^{E}(t_i)$ is on obstacle side of plane **then**
            return CheckSection(t_s, t_{i-1})
    return feasible

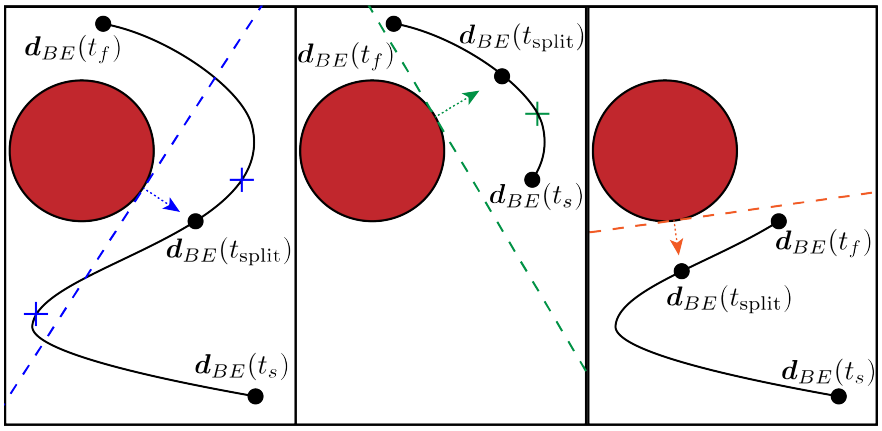

Figure 5.1: A graphical depiction of Algorithm 1. Three sequential calls to CheckSection (as defined in Algorithm 1) are shown. The red circle represents the convex obstacle and the solid black line represents the trajectory in time. In the first call to CheckSection (shown in the left panel), two critical points (drawn as crosses) of the distance to the separating plane (drawn as a dashed line) are found. The trajectory is found to cross the separating plane between t_f and the critical points occurring after t_{split}, prompting a recursive call to CheckSection. As shown in the middle panel, this sub-trajectory is found to not collide with the obstacle because it lies entirely on the opposite side of the newly computed separating plane. Next, the original trajectory (left) is again found to cross the separating plane between t_s and t_{split}, leading to a second recursive call to CheckSection. As shown in the right panel, this sub-trajectory is also found to lie entirely on the opposite side of the newly computed separating plane, proving that the entire trajectory does not collide with the obstacle.

5.4 Performance Measures

In this section we provide two simulations used to benchmark the performance of the proposed algorithm.[1] The algorithm was implemented in C++ and compiled with GCC version 5.4.0 with the highest speed optimization settings. All simulations were ran as a single thread on a laptop with a 1.80GHz Intel i7-8550U processor.

Monte Carlo simulation with random obstacles

First, a Monte Carlo simulation was conducted in order to characterize the computational time required to perform collision detection on a single candidate trajectory. The methods of [77] are used to generate the candidate trajectories and check whether the generated

[1] An implementation can be found at `https://github.com/nlbucki/RapidQuadcopterCollisionDetection`

Table 5.1: Average collision detection time.

	Feasible	Infeasible	Indeterminable
Fraction of trajectories	95.99%	4.01%	$< 0.01\%$
Collision detection time	1.44 µs	1.36 µs	11.59 µs

trajectory satisfies some given input bounds. Any trajectory requiring a mass-normalized thrust that is not between $5\,\mathrm{m\,s^{-2}}$ and $30\,\mathrm{m\,s^{-2}}$ or that requires an angular velocity of greater than $20\,\mathrm{rad\,s^{-1}}$ is discarded.

Candidate trajectories are generated with a fixed initial position of $\boldsymbol{d}_{BE}^{E}(0) = (0, 0, 0)$. The final position, initial and final velocity, and initial and final acceleration along each axis are generated from uniform distributions over the intervals (−4 m, 4 m), ($-4\,\mathrm{m\,s^{-1}}$, $4\,\mathrm{m\,s^{-1}}$), and ($-4\,\mathrm{m\,s^{-2}}$, $4\,\mathrm{m\,s^{-2}}$) respectively. The length of time of the trajectory is sampled uniformly at random between 0.2 s and 4 s. A sphere with radius sampled uniformly at random on (0.1 m, 1.5 m) and positions sampled uniformly at random on (−4 m, 4 m) along each axis is used as an obstacle. The minimum collision detection time per section $t_{\min}$ is chosen to be 2 ms.

For 10^9 such trials, the average time required to detect collisions was 1.44 µs, and of the candidate trajectories, 95.99% did not collide with the obstacle. Table 5.1 shows the computation time required depending on whether the trajectory was found to be feasible, infeasible, or of indeterminable feasibility.

Monte Carlo simulation with constant obstacles

A second Monte Carlo simulation involving generating collision free trajectories that bring the vehicle to rest was also conducted. This scenario is of interest in the case where, for example, the vehicle must perform an emergency stopping maneuver (e.g. when an unexpected obstacle appears in the path of the vehicle while flying at high speed). The simulation is run in batches of 100 candidate trajectories, where each candidate trajectory starts from the same initial state and ends at rest at a position sampled uniformly at random along each axis on (−2.5 m, 2.5 m). For each batch, the initial position of the vehicle is constrained to be (−2.5 m, 0 m, 0 m), the initial velocity and acceleration in the x-direction are sampled uniformly at random on ($2\,\mathrm{m\,s^{-1}}$, $8\,\mathrm{m\,s^{-1}}$) and $4\,\mathrm{m\,s^{-2}}$, $10\,\mathrm{m\,s^{-2}}$) respectively, and the initial velocity and acceleration in the y- and z-directions are sampled uniformly at random on ($-2\,\mathrm{m\,s^{-1}}$, $2\,\mathrm{m\,s^{-1}}$) and ($-2\,\mathrm{m\,s^{-2}}$, $2\,\mathrm{m\,s^{-2}}$) respectively. The length of time of the candidate trajectories is sampled uniformly at random between 0.5 s and 2 s. The positions and orientations of the obstacles, represented as five long rectangular prisms, are fixed as shown in Figure 5.2, which additionally shows the candidate trajectories of a single batch.

One million batches were simulated. The average time required to find the first collision free trajectory was 14.6 µs. On average, each trajectory required 0.1 µs to generate, 0.5 µs to check for satisfaction of constraints on the total thrust and angular velocity, and 7.7 µs to

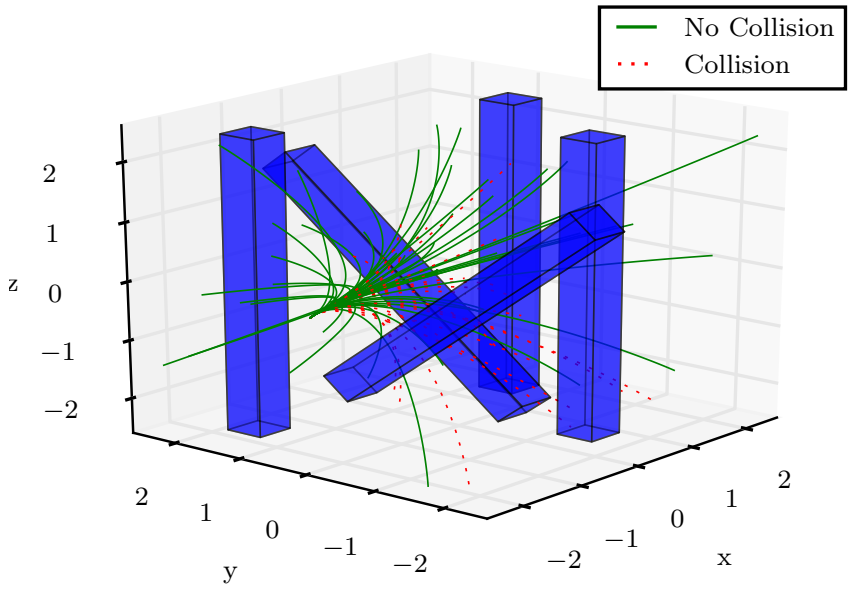

Figure 5.2: Visualization of obstacle distribution and stopping trajectories. Obstacles are represented by blue rectangular prisms. Solid green lines represent the collision free trajectories from a single batch of 100 trajectories, and dotted red lines represent the trajectories that would collide with an obstacle. On average, the first feasible stopping trajectory was found in 14.6 µs.

detect any collisions with the five obstacles. For each batch, an average of 60.2% of generated trajectories were collision free.

5.5 Dynamic Obstacle Collision Detection

In the previous section we showed that our algorithm is capable of detecting collisions between given quadcopter trajectories and static convex obstacles in an environment. This method is easily extended to detect collisions with dynamic obstacles whose boundaries do not rotate, and whose position trajectories are described by fifth order or below polynomial in time. Example applications of this method include detecting collisions between two quadcopters with different trajectories or between a projectile and a moving quadcopter.

Let $\boldsymbol{x}_{\mathcal{O}}(t)$ be the predicted trajectory of the center of a given obstacle. The relative position of the obstacle and the quadcopter at any time t is then

$$\tilde{\boldsymbol{x}}(t) = \boldsymbol{d}_{BE}^{E}(t) - \boldsymbol{x}_{\mathcal{O}}(t) \tag{5.8}$$

where each component of $\tilde{\boldsymbol{x}}(t)$ will a polynomial in time if each component of both $\boldsymbol{d}_{BE}^{E}(t)$ and $\boldsymbol{x}_{\mathcal{O}}(t)$ are also polynomials in time.

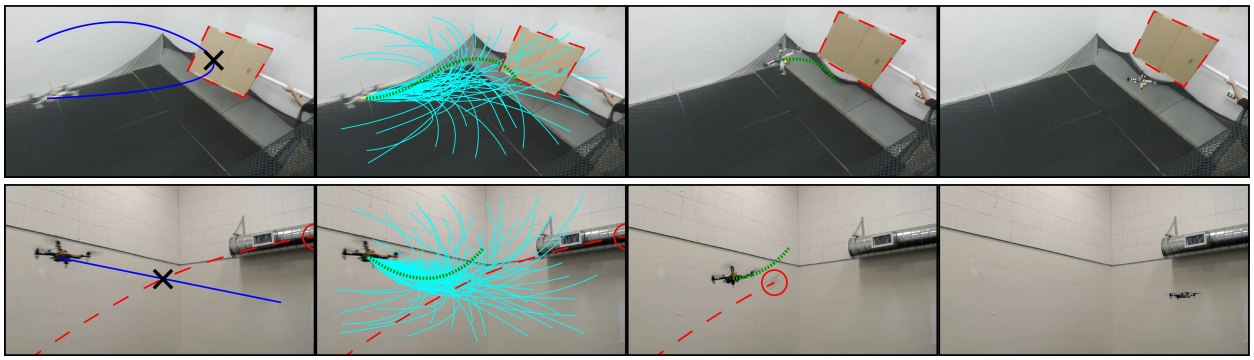

Figure 5.3: A quadcopter avoiding an unexpected surface (top) and a thrown projectile (bottom). The images move forward in time from left to right. The original desired trajectory of the vehicle is shown with a solid blue line, and the position of the surface and predicted trajectory of the projectile are both shown by a dashed red line. In the first frame a collision is predicted to occur if the quadcopter remains on its current trajectory. In the second frame, a large number of alternative trajectories are generated (shown as solid cyan lines). Alternative trajectories that do not satisfy state and input constraints are discarded, and the minimum average jerk trajectory that satisfies all constraints is chosen (shown as a dotted green line). In the experiments shown, 14,610 and 2,371 candidate trajectories were generated and evaluated in 15 ms to avoid the surface and projectile respectively. In the third frame, the avoidance trajectory is tracked while continuing to detect predicted collisions with the obstacle and replanned if necessary. Finally, the fourth frame shows the vehicle successfully coming to rest without colliding with the surface (top), and generating and tracking a trajectory that brings the vehicle to the original desired end position (bottom).

Recall that we model the quadcopter as a sphere with radius r_q. A collision between the quadcopter and the dynamic obstacle occurs if $\tilde{\boldsymbol{x}}(t)$ intersects with an obstacle centered at the origin of the same size as the dynamic obstacle but enlarged by r_q in each direction. Because the boundary of the obstacle is required to not rotate, the same methods described in Section 5.3 may be used to detect collisions between $\tilde{\boldsymbol{x}}(t)$ and the enlarged obstacle centered at the origin. Obstacles with boundaries that do rotate may be straight-forwardly encoded by enclosing them convex shapes with boundaries that do not rotate (e.g. spheres) at the penalty of introducing conservatism to the collision detection.

5.6 Experimental Results

This section presents experimental results where the proposed algorithm is used to enable a quadcopter to avoid unexpected static and dynamic obstacles. For the static obstacle case, we interrupt the motion of the quadcopter while following a trajectory by placing a surface

in the path of the vehicle, forcing it to rapidly plan a new trajectory to avoid the collision and bring the vehicle to rest. For the dynamic obstacle case, we throw a projectile at the vehicle while it is following a trajectory to a goal position, again forcing it to rapidly plan a new trajectory to avoid the projectile and then continue to the original goal position. All experiments can be viewed at `https://youtu.be/cpskvQPhpoY`.

The quadcopter has a mass of 685 g, and receives thrust and angular velocity commands at 50 Hz via radio from an offboard laptop. Collision detection and trajectory generation is performed using the same laptop as used for benchmarking in Section 5.4. The position and attitude of the quadcopter and obstacles are measured using a motion capture system at 200 Hz.

During each controller time step, we check whether any collisions are predicted to occur between the quadcopter and the obstacle used for each experiment. If a collision is detected, a new trajectory is generated that is not predicted to collide with the obstacle and ends at rest with zero velocity and zero acceleration. We sample candidate end positions for the avoidance trajectory uniformly at random in a $3.2\,\mathrm{m} \times 5.2\,\mathrm{m} \times 1\,\mathrm{m}$ rectangular space and sample durations of the avoidance trajectory uniformly at random from 0.5 s to 2 s. While tracking the avoidance trajectory, we continue to check for predicted collisions and generate a new avoidance trajectory if necessary.

When searching for feasible trajectories during both experiments, we generate and evaluate candidate trajectories for 15 ms, and at the end of the allocated time choose the trajectory with the minimum average jerk that satisfies all state and input constraints. We choose the trajectory with the minimum average jerk in order to favor less aggressive trajectories. When evaluating each candidate trajectory, we first compute the average jerk of the trajectory and reject it if it has a higher average jerk than any previously found state and input feasible trajectory. Next, we use the methods of [77] to check that the total mass-normalized thrust f remains between $5\,\mathrm{m\,s^{-2}}$ and $30\,\mathrm{m\,s^{-2}}$ and that the maximum angular velocity remains bellow $20\,\mathrm{rad\,s^{-1}}$, as these are the physical limits of the experimental vehicle. We then check that the candidate trajectory stays within a box of $3.4\,\mathrm{m} \times 5.4\,\mathrm{m} \times 3.1\,\mathrm{m}$ to prevent the vehicle from flying into the ceiling, floor, or walls. Finally, we check that the candidate trajectory does not collide with any obstacles using Algorithm 1 with $t_{\min} = 0.002\,\mathrm{s}$.

Static obstacle avoidance

For the static obstacle avoidance experiment, we define the static obstacle as a rectangular prism measuring $1.64\,\mathrm{m} \times 1.43\,\mathrm{m} \times 0.78\,\mathrm{m}$, which includes both the increase in size in each direction necessary to account for the true size of the quadcopter and a small buffer to account for trajectory tracking and estimation errors. The quadcopter tracks two predefined trajectories that result in a roughly circular motion with an average speed of $5\,\mathrm{m\,s^{-1}}$, and checks these trajectories for collisions with the rectangular prism at each controller time step. The obstacle is then moved by hand in front of the vehicle about 0.5 s before the vehicle would pass, causing a collision to be predicted and an avoidance trajectory to be generated that brings the vehicle to rest without colliding with the obstacle. Figure 5.3 shows a sequence

of images from the experiment. For the experiment shown in Figure 5.3, 14,610 candidate avoidance trajectories were evaluated in the allocated 15 ms after first predicting a collision with the obstacle (recall that the controller runs with a 20 ms period).

Dynamic obstacle avoidance

For the dynamic obstacle avoidance experiment, we throw a projectile at the vehicle while it is performing a rest to rest maneuver from some initial position to some final position $\boldsymbol{p}_f$. If a collision between the quadcopter and the projectile is predicted, the quadcopter rapidly plans a trajectory to avoid the projectile. The projectile is thrown by hand, meaning that its trajectory can only be predicted by the system after it has been thrown. The position $\boldsymbol{x}_p(0)$ and velocity $\dot{\boldsymbol{x}}_p(0)$ of the projectile at the current controller time step are estimated using position measurements received from the motion capture system and a Kalman filter. The position of the projectile $\boldsymbol{x}_p(t)$ is then predicted over a five second time horizon as a quadratic function of time that depends on $\boldsymbol{x}_p(0)$ and $\dot{\boldsymbol{x}}_p(0)$:

$$\boldsymbol{x}_p(t) = \boldsymbol{x}_p(0) + \dot{\boldsymbol{x}}_p(0)t + \frac{1}{2}\boldsymbol{g}^E t^2 \tag{5.9}$$

The minimum allowable distance between the center of mass of the projectile and the center of the quadcopter is chosen to be 40 cm, which is chosen such that there is at least 10 cm separation between the quadcopter and projectile to account for any trajectory tracking and estimation errors. During each controller time step, we check whether the projectile is predicted to collide with the quadcopter by checking whether their relative position $\tilde{\boldsymbol{x}}(t)$ ever enters a sphere of radius 40 cm centered at the origin, and begin generating an avoidance trajectories if a collision is detected. While evaluating candidate avoidance trajectories, we not only check that the avoidance trajectory will not collide with the projectile during the maneuver, but also check that the projectile will not collide with the quadcopter after the quadcopter has reached the end position of the avoidance trajectory.

While tracking the avoidance trajectory, we try to generate sample return trajectories at each controller time step that bring the quadcopter from its current state to rest at the originally desired end position $\boldsymbol{p}_f$. Durations of the candidate return trajectories are sampled between 0.5 s and 4 s. Once a feasible trajectory is found that does not collide with the projectile and ends at $\boldsymbol{p}_f$, the avoidance trajectory is interrupted and the vehicle begins tracking the return trajectory. Figure 5.3 shows a sequence of images from the experiment. For the experiment shown in Figure 5.3, 2,371 candidate avoidance trajectories were evaluated in the allocated 15 ms after first predicting a collision with the projectile.

5.7 Conclusion

In this chapter we presented a method for quickly detecting whether a polynomial trajectory collides with a convex obstacle. This method can be applied to both static convex obstacles

and dynamic obstacles whose boundaries do not rotate. We used the proposed algorithm to perform rapid collision detection of quadcopter trajectories, which can be modeled by fifth order polynomials in time. The ability to rapidly assess whether a given trajectory will collide with obstacles allows for a collision-free trajectory to be found in a short period of time by generating and checking many candidate trajectories for collisions. Because such a large number of the candidate trajectories can be generated and evaluated in such a short period of time, the vehicle is able to plan collision-free trajectories within milliseconds, enabling the vehicle to avoid obstacles that suddenly appear while the vehicle is flying at high speeds.

Furthermore, the computational efficiency of the proposed algorithm does not require the use of expensive or high power computational hardware onboard the vehicle. To this end, the following chapter extends the efficient collision checking methods of this chapter and demonstrates how they can enable a quadcopter to be flown autonomously using only a low-power onboard computer.

Chapter 6

Computationally Efficient Trajectory Generation in Unknown Environments

This chapter extends the ideas presented in the previous chapter in a way that allows for computationally efficient collision checking in previously unseen environments. Unlike the previous chapter where obstacles in the environment were represented as convex shapes at known positions, in this chapter we use depth images to represent the environment. Thus, by using an onboard depth camera to sense the environment, the proposed computationally efficient collision checking algorithm allows for autonomous flight using a low-power onboard computer.

The algorithm presented in the chapter is called RAPPIDS, which is a novel collision checking and planning algorithm for quadcopters that is capable of quickly finding local collision-free trajectories given a single depth image from an onboard camera. This pyramid-based spatial partitioning method enables rapid collision detection between candidate trajectories and the environment. By leveraging the efficiency of our collision checking method, we shown how a local planning algorithm can be run at high rates on computationally constrained hardware, evaluating thousands of candidate trajectories in milliseconds. The performance of the algorithm is compared to existing collision checking methods in simulation, showing our method to be capable of evaluating orders of magnitude more trajectories per second. Experimental results are presented showing a quadcopter quickly navigating a previously unseen cluttered environment by running the algorithm on an ODROID-XU4 at 30 Hz.

Note that the material presented in this chapter is based on the following previously published work, differing primarily in the notation used to describe the dynamics of the vehicle.

- Nathan Bucki, Junseok Lee, and Mark W Mueller. "Rectangular pyramid partitioning using integrated depth sensors (RAPPIDS): A fast planner for multicopter navigation".

In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4626–4633

6.1 Introduction

The ability to perform high-speed flight in cluttered, unknown environments can enable a number of useful tasks, such as the navigation of a vehicle through previously unseen areas and rapid mapping of new environments. Many existing planning algorithms for navigation in unknown environments have been developed for quadcopters, and can generally be classified as map-based algorithms, memoryless algorithms, or a mixture of the two.

Map-based algorithms first fuse sensor data into a map of the surrounding environment, and then perform path planning and collision checking using the stored map. For example, [81] uses a local map to solve a nonconvex optimization problem that returns a smooth trajectory which remains far from obstacles. Similarly, [82, 72, 73, 83] each find a series of convex regions in the free-space of a dynamically updated map, and then use optimization-based methods to find a series of trajectories through the convex regions. Although these methods are generally able to avoid getting stuck in local minima (e.g. a dead end at the end of a hallway), they generally require long computation times to fuse recent sensor data into the global map.

Memoryless algorithms, however, only use the latest sensor measurements for planning. For example, [84] and [71] both use depth images to perform local planning by organizing the most recently received depth data into a k-d tree, which enables the distance from a given point to the nearest obstacle to be quickly computed. The k-d tree is then used to perform collision checking on a number of candidate trajectories, at which point the optimal collision-free candidate trajectory is chosen to track. A different memoryless algorithm is presented in [85] which inflates obstacles in the depth image based on the size of the vehicle, allowing for trajectories to be evaluated directly in image space. In [86], a significant portion of computation is performed offline in order to speed up online collision checking. The space around the vehicle is first split into a grid, a finite set of candidate trajectories are generated, and the grid cells with which each trajectory collides are then computed. Thus, when flying the vehicle, if an obstacle is detected in a given grid cell, the corresponding trajectories can be quickly determined to be in collision. However, such offline methods have the disadvantage of constraining the vehicle to a less expressive set of possible candidate trajectories, e.g. forcing the vehicle to only travel at a single speed.

Several algorithms also leverage previously gathered data while handling the latest sensor measurements separately, allowing for more collision-free trajectories to be found than when using memoryless methods while maintaining a fast planning rate. For example, in [87] a stereo camera pair is used onboard a fixed-wing UAV to detect obstacles at a specific distance in front of the vehicle, allowing for a local map of obstacles to be generated as the vehicle moves forward. In [88] a number of recent depth images are used to find the minimum-uncertainty view of a queried point in space, essentially giving the vehicle a wider field of view for planning. Additionally, in [89] the most recent depth image is both organized into

a k-d tree and fused into a local map, allowing for rapid local planning in conjunction with slower global planning.

Although the previously discussed planning algorithms have been shown to perform well in complex environments, they typically require the use of an onboard computer with processing power roughly equivalent to a modern laptop. This requirement can significantly increase the cost, weight, and power consumption of a vehicle compared to one with more limited computational resources. We address this problem by introducing a novel spatial partitioning and collision checking method to find collision-free trajectories through the environment at low computational cost, enabling rapid path planning on vehicles with significantly lower computational resources than previously developed systems.

The proposed planning algorithm, classified as a memoryless algorithm, takes the latest vehicle state estimate and a single depth image from an onboard camera as input. The depth image is used to generate a number of rectangular pyramids that approximate the free space in the environment. As described in later sections, the use of pyramids in conjunction with a continuous-time representation of the vehicle trajectory allows for any given trajectory to be efficiently labeled as either remaining in the free space, i.e. inside the generated pyramids, or as being potentially in collision with the environment. Thus, a large number of candidate trajectories can be quickly generated and checked for collisions, allowing for the lowest cost trajectory, as specified by a user provided cost function, to be chosen for tracking until the next depth image is captured. Furthermore, by choosing a continuous-time representation of the candidate trajectories, each trajectory can be quickly checked for input feasibility using existing methods.

The use of pyramids to approximate the free space is advantageous because they can be created efficiently by exploiting the structure of a depth image, they can be generated on an as-needed basis (avoiding the up-front computation cost associated with other spatial partitioning data structures such as k-d trees), and because they inherently prevent occluded/unknown space from being marked as free space. Additionally, because our method is a memoryless method rather than a map-based method, it is robust to common mapping errors resulting from poor state estimation (e.g. incorrect loop closures).

6.2 System Model and Relevant Properties

In this section we describe the form of the trajectories used for planning and several of their relevant properties. These properties are later exploited to perform efficient collision checking between the trajectories and the environment.

We assume the vehicle is equipped with sensors capable of producing depth images that can be modeled using the standard pinhole camera model with focal length f. Let the depth-camera-fixed frame C be located at the focal point of the image with z-axis $\boldsymbol{z}_C$ perpendicular to the image plane. The point at position (X, Y, Z) written in the depth-camera-fixed frame is then located $x = f\frac{X}{Z}$ pixels horizontally and $y = f\frac{Y}{Z}$ pixels vertically from the image center with depth value Z.

Trajectory and Collision Model

This chapter uses the same system model and trajectory representation as described in Section 5.2. That is, the desired position trajectory of the vehicle can then be written as

$$\boldsymbol{d}_{BE}^{E}(t) = \frac{\boldsymbol{\alpha}}{120}t^5 + \frac{\boldsymbol{\beta}}{24}t^4 + \frac{\boldsymbol{\gamma}}{6}t^3 + \frac{\ddot{\boldsymbol{d}}_{BE}^{E}(0)}{2}t^2 + \dot{\boldsymbol{d}}_{BE}^{E}(0)t + \boldsymbol{d}_{BE}^{E}(0) \tag{6.1}$$

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma} \in \mathbb{R}^3$ are constants that depend only on $\boldsymbol{d}_{BE}^{E}(T)$, $\dot{\boldsymbol{d}}_{BE}^{E}(T)$, $\ddot{\boldsymbol{d}}_{BE}^{E}(T)$, and T. As stated in the previous chapter, we refer the reader to [77] for details regarding this relation, as well as methods for quickly checking whether constraints on the minimum and maximum thrust and magnitude of the angular velocity of the quadcopter are satisfied throughout the duration of the trajectory. We define $\boldsymbol{d}_{BE}^{E}(t)$ as a collision-free trajectory if a sphere $\mathcal{S}$ centered at $\boldsymbol{d}_{BE}^{E}(t)$ that contains the vehicle does not intersect with any obstacles for the duration of the trajectory.

We additionally define a similar trajectory $\boldsymbol{s}_c(t)$ with initial position $\boldsymbol{s}_c(0)$ coincident with the depth-camera-fixed frame C such that

$$\boldsymbol{s}_c(t) = \frac{\boldsymbol{\alpha}}{120}t^5 + \frac{\boldsymbol{\beta}}{24}t^4 + \frac{\boldsymbol{\gamma}}{6}t^3 + \frac{\ddot{\boldsymbol{d}}_{BE}^{E}(0)}{2}t^2 + \dot{\boldsymbol{d}}_{BE}^{E}(0)t + \boldsymbol{s}_c(0) \tag{6.2}$$

The trajectory $\boldsymbol{s}_c(t)$ is used for collision checking rather than directly using the trajectory of the center of mass $\boldsymbol{d}_{BE}^{E}(t)$ because $\boldsymbol{s}_c(t)$ originates at the focal point of the depth image, allowing for the use of the advantageous properties of $\boldsymbol{s}_c(t)$ described in the following subsections. Let $\mathcal{S}_C$ be a sphere centered at $\boldsymbol{s}_c(t)$ with radius r that contains the sphere $\mathcal{S}$. If the larger sphere $\mathcal{S}_C$ does not intersect with any obstacles for the duration of the trajectory, we can then also verify that the sphere containing the vehicle $\mathcal{S}$ remains collision-free. Thus, we can use $\boldsymbol{s}_c(t)$ and its advantageous properties to check if $\boldsymbol{d}_{BE}^{E}(t)$ is collision-free at the expense of a small amount of conservativeness related to the difference in size between the outer sphere $\mathcal{S}_C$ and inner sphere $\mathcal{S}$.

Trajectory sections with monotonically changing depth

We split a given trajectory, e.g. $\boldsymbol{s}_c(t)$, into different sections with monotonically increasing or decreasing distance along the z-axis $\boldsymbol{z}_C$ of the depth-camera-fixed frame C (i.e. into the depth image) as follows. First, we compute the rate of change of $\boldsymbol{s}_c(t)$ along $\boldsymbol{z}_C$ as $\dot{d}_z(t) = \boldsymbol{z}_C \cdot \dot{\boldsymbol{s}}_c(t)$. Then, by solving $\dot{d}_z(t) = 0$ for $t \in [0, T]$, we can find points $\mathcal{T}_\mathrm{z}$ at which $\boldsymbol{s}_c(t)$ might change direction along $\boldsymbol{z}_C$, defined as

$$\mathcal{T}_\mathrm{z} = \{t_i : t_i \in [0, T],\ \dot{d}_z(t_i) = 0\} \cup \{0, T\} \tag{6.3}$$

Note $\mathcal{T}_\mathrm{z}$ can be computed in closed-form because it only requires finding the roots of the fourth order polynomial $\dot{d}_z(t) = 0$.

Splitting the trajectory into these monotonic sections is advantageous for collision checking because we can compute the point of each monotonic section with the deepest depth from the camera by simply evaluating the endpoints of the section. Thus, we can avoid performing collision checking with any obstacles at a deeper depth than the deepest point of the trajectory.

Trajectory-Plane Intersections

A similar method can be used to quickly determine if and/or when a given trajectory intersects with an arbitrary plane defined by a point $\boldsymbol{p} \in \mathbb{R}^3$ and unit normal $\boldsymbol{n} \in \mathbb{R}^3$. Let the distance of the trajectory from the plane be written as $d(t) = \boldsymbol{n} \cdot (\boldsymbol{s}_c(t) - \boldsymbol{p})$. The set of times $\mathcal{T}_{\text{cross}}$ at which $\boldsymbol{s}_c(t)$ intersects the given plane are then defined as

$$\mathcal{T}_{\text{cross}} = \{t_i : t_i \in [0, T],\ d(t_i) = 0\} \tag{6.4}$$

requiring the solution of the equation $d(t) = 0$. Unfortunately, $d(t)$ is in general a fifth order polynomial, meaning that its roots cannot be computed in closed-form and require more computationally expensive methods to find.

To this end, we extend [77] and [66] in presenting the conditions under which finding $\mathcal{T}_{\text{cross}}$ can be reduced to finding the roots of a fourth order polynomial. Specifically, if a single crossing time of $d(t)$ is known a priori, $d(t) = 0$ can be solved by factoring out the known root and solving the remaining fourth order polynomial. This property is satisfied, for example, by any plane with $\boldsymbol{p} := \boldsymbol{s}_c(0)$ (i.e. a plane that intersects the initial position of the trajectory), resulting in the following equation for $d(t)$:

$$d(t) = \boldsymbol{n} \cdot \left(\frac{\boldsymbol{\alpha}}{120} t^4 + \frac{\boldsymbol{\beta}}{24} t^3 + \frac{\boldsymbol{\gamma}}{6} t^2 + \frac{\ddot{\boldsymbol{d}}^E_{BE}(0)}{2} t + \dot{\boldsymbol{d}}^E_{BE}(0) \right) t \tag{6.5}$$

Thus, the remaining four unknown roots of (6.5) can be computed using the closed-form equation for the roots of a fourth order polynomial, allowing for $\mathcal{T}_{\text{cross}}$ to be computed extremely quickly. As described in the following section, we exploit this property in order to quickly determine when a given trajectory leaves a given pyramid.

6.3 Algorithm Description

In this section we describe the our novel pyramid-based spatial partitioning method, its use in performing efficient collision checking, and the algorithm used to search for the best collision-free trajectory.[1]

[1] An implementation of the algorithm can be found at `https://github.com/nlbucki/RAPPIDS`

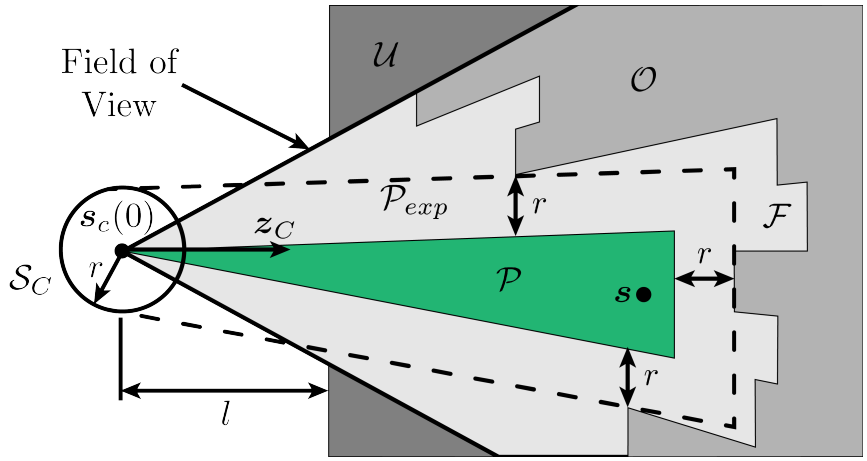

Figure 6.1: 2D side view illustrating the generation of a single pyramid $\mathcal{P}$, shown in green, from a single depth image and given point $\boldsymbol{s}$. The depth values of each pixel are used to define the division between free space $\mathcal{F}$ and occupied space $\mathcal{O}$. Because the depth camera has a limited field of view, we additionally consider any space outside the field of view farther than distance l from the camera to be unknown space $\mathcal{U}$, which is treated the same as occupied space. The expanded pyramid $\mathcal{P}_{exp}$ (dash-dotted line) is first generated such that it does not contain any portion of $\mathcal{O}$ or $\mathcal{U}$, and then used to define pyramid $\mathcal{P}$ such that it is distance r from any obstacles.

Pyramid generation

For each depth image generated by the vehicle's depth sensor, we partition the free space of the environment using a number of rectangular pyramids, where the apex of each pyramid is located at the origin of the depth camera-fixed frame C (i.e. at $\boldsymbol{s}_c(0)$), and the base of each pyramid is located at some depth Z and is perpendicular to the z-axis of the depth camera-fixed frame $\boldsymbol{z}_C$ as shown in Figure 6.1.

The depth value stored in each pixel of the image is used to define the separation of free space $\mathcal{F}$ and occupied space $\mathcal{O}$. We additionally treat the space $\mathcal{U}$ located outside the field of view of the camera at depth l from the camera focal point as occupied space. Pyramid $\mathcal{P}$ is defined such that while trajectory $\boldsymbol{s}_c(t)$ remains inside $\mathcal{P}$, the sphere containing the vehicle $\mathcal{S}_C$ will not intersect with any occupied space, meaning that the segment of $\boldsymbol{s}_c(t)$ inside $\mathcal{P}$ can be considered collision-free.

Note that if $\boldsymbol{s}_c(t)$ remains inside the pyramid, we can not only guarantee that the vehicle will not collide with any obstacles detected by the depth camera, but that the vehicle will not collide with any occluded obstacles either. This differs from other methods that treat each pixel in the depth image as an individual obstacle to be avoided, which can result in the generation of over-optimistic trajectories that may collide with unseen obstacles. Furthermore, our method straightforwardly incorporates field of view constraints, allowing

for the avoidance of nearly all unseen obstacles in addition to those detected by the depth camera.

The function InflatePyramid is used to generate a pyramid $\mathcal{P}$ by taking an initial point $\boldsymbol{s}$ as input and returning either a pyramid containing $\boldsymbol{s}$ or a failure indicator. In this work we choose $\boldsymbol{s}$ to be the endpoint of a given trajectory that we wish to check for collisions, and only generate a new pyramid if $\boldsymbol{s}$ is not already contained in an existing pyramid (details are provided in the following subsection). We start by projecting $\boldsymbol{s}$ into the depth image and finding the nearest pixel p. Then, pixels of the image are read in a spiral about pixel p in order to compute the largest possible expanded rectangle $\mathcal{P}_{exp}$ that does not contain any occupied space. Finally, pyramid $\mathcal{P}$ is computed by shrinking the expanded pyramid $\mathcal{P}_{exp}$ such that each face of $\mathcal{P}$ is not within vehicle radius r of occupied space. Further details regarding our implementation of InflatePyramid can be found online.[1]

This method additionally allows for pyramid generation failure indicators to be returned extremely quickly. For example, consider the case where the initial point $\boldsymbol{s}$ exists inside occupied space $\mathcal{O}$. Then, only the depth value of the nearest pixel p must be read before finding that no pyramid containing $\boldsymbol{s}$ can be generated, requiring only a single pixel of the depth image to be processed. This property greatly reduces the number of operations required to determine when a given point is in collision with the environment.

Collision checking using pyramids

Algorithm 2 describes how the set of all previously generated pyramids $\mathcal{G}$ is used to determine whether a given trajectory $\boldsymbol{s}_c(t)$ will collide with the environment. A trajectory is considered collision-free if it remains inside the space covered by $\mathcal{G}$ for the full duration of the trajectory. An example of Algorithm 2 is shown in Figure 6.2.

We first split the trajectory $\boldsymbol{s}_c(t)$ into sections with monotonically changing depth as described in Section 6.2, and insert the sections into list $\mathcal{M}$ using GetMonotonicSections (line 4). Then, for each monotonic section $\bar{\boldsymbol{s}}_c(t)$, we compute the deepest point $\bar{\boldsymbol{s}}$ (i.e. one of the endpoints of the section), and try to find a pyramid containing that point (line 6-8). The function FindContainingPyramid (line 8) returns either a pyramid that contains $\bar{\boldsymbol{s}}$ or null, indicating no pyramid containing $\bar{\boldsymbol{s}}$ was found. If no pyramid in $\mathcal{G}$ contains $\bar{\boldsymbol{s}}$, we attempt to generate a new pyramid using the method described in the previous subsection (line 10), but if pyramid generation fails we declare the trajectory to be in collision (line 12).

Next, we try to compute the deepest point at which the monotonic section $\bar{\boldsymbol{s}}_c(t)$ intersects one of the four lateral faces of the pyramid $\mathcal{P}$. Using the method described in Section 6.2, we compute the times at which $\bar{\boldsymbol{s}}_c(t)$ intersects each lateral face of the pyramid, and choose the time $t^{\downarrow}$ at which $\bar{\boldsymbol{s}}_c(t)$ has the greatest depth (line 14). If $\bar{\boldsymbol{s}}_c(t)$ is found to not collide with any of the lateral faces of the pyramid, then it necessarily must remain inside the pyramid and the section can be declared collision-free. However, if $\bar{\boldsymbol{s}}_c(t)$ does collide with one of the lateral faces of the pyramid, we split it at $t^{\downarrow}$ and add the section of $\bar{\boldsymbol{s}}_c(t)$ that is outside of the pyramid to $\mathcal{M}$ (line 16). Thus, if each subsection of the trajectory is found to be inside

Algorithm 2 Single Trajectory Collision Checking

1: **input:** Trajectory $\boldsymbol{s}_c(t)$ to be checked for collisions, set of all previously found pyramids $\mathcal{G}$, depth image $\mathcal{D}$
2: **output:** Boolean indicating if trajectory is collision-free, updated set of pyramids $\mathcal{G}$
3: **function** IsCollisionFree($\boldsymbol{s}_c(t)$, $\mathcal{G}$, $\mathcal{D}$)
4: $\quad\mathcal{M} \leftarrow$ GetMonotonicSections($\boldsymbol{s}_c(t)$)
5: $\quad$**while** $\mathcal{M}$ is not empty **do**
6: $\quad\quad\bar{\boldsymbol{s}}_c(t) \leftarrow$ pop($\mathcal{M}$)
7: $\quad\quad\bar{\boldsymbol{s}} \leftarrow$ DeepestPoint($\bar{\boldsymbol{s}}_c(t)$)
8: $\quad\quad\mathcal{P} \leftarrow$ FindContainingPyramid($\mathcal{G}, \bar{\boldsymbol{s}}$)
9: $\quad\quad$**if** $\mathcal{P}$ is null **then**
10: $\quad\quad\quad\mathcal{P} \leftarrow$ InflatePyramid($\bar{\boldsymbol{s}}, \mathcal{D}$)
11: $\quad\quad\quad$**if** $\mathcal{P}$ is null **then**
12: $\quad\quad\quad\quad$**return** false
13: $\quad\quad\quad$push($\mathcal{P}$) $\rightarrow \mathcal{G}$
14: $\quad\quad t^{\downarrow} \leftarrow$ FindDeepestCollisionTime($\mathcal{P}, \bar{\boldsymbol{s}}_c(t)$)
15: $\quad\quad$**if** $t^{\downarrow}$ is not null **then**
16: $\quad\quad\quad$push(GetSubsection($\bar{\boldsymbol{s}}_c(t), t^{\downarrow}$)) $\rightarrow \mathcal{M}$
17: $\quad$**return** true

the space covered by the set of pyramids $\mathcal{G}$, then the trajectory is declared collision-free (line 17).

Note that this method of collision checking allows for pyramids to be generated on an as-needed basis rather than requiring all pyramids to be generated in a batch process when a new depth image arrives. This additionally avoids generating unneeded pyramids; only those required for collision checking are created.

Planning algorithm

Algorithm 3 describes the path planning algorithm used in this work. The algorithm takes as input the most recently received depth image and vehicle state estimate, where the state estimate partially defines each candidate trajectory as given in (6.2). Within a user-specified time budget, the algorithm repeatedly generates and evaluates candidate trajectories for cost and the satisfaction of constraints, returning the lowest cost trajectory that satisfies all constraints. We choose to use a random search algorithm due to its simplicity and probabilistic optimality, though the collision checking algorithm presented in the previous subsection can be used in conjunction with other planning algorithms as well (see [90], for example).

Let GetNextCandidateTraj be defined as a function that returns a randomly generated candidate trajectory $\boldsymbol{s}_c(t)$ using the methods described in [77] (line 7). The function Cost is a user-specified function used to compare candidate trajectories (line 8). In this

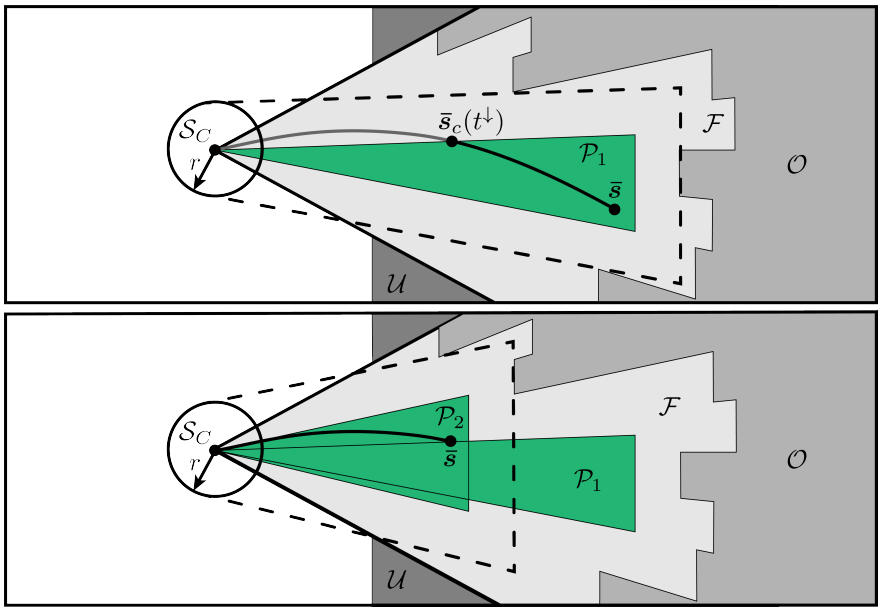

Figure 6.2: 2D example of the collision checking method described by Algorithm 2 as used to check a given trajectory for collisions. The trajectory is first split into sections with monotonically changing depth, which are stored in list $\mathcal{M}$. Top: A single trajectory section $\bar{\boldsymbol{s}}_c(t)$ is chosen from list $\mathcal{M}$. The deepest point of the trajectorysection $\bar{\boldsymbol{s}}$ is computed and pyramid $\mathcal{P}_1$ containing $\bar{\boldsymbol{s}}$ is generated. The trajectory $\bar{\boldsymbol{s}}_c(t)$ is then subdivided into a section that remains inside the pyramid (black) and a section that leaves the pyramid (gray). Bottom: The trajectory section that leaves $\mathcal{P}_1$ is checked for collisions in the same manner. Pyramid $\mathcal{P}_2$ is generated using the deepest point of the trajectory section, and then used to verify that the trajectory section does not collide with the environment.

work, we define Cost to be the following, where $\boldsymbol{d}$ is a desired exploration direction:

$$\textsc{Cost}(\boldsymbol{s}_c(t)) = \frac{\boldsymbol{d} \cdot (\boldsymbol{s}_c(0) - \boldsymbol{s}_c(T))}{T} \tag{6.6}$$

That is, better trajectories are those that cause the vehicle to move quickly in the desired direction $\boldsymbol{d}$. Note, however, that Cost can be defined arbitrarily by the user to include other objectives based on the desired behavior of the vehicle (e.g. to favor increased distance to other vehicles or people).

The function IsDynamicallyFeas (line 9) checks whether the given candidate trajectory satisfies constraints on the total thrust and angular velocity of the vehicle using methods described in [77]. Finally, the candidate trajectory is checked for collisions with the environment using IsCollisionFree (line 10). We check for collisions with the environment last because it is the most computationally demanding step of the process.

Algorithm 3 Lowest Cost Trajectory Search

input: Latest depth image $\mathcal{D}$ and vehicle state
output: Lowest cost collision-free trajectory found $\boldsymbol{s}_c^*(t)$ or an undefined trajectory (indicating failure)
function FindLowestCostTrajectory()
    $\boldsymbol{s}_c^*(t) \leftarrow$ undefined with $\text{Cost}(\boldsymbol{s}_c^*(t)) = \infty$
    $\mathcal{G} \leftarrow \emptyset$
    while computation time not exceeded **do**
        $\boldsymbol{s}_c(t) \leftarrow$ GetNextCandidateTraj()
        if $\text{Cost}(\boldsymbol{s}_c(t)) < \text{Cost}(\boldsymbol{s}_c^*(t))$ **then**
            if IsDynamicallyFeas($\boldsymbol{s}_c(t)$) **then**
                if IsCollisionFree($\boldsymbol{s}_c(t)$, $\mathcal{G}$, $\mathcal{D}$) **then**
                    $\boldsymbol{s}_c^*(t) \leftarrow \boldsymbol{s}_c(t)$
    return $\boldsymbol{s}_c^*(t)$

In this way, Algorithm 3 can be used as a high-rate local planner that ensures the vehicle avoids obstacles, while a global planner that may require significantly more computation time can be used to specify high-level goals (e.g. the exploration direction $\boldsymbol{d}$) without the need to worry about obstacle avoidance or respecting the dynamics of the vehicle. We run Algorithm 3 in a receding-horizon fashion, where each new depth image is used to compute a new collision-free trajectory. We additionally constrain our candidate trajectories to bring the vehicle to rest, so that if no feasible trajectories are found during a given planning step, the last feasible trajectory can be tracked until the vehicle comes to rest.

6.4 Algorithm Performance

In this section we assess the performance of the proposed algorithm in terms of its conservativeness in labeling trajectories as collision-free, its speed, and its ability to evaluate a dense set of candidate trajectories in various amounts of compute time. We additionally compare our method to other state-of-the-art memoryless planning algorithms.

To benchmark our collision checking method, we conduct various Monte Carlo simulations using a series of randomly generated synthetic depth images and vehicle states. Examples of several generated depth images are shown in Figure 6.3. The image is generated by placing two 20 cm thick rectangles with random orientations in front of the camera at distances sampled uniformly at random on (1.5 m, 3 m). Note that this choice of obstacles is arbitrary; any number, distribution, or type of obstacles could be used to conduct such tests. However, rather than trying to emulate a specific type of environment, we choose to use obstacles in our benchmark that are primarily easy to both visualize and reason about conceptually. Furthermore, the use of such obstacles does not unfairly benefit the proposed collision checking method by, e.g., breaking the free-space into regions that may be easier to describe using

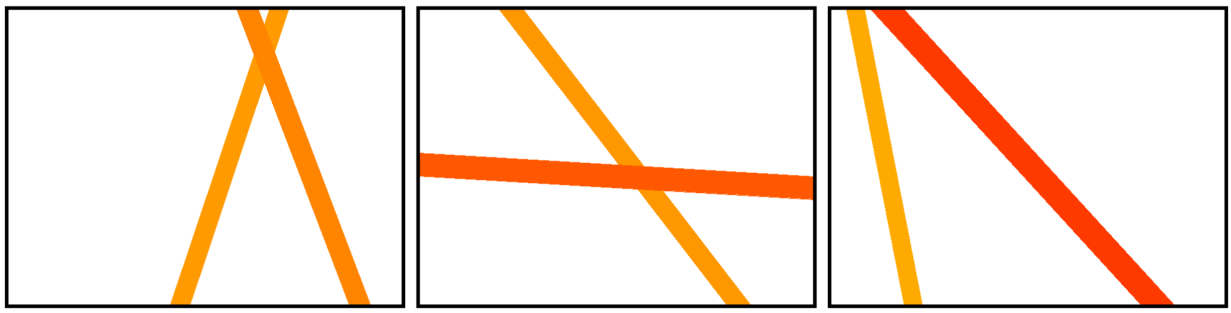

Figure 6.3: Three examples of synthetic depth images used for benchmarking the proposed algorithm. Two rectangular obstacles are generated at different constant depths. The background is considered to be at infinite depth.

pyramids.

The initial velocity of the vehicle in the camera-fixed z-direction $\boldsymbol{z}_C$ is sampled uniformly on $(0\,\mathrm{m\,s^{-1}}, 4\,\mathrm{m\,s^{-1}})$, and the initial velocity of the vehicle in both the x-direction $\boldsymbol{x}_C$ and y-direction $\boldsymbol{y}_C$ is sampled uniformly on $(-1\,\mathrm{m\,s^{-1}}, 1\,\mathrm{m\,s^{-1}})$. We assume the camera is mounted such that $\boldsymbol{z}_C$ is perpendicular to the thrust direction of the vehicle, and thus set the initial acceleration of the vehicle in both the $\boldsymbol{x}_C$ and $\boldsymbol{z}_C$ directions to zero. The initial acceleration in the $\boldsymbol{y}_C$ direction is sampled uniformly on $(-5\,\mathrm{m\,s^{-2}}, 5\,\mathrm{m\,s^{-2}})$.

The planner generates candidate trajectories that come to rest at randomly sampled positions in the field of view of the depth camera. Specifically, positions in the depth image are sampled uniformly in pixel coordinates and then deprojected to a depth that is sampled uniformly on $(1.5\,\mathrm{m}, 3\,\mathrm{m})$. The duration of each trajectory is sampled uniformly on $(2\,\mathrm{s}, 3\,\mathrm{s})$.

The algorithm was implemented in C++ and compiled using GCC version 5.4.0 with the -O3 optimization setting. Three platforms were used to assess performance: a laptop with an Intel i7-8550U processor set to performance mode, a Jetson TX2, and an ODROID-XU4. The algorithm is run as a single thread in all scenarios.

Conservativeness

We first analyze the accuracy of the collision checking method described by Algorithm 2. A key property of our method is that it will never erroneously label a trajectory as collision-free that either collides with an obstacle or has the potential to collide with an occluded obstacle. Such a property is typically a requirement for collision checking algorithms used with aerial vehicles, as a trajectory mislabeled as collision-free can result in a catastrophic crash resulting in a total system failure.

However, because the generated pyramids cannot exactly describe the free space of the environment, our method may erroneously label some collision-free trajectories as being in-collision. In order to quantify this conservativeness, we compare our method to a ground-

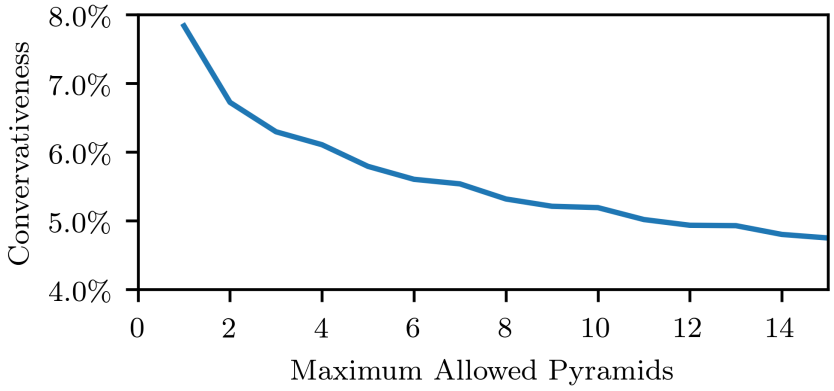

Figure 6.4: Conservativeness of the collision checking algorithm as a function of the maximum number of pyramids allowed to be generated. We define conservativeness as the number of trajectories erroneously labeled as in-collision divided by the total number of trajectories labeled as in-collision. The free-space of the environment is described with increasing detail as more pyramids are allowed to be generated, leading to a lower number of trajectories being erroneously labeled as in-collision.

truth, ray-tracing based collision checking method capable of considering both field-of-view constraints and occluded obstacles. We define conservativeness as the number of trajectories erroneously labeled as in-collision divided by the total number of trajectories labeled (both correctly and incorrectly) as in-collision. A single test consists of first generating a synthetic scene and random initial state of the vehicle as previously described. We then generate 1000 random trajectories for each scene, and perform collision checking both with our method and the ground-truth method. The number of trajectories both correctly and incorrectly labeled as in-collision are averaged over 10^4 such scenes. Additionally, in order to quantify how well the environment can be described using the pyramids generated by our method, we limit the number of pyramids the collision checking algorithm is allowed to use, and repeat this test for various pyramid limits.

Figure 6.4 shows how the over-conservativeness of our method decreases as the number of pyramids allowed to be used for collision checking increases. The percent of mislabeled trajectories is initially higher because the environment cannot be described with high accuracy using fewer pyramids. However, conservativeness remains nearly constant for larger pyramid limits, indicating that our method may erroneously mislabel a small number of collision-free trajectories (e.g. those in close proximity to obstacles). Note that we do not limit the number of pyramids generated when using the planning algorithm described in Algorithm 3, as we have found it to be unnecessary in practice.

Table 6.1: Average Collision Checking Time Per Trajectory

	Computer	Single Trajectory Collision Check (µs)
Florence et al.[2][84]	i7 NUC	56
Lopez et al.[2] [71]	i7-2620M	48
RAPPIDS (ours)	i7-8550U	1.20
RAPPIDS (ours)	Jetson TX2	3.81
RAPPIDS (ours)	ODROID-XU4	8.72

Collision Checking Speed

Next we compare our collision checking method to the state-of-the-art k-d tree based methods described in [84] and [71]. Both our method and k-d tree methods involve two major steps: the building of data structures (i.e. a k-d tree, or the pyramids described in this chapter) and the use of those data structures to perform collision checking with the environment. Our method differs from k-d tree based methods, however, in its over-conservativeness. Specifically, we consider trajectories that pass through occluded space to be in collision with the environment, while k-d tree based methods only consider trajectories that pass within the vehicle radius of detected obstacles to be in collision.

We compare our method to the k-d tree methods by first limiting the amount of time allocated for pyramid generation such that it is similar to the time required to build a k-d tree as reported in [84] and [71] (roughly 1.81 ms). We then check 1000 trajectories for collisions, and compute the average time required to check a single trajectory for collisions using the generated pyramids. Similar to [84] and [71], we use a 160×120 resolution depth image which we generate using the previously described method, and average our results over 10^4 Monte Carlo trails.

Table 6.1 shows how the average performance of our method outperforms the best-case results reported by [84] and [71]. On average 27.5, 19.3, and 15.3 pyramids were generated during the allocated 1.81 ms pyramid generation time on i7, TX2, and ODROID platforms respectively. The difference in computation time can be reasoned about using a time complexity analysis. Let a given depth image contain n pixels. Then $O(n\log(n))$ operations are required to build a k-d tree, while $O(n)$ operations are required to generate a single pyramid (of which there are typically few). Because a single k-d tree query takes $O(\log(n))$ time, if the trajectory must be checked for collisions at m sample points along the trajectory, then the time complexity for checking a single trajectory for collisions is $O(m\log(n))$. However, collision checking a single trajectory using our method can be done in near constant time, as it only requires finding the roots of several fourth order polynomials (which is done in closed-form) as described in Section 6.2. Additionally, note that while an entire k-d tree must be built before being used to check trajectories for collisions, the pyramids generated by our method can be built on an as-needed basis, reducing computation time even further.

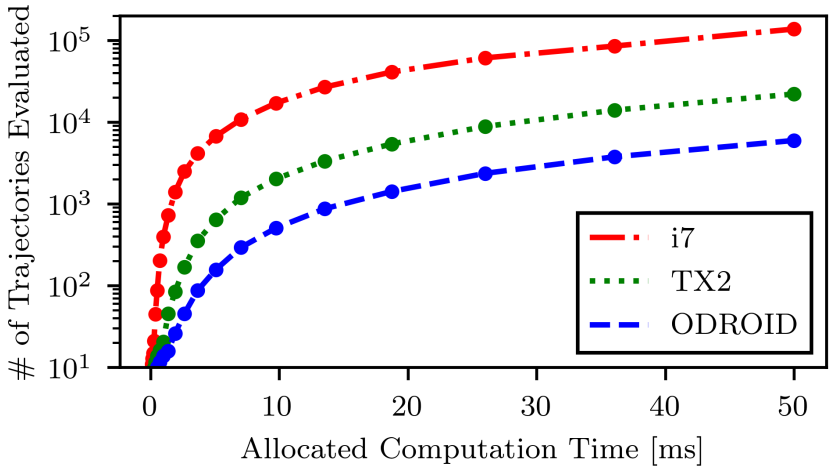

Figure 6.5: Average planner performance as a function of allocated computation time across various platforms. As computation time increases, the number of trajectories evaluated increases at different rates for platforms with different amounts of computation power.

Overall Planner Performance

Finally, we describe the overall performance of the planner, i.e. Algorithm 3, using the same Monte Carlo simulation but with 640×480 resolution depth images, which are the same resolution as those used in the physical experiments described in the following section. The number of trajectories evaluated by the planner is used as a metric of performance, where a larger number of generated trajectories indicates a better coverage of the trajectory search space and thus higher likelihood of finding the lowest possible cost trajectory within the allocated planning time.

Figure 6.5 shows the results of running the planner for 10^4 Monte Carlo trails each on the i7-8550U processor, the Jetson TX2, and the ODROID-XU4 for computation time limits between 0 ms and 50 ms. Naturally, as computation time increases, the average number of trajectories evaluated increases monotonically. Furthermore, we observe that the i7-8550U outperforms the Jetston TX2, which outperforms the ODROID-XU4. The difference in performance can be explained by the fact that the Jetston TX2 and especially the ODROID-XU4 are intended to be low-power devices capable of being used in embedded applications. However, due to the computational efficiency of our collision checking method, we found that even the ODROID-XU4 is capable of evaluating a sufficiently large number of trajectories within a small amount of time. This makes it feasible to use low-power devices such as the ODROID-XU4 as onboard flight controllers while still achieving fast, reactive flight.

[2]The best reported average collision checking time required per trajectory is used for comparison.

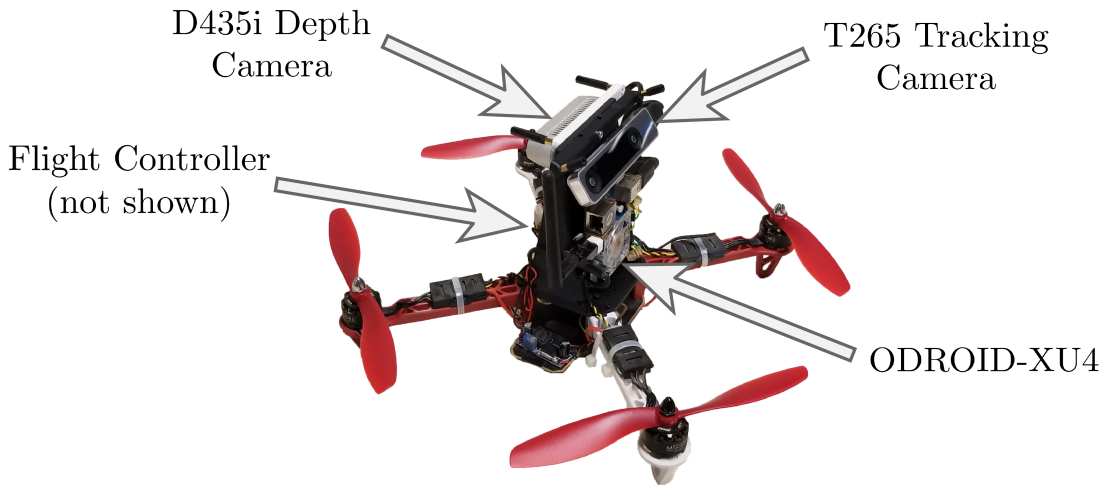

Figure 6.6: Vehicle used in experiments. A RealSense D435i depth camera is used to acquire depth images, and a RealSense T265 tracking camera is used to obtain state estimates of the vehicle. The proposed algorithm is run on an ODROID-XU4, which sends desired thrust and angular velocity commands to a Crazyflie 2.0 flight controller.

6.5 Experimental Results

In this section we demonstrate the use of the proposed algorithm on an experimental quadcopter, shown in Figure 6.6. The quadcopter has a mass of 1.0 kg, and is equipped with an ODROID-XU4, RealSense D435i depth camera, RealSense T265 tracking camera, and Crazyflie 2.0 flight controller. The ODROID is used in order to demonstrate the feasibility of running the proposed algorithm at high rates on cheap, low mass, and low power hardware. The tracking camera provides pose estimates to the ODROID at 200 Hz, which a Kalman filter uses to compute translational velocity estimates. Filtered acceleration estimates are obtained at 100 Hz using the IMU onboard the crazyflie flight controller. The depth camera is configured to capture 640×480 resolution depth images at 30 Hz, and the proposed planning algorithm is run for 30 ms when each new depth image arrives using the latest state estimate provided by the Kalman filter. If no collision-free trajectories can be found during a given planning stage, the vehicle continues to track the most recently found collision-free trajectory from a previous planning stage until either a new collision-free trajectory is found or the vehicle comes to rest.

The vehicle was commanded to fly in a U-shaped tunnel environment that was previously unseen by the vehicle, shown in Figure 6.7. Each branch of the tunnel measured roughly 2.5 m in width and height, 20 m in length, and was filled with various obstacles for the vehicle to avoid. The candidate trajectories generated by the planner were generated using the same

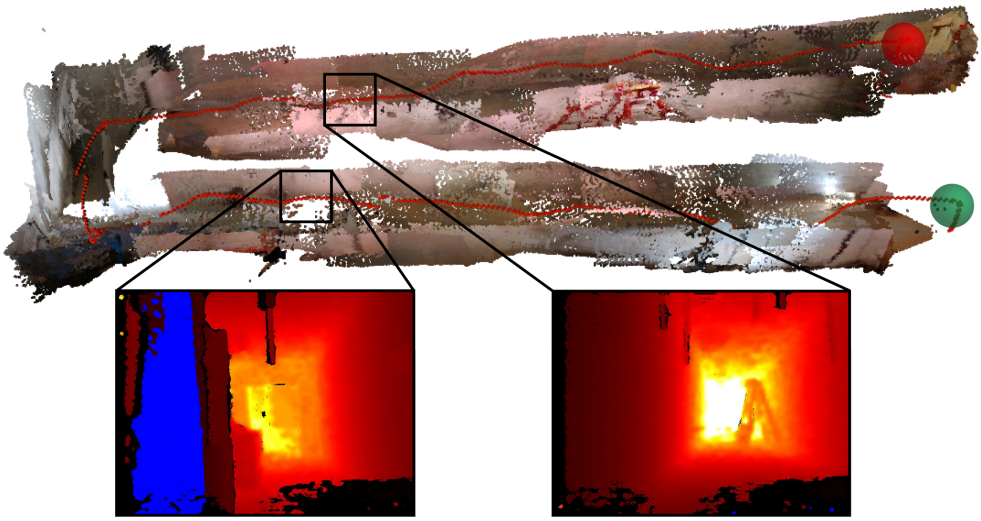

Figure 6.7: Visualization of flight experiment in U-shaped tunnel environment. The path of the vehicle is shown as a red line. The vehicle starts at the green sphere and ends at the red sphere. The map of the environment (top) is generated at the end of the experiment using depth images captured by the depth camera. Two depth images (bottom) where no collision-free trajectories were found are shown to illustrate cases where the planner fails. Pixels with depth values less than 0.75 m but greater than the vehicle radius are highlighted in blue. In the left image, an obstacle occludes a significant portion of the image, preventing collision-free trajectories from being found due to the proximity of the obstacle to the vehicle. In the right image, a very small amount of noise is present near the bottom of the image, causing the planner to hallucinate the presence of close proximity obstacles in what would otherwise be free-space.

method described in Section 6.4. [3]

The desired exploration direction $\boldsymbol{d}$ used to compute the cost of each candidate trajectory as given by (6.6) is set as follows. We initialize $\boldsymbol{d}$ to be horizontal and to point down the first hallway. When the vehicle is at rest and no feasible trajectories are found by the planner, the desired exploration direction $\boldsymbol{d}$ is rotated 90° to the right of the vehicle, allowing the vehicle to navigate around corners. We then stop the test when the vehicle reaches the end of the second hallway. We use this method of choosing the exploration direction simply as a matter of convenience in demonstrating the use of our algorithm in a cluttered environment. However, many other suitable methods of providing high-level goals to our algorithm can be used (e.g. [91]), but are typically application dependent and thus are not discussed here.

During the experiment, the vehicle was able to find at least one collision-free trajectory

[3] A video of the experiment can be viewed at `https://youtu.be/Pp-HIT9S6ao`

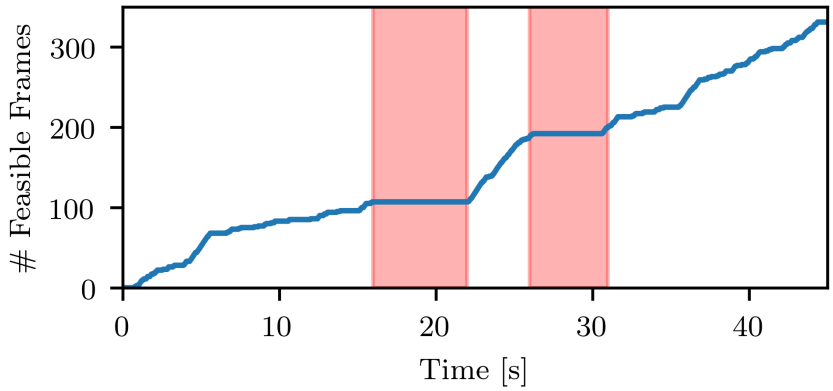

Figure 6.8: Cumulative number of planning stages where at least one collision-free trajectory was found. The sections of the graph highlighted in red correspond to periods in which the vehicle is facing the wall at the end of the hallway. A large increase in successful planning stages is observed between 22 s and 26 s when the vehicle is navigating in the relatively uncluttered area between the two hallways.

in 35.3% of the 30 ms planning stages. Of the planning stages where at least one feasible trajectory was found, 2069.2 candidate trajectories and 2.9 pyramids were generated on average. The vehicle traveled approximately 40 m over 43 s, and attained a maximum speed of 2.66 m s^{-1}.

The low percentage of planning stages where at least one collision-free trajectory was found primarily are cases where the vehicle passes closely to obstacles and also by the significant amount of noise present in the depth images. Figure 6.7 shows examples of both cases. Note that the amount of noise present in the depth images can be reduced via filtering, although this may lead to the potential misdetection of small and/or thin obstacles. Additionally, Figure 6.8 shows how the successful planning stages are distributed throughout the experiment. A lower percent of successful planning stages is observed when the vehicle is navigating the cluttered hallways than the relatively open area between the two hallways, which is potentially due to the difference in obstacle density and lighting conditions (leading to a difference in depth image noise levels).

6.6 Conclusion

In this chapter we presented a novel pyramid-based spatial partitioning method that allows for efficient collision checking between a given trajectory and the environment as represented by a depth image. The method allows quadcopters with limited computational resources to quickly navigate cluttered environments by generating collision-free trajectories at high rates. A comparison to existing state-of-the-art depth-image-based path planning methods was performed via Monte Carlo simulation, showing our method to significantly reduce the

computation time required to perform collision checking with the environment while being more conservative than other methods by implicitly considering occluded obstacles. Finally, real-world experiments were presented that demonstrate the use of our algorithm on computationally low-power hardware to perform fully autonomous flight in a previously unseen, cluttered environment. Similar to the previous chapter, this chapter has demonstrated how improving the computational efficiency of path planning methods used with quadcopters can reduce the hardware requirements of such vehicles.

Chapter 7

Conclusions and Future Work

In this dissertation two novel quadcopter designs were presented which can enhance the operational capabilities of quadcopters, and two novel path planning techniques were presented that reduce the computational requirements of onboard computers needed to fly such vehicles autonomously. These advances were focused on maintaining the aspects of quadcopters that make them useful platforms for accomplishing various tasks (e.g. design simplicity and low cost) while extending their ability to perform new tasks or to operate in new environments. In this chapter we first summarize the individual topics covered in the preceding chapters, and finally conclude with a discussion of the potential extensions of this work in the future.

The first novel quadcopter design focused on reducing the sensitivity of quadcopters to torque disturbances by increasing the net angular momentum of the vehicle using a momentum wheel. It was shown that the sensitivity of the vehicle to torque disturbances monotonically decreases with increasing angular momentum, implying that spinning the momentum wheel faster will strictly improve the torque disturbance rejection capabilities of the vehicle. Furthermore, because this effect scales with angular momentum, a lightweight momentum wheel can achieve the same effect as a heavier momentum wheel by spinning faster at the expense of requiring a higher kinetic energy to be stored in the wheel. Thus, such a vehicle design was shown to allow for quadcopters to operate in environments with large torque disturbances (e.g. hail storms and high wind-shear environments), with only a marginal increase in mass required due to the weight of the momentum wheel.

In the following chapter, a vehicle capable of changing shape mid-flight was presented. Similar to the vehicle using an added momentum wheel, this vehicle was designed to minimize the difference between it and a conventional quadcopter, while allowing novel tasks to be performed outside the abilities of a conventional quadcopter. The only difference between the proposed vehicle design and a conventional quadcopter was the use of passive rotary joints that connect the four arms of the vehicle to the central body. These additional unactuated degrees of freedom where then shown to enable the vehicle to change shape midair, allowing the vehicle to traverse passageways that would otherwise be impassable, as well as perform some basic manipulation and perching tasks. Specifically, the vehicle required no actuators or sensors that were not present on a conventional quadcopter, allowing for a simplistic

design to be used to accomplish several novel tasks.

In next chapters, two similar algorithms were presented that allowed for computationally efficient collision checking to be performed between quadcopter trajectories and environments represented using either convex obstacles or depth images. Due to the computational efficiency of the proposed algorithms, it was shown that a low-power onboard computer can be used to fly a quadcopter through cluttered environments autonomously, reducing the onboard computational hardware requirements of a quadcopter compared to existing methods. The computational efficiency of the proposed algorithms was demonstrated using both Monte Carlo simulations as well as several flight experiments, where the algorithms were shown to be able to control a quadcopter to avoid both static and fast-moving dynamic obstacles. In contrast to the chapters focused on design changes to a conventional quadcopter, these chapters focused on methods that allow for less expensive and lower power computers to be used onboard vehicles while still allowing them to navigate autonomously.

Thus, the previous chapters have shown how the performance and utility of quadcopters can be advanced beyond the current state-of-the-art using both subtle design changes and algorithmic advances. These changes enhance the ability of quadcopters to operate in new environments, namely environments with high torque disturbances, narrow passageways, and environments with both static and dynamic obstacles that are to be avoided at high speeds. Note that although operation in these environments could perhaps be marginally improved via either design changes or control changes alone, in this work we have leveraged the joint development of both novel quadcopter designs and control algorithms together to allow for much more significant improvements upon what conventional quadcopters can achieve.

In conclusion, this dissertation has focused on a variety of ways that the usefulness of quadcopters can be improved in terms of design, control, and path planning. In each case we examined changes that have the potential to improve the functionality of a quadcopter without significantly diminishing the aspects that make a conventional quadcopter useful (e.g. VTOL capability, mechanical simplicity, etc.). Specifically, the use of a momentum wheel was shown to allow a quadcopter to fly in environments with large torque disturbances, the use of passive hinges was shown to allow a quadcopter to traverse narrow gaps and perform grasping and perching tasks, and the use of novel path planning algorithms was shown to allow for low power computers to be used to fly quadcopters autonomously through cluttered environments. Thus, we argue that the utility of quadcopters can be improved using the aforementioned ideas, with only relatively small (or nonexistent) tradeoffs required to achieve such improvements in utility.

7.1 Future Work

Although there are many interesting avenues of research stemming from the topics covered in this dissertation, several particularly interesting topics are included below.

Related to the vehicle with added angular momentum as described in Chapter 3, further investigation is warranted of the type and frequency of disturbances encountered in specific

real-world scenarios (e.g. hail storms, tornadoes, etc.). This type of analysis could help better quantify the gain in utility offered by the use of angular momentum as a disturbance rejection mechanism. Furthermore, the design of a more lightweight momentum wheel that is capable of safely spinning at higher angular velocities could be an interesting design challenge. Because the angular momentum of the wheel scales with the angular velocity of the wheel, in theory the mass of the wheel can be reduced so long as the wheel can be spun at a proportionally higher speed, though (as noted in the chapter) the energy stored in the wheel will increase quadratically with wheel speed.

Relating to the work on the morphing vehicle described in Chapter 4, interesting planning and navigation questions still remain. Because the described vehicle can utilize its morphing capabilities to traverse areas the vehicle would otherwise be unable to traverse in its normal configuration, there is an opportunity to design novel path-planning algorithms that are able to automatically decide when and where the vehicle should change configurations.

Finally, the material related to planning presented in Chapters 5 and 6 could be extended by investigating the usefulness of optimization based path planning methods. While the presented methods use random sampling to find nearly-optimal collision-free trajectories, in principle more sophisticated nonconvex optimization methods could be used to improve the efficiency with which the optimal trajectory is found during a given planning step. Although this would not necessarily improve the ability of the vehicle to avoid collisions, such methods have the potential to either improve the quality of generated trajectories or to reduce the computation time required to generate collision-free trajectories.

In a more broad view, the future development of the quadcopter is likely to focus on improving range and flight time as well. For example, many recent quadcopter-like designs (such as [92]) have focused on allowing for flight in a fixed-wing configuration while still enabling the vertical takeoff and landing capability that makes the quadcopter so useful. These quadcopter-like vehicles are able to fly long distances more efficiently than conventional quadcopter designs by taking advantage of the aerodynamic properties of the vehicle, and could potentially prove more useful than a conventional quadcopter design for certain applications requiring long flight times.

In summary, there are numerous opportunities for improvement upon the conventional quadcopter design and the algorithms used to control them. Although several such design and control changes were presented in this dissertation, the full mechanical and algorithmic design space is yet to be exhaustively explored, and there are likely even more ideas capable of improving the utility of such aerial vehicles in the future.

Bibliography

[1] Sonia Waharte and Niki Trigoni. "Supporting search and rescue operations with UAVs". In: *2010 International Conference on Emerging Security Technologies (EST)*. IEEE. 2010, pp. 142–147.

[2] Teodor Tomic et al. "Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue". In: *IEEE Robotics & Automation Magazine* 19.3 (2012), pp. 46–56.

[3] Huy Xuan Pham et al. "A distributed control framework of multiple unmanned aerial vehicles for dynamic wildfire tracking". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50.4 (2018), pp. 1537–1548.

[4] Youngjun Choi et al. "Multi-UAV trajectory optimization utilizing a NURBS-based terrain model for an aerial imaging mission". In: *Journal of Intelligent & Robotic Systems* 97.1 (2020), pp. 141–154.

[5] Abdul Nishar et al. "Thermal infrared imaging of geothermal environments and by an unmanned aerial vehicle (UAV): A case study of the Wairakei–Tauhara geothermal field, Taupo, New Zealand". In: *Renewable Energy* 86 (2016), pp. 1256–1264.

[6] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. "UAV-based IoT platform: A crowd surveillance use case". In: *IEEE Communications Magazine* 55.2 (2017), pp. 128–134.

[7] Sarra Berrahal et al. "Border surveillance monitoring using quadcopter UAV-aided wireless sensor networks". In: *Journal of Communications Software and Systems* 12.1 (2016), pp. 67–82.

[8] Inkyu Sa and Peter Corke. "Vertical infrastructure inspection using a quadcopter and shared autonomy control". In: *Field and Service Robotics*. Springer. 2014, pp. 219–232.

[9] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments". In: *Robotics Research*. Springer, 2016, pp. 649–666.

[10] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[11] Davide Falanga et al. "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5774–5781.

[12] Inkyu Sa and Peter Corke. "System identification, estimation and control for a cost effective open-source quadcopter". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2202–2209.

[13] Vaibhav Ghadiok, Jeremy Goldin, and Wei Ren. "On the design and development of attitude stabilization, vision-based navigation, and aerial gripping for a low-cost quadrotor". In: *Autonomous Robots* 33.1 (2012), pp. 41–68.

[14] Paul Pounds et al. "Design of a four-rotor aerial robot". In: *Proceedings of the 2002 Australasian Conference on Robotics and Automation (ACRA 2002)*. Australian Robotics & Automation Association. 2002, pp. 145–150.

[15] P. H. Zipfel. *Modeling and Simulation of Aerospace Vehicle Dynamics*. 2nd ed. American Institute of Aeronautics and Astronautics, 2007.

[16] Brian Anderson and John Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall International, 1989.

[17] Nathan Bucki and Mark W Mueller. "Improved Quadcopter Disturbance Rejection Using Added Angular Momentum". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018.

[18] Nathan Bucki and Mark W Mueller. "A novel multicopter with improved torque disturbance rejection through added angular momentum". In: *International Journal of Intelligent Robotics and Applications* 3.2 (2019), pp. 131–143.

[19] Steven Waslander and Carlos Wang. "Wind disturbance estimation and rejection for quadrotor position control". In: *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*. 2009, p. 1983.

[20] Lénaïck Besnard, Yuri B Shtessel, and Brian Landrum. "Quadrotor vehicle control via sliding mode controller driven by sliding mode disturbance observer". In: *Journal of the Franklin Institute* 349.2 (2012), pp. 658–684.

[21] David Cabecinhas, Rita Cunha, and Carlos Silvestre. "A nonlinear quadrotor trajectory tracking controller with disturbance rejection". In: *Control Engineering Practice* 26 (2014), pp. 1–10.

[22] Rongting Zhang, Quan Quan, and K-Y Cai. "Attitude control of a quadrotor aircraft subject to a class of time-varying disturbances". In: *IET control theory & applications* 5.9 (2011), pp. 1140–1146.

[23] Peter W Likins. "Attitude stability criteria for dual spin spacecraft." In: *Journal of Spacecraft and Rockets* 4.12 (1967), pp. 1638–1643.

[24] DL Mingori. "Effects of energy dissipation on the attitude stability of dual-spin satellites". In: *AIAA Journal* 7.1 (1969), pp. 20–27.

[25] Matthew Piccoli and Mark Yim. "Passive stability of a single actuator micro aerial vehicle". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 5510–5515.

[26] Mark W Mueller and Raffaello D'Andrea. "Relaxed hover solutions for multicopters: Application to algorithmic redundancy and novel vehicles". In: *The International Journal of Robotics Research* 35.8 (2016), pp. 873–889.

[27] Weixuan Zhang, Mark W Mueller, and Raffaello D'Andrea. "A controllable flying vehicle with a single moving part". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3275–3281.

[28] Scott Driessens and Paul Pounds. "The triangular quadrotor: a more efficient quadrotor configuration". In: *IEEE Transactions on Robotics* 31.6 (2015), pp. 1517–1526.

[29] Markus Ryll, Heinrich H Bülthoff, and Paolo Robuffo Giordano. "First flight tests for a quadrotor UAV with tilting propellers". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 295–302.

[30] Dustin Alexander Wallace. "Dynamics and Control of a Quadrotor with Active Geometric Morphing". MA thesis. 2016.

[31] Mark Wilfried Mueller and Raffaello D'Andrea. "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[32] Matthew Piccoli and Mark Yim. "Passive stability of vehicles without angular momentum including quadrotors and ornithopters". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1716–1721.

[33] Alex Kushleyev et al. "Towards a swarm of agile micro quadrotors". In: *Autonomous Robots* 35.4 (2013), pp. 287–300.

[34] Michael Green and David Limebeer. *Linear Robust Control*. Dover Publications, 1995.

[35] Barnes W. McCormick. *Aerodynamics Aeronautics and Flight Mechanics*. John Wiley & Sons, Inc, 1995.

[36] Mark Wilfried Mueller. "Multicopter attitude control for recovery from large disturbances". In: *CoRR* abs/1802.09143 (2018). arXiv: `1802.09143`. URL: `http://arxiv.org/abs/1802.09143`.

[37] Arthur Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control*. CRC Press, 1975.

[38] Nathan Bucki and Mark W Mueller. "Design and Control of a Passively Morphing Quadcopter". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9116–9122.

[39] Nathan Bucki, Jerry Tang, and Mark W Mueller. "Design and Control of a Midair Reconfigurable Quadcopter using Unactuated Hinges". In: *arXiv preprint arXiv:2103.16632* (2021).

[40] Amanda Bouman et al. "Design and Autonomous Stabilization of a Ballistically-Launched Multirotor". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 8511–8517.

[41] Stefano Mintchev et al. "Foldable and self-deployable pocket sized quadrotor". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2190–2195.

[42] Na Zhao et al. "The deformable quad-rotor: Design, kinematics and dynamics characterization, and flight performance validation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 2391–2396.

[43] Dangli Yang et al. "Design, Planning, and Control of an Origami-inspired Foldable Quadrotor". In: *2019 American Control Conference (ACC)*. IEEE. 2019, pp. 2551–2556.

[44] A Desbiez et al. "X-Morf: A crash-separable quadrotor that morfs its X-geometry in flight". In: *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. IEEE. 2017, pp. 222–227.

[45] Ye Bai and Srikanth Gururajan. "Evaluation of a Baseline Controller for Autonomous "Figure-8" Flights of a Morphing Geometry Quadcopter: Flight Performance". In: *Drones* 3.3 (2019), p. 70.

[46] Davide Falanga et al. "The foldable drone: A morphing quadrotor that can squeeze and fly". In: *IEEE Robotics and Automation Letters* 4.2 (2018), pp. 209–216.

[47] Amedeo Fabris et al. "Geometry-aware Compensation Scheme for Morphing Drones". In: *arXiv preprint arXiv:2003.03929* (2020).

[48] Giovanny Ortega Vargas et al. "Dynamic modeling of a multi-rotorcraft uas with morphing capabilities". In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2015, pp. 963–971.

[49] Akinori Sakaguchi, Takashi Takimoto, and Toshimitsu Ushio. "A Novel Quadcopter with A Tilting Frame using Parallel Link Mechanism". In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2019, pp. 674–683.

[50] Valentin Riviere, Augustin Manecy, and Stéphane Viollet. "Agile robotic fliers: A morphing-based approach". In: *soft robotics* 5.5 (2018), pp. 541–553.

[51] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. "Construction with quadrotor teams". In: *Autonomous Robots* 33.3 (2012), pp. 323–336.

[52] Justin Thomas et al. "Avian-inspired grasping for quadrotor micro UAVs". In: *ASME 2013 international design engineering technical conferences and computers and information in engineering conference*. American Society of Mechanical Engineers Digital Collection. 2013.

[53] Matko Orsag et al. "Stability control in aerial manipulation". In: *2013 American Control Conference*. IEEE. 2013, pp. 5581–5586.

[54] Christopher Korpela, Matko Orsag, and Paul Oh. "Towards valve turning using a dual-arm aerial manipulator". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 3411–3416.

[55] Tomoki Anzai et al. "Aerial grasping based on shape adaptive transformation by halo: horizontal plane transformable aerial robot with closed-loop multilinks structure". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6990–6996.

[56] Daniel Mellinger et al. "Design, modeling, estimation and control for aerial grasping and manipulation". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 2668–2673.

[57] Xiangdong Meng, Yuqing He, and Jianda Han. "Survey on Aerial Manipulator: System, Modeling, and Control". In: *Robotica* (), pp. 1–30.

[58] Hossein Bonyan Khamseh, Farrokh Janabi-Sharifi, and Abdelkader Abdessameud. "Aerial manipulation - A literature survey". In: *Robotics and Autonomous Systems* 107 (2018), pp. 221–235.

[59] Elliot W Hawkes et al. "Dynamic surface grasping with directional adhesion". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 5487–5493.

[60] Arash Kalantari et al. "Autonomous perching and take-off on vertical walls for a quadrotor micro air vehicle". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 4669–4674.

[61] Katie M Popek et al. "Autonomous Grasping Robotic Aerial System for Perching (AGRASP)". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.

[62] Kaiyu Hang et al. "Perching and resting - A paradigm for UAV maneuvering with modularized landing gears". In: *Science Robotics* 4.28 (2019), eaau6637.

[63] Courtney E Doyle et al. "An avian-inspired passive mechanism for quadrotor perching". In: *IEEE/ASME Transactions On Mechatronics* 18.2 (2012), pp. 506–517.

[64] Matthias Faessler, Davide Falanga, and Davide Scaramuzza. "Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 476–482.

[65] Ingo Schiffner et al. "Minding the gap: in-flight body awareness in birds". In: *Frontiers in zoology* 11.1 (2014), p. 64.

[66] Nathan Bucki and Mark W Mueller. "Rapid Collision Detection for Multicopter Trajectories". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

[67] Steven LaValle. "Rapidly-exploring random trees: A new tool for path planning". In: (1998).

[68] Lydia Kavraki et al. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Vol. 12. International Transactions on Robotics and Automation, 1994.

[69] Lucas Janson et al. "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions". In: *The International Journal of Robotics Research* 34.7 (2015), pp. 883–921.

[70] Joshua Bialkowski et al. "Efficient collision checking in sampling-based motion planning via safety certificates". In: *The International Journal of Robotics Research* 35.7 (2016), pp. 767–796.

[71] Brett T Lopez and Jonathan P How. "Aggressive collision avoidance with limited field-of-view sensing". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1358–1365.

[72] Sikang Liu et al. "High speed navigation for quadrotors with limited onboard sensing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1484–1491.

[73] Jing Chen, Tianbo Liu, and Shaojie Shen. "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1476–1483.

[74] Ji Zhang et al. "P-CAP: Pre-computed Alternative Paths to Enable Aggressive Aerial Maneuvers in Cluttered Environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 8456–8463.

[75] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2012, pp. 1917–1922.

[76] Tobias Nägeli et al. "Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization". In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1696–1703.

[77] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. "A computationally efficient motion primitive for quadrocopter trajectory generation". In: *IEEE Transactions on Robotics* 31.6 (2015), pp. 1294–1310.

[78] Steven LaValle. *Planning algorithms*. Cambridge university press, 2006.

[79] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 2520–2525.

[80] Nathan Bucki, Junseok Lee, and Mark W Mueller. "Rectangular pyramid partitioning using integrated depth sensors (RAPPIDS): A fast planner for multicopter navigation". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4626–4633.

[81] Helen Oleynikova et al. "Continuous-time trajectory optimization for online UAV replanning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 5332–5339.

[82] Fei Gao, Yi Lin, and Shaojie Shen. "Gradient-based online safe trajectory generation for quadrotor flight in complex environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3681–3688.

[83] Jesus Tordesillas, Brett T Lopez, and Jonathan P How. "FaSTraP: Fast and Safe Trajectory Planner for Flights in Unknown Environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

[84] Pete Florence, John Carter, and Russ Tedrake. "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps". In: *Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 2016.

[85] Larry Matthies et al. "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3242–3249.

[86] Ji Zhang et al. "Maximum Likelihood Path Planning for Fast Aerial Maneuvers and Collision Avoidance". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

[87] Andrew J Barry and Russ Tedrake. "Pushbroom stereo for high-speed navigation in cluttered environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3046–3052.

[88] Peter R Florence et al. "Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7631–7638.

[89] Markus Ryll et al. "Efficient Trajectory Planning for High Speed Flight in Unknown Environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 732–738.

[90] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894.

[91] Titus Cieslewski, Elia Kaufmann, and Davide Scaramuzza. "Rapid exploration with multi-rotors: A frontier selection method for high speed flight". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 2135–2142.

[92] Ximin Lyu et al. "Design and implementation of a quadrotor tail-sitter vtol uav". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3924–3930.