STIPPLING WITH QUADROTORS

Brendan Galea

Master of Science

School of Computer Science

McGill University

Montreal,Quebec

2016-08-08

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of master of science.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Professor Paul Kry for his support, guidance, and valued feedback. I would also like to thank Ehsan Kia for his contribution towards determining stipple order and dynamic updates, and Nicholas Aird for his initial work which provided the starting point for this project.

ABSTRACT

We describe a method for creating stippled prints using a quadrotor flying robot. At a low level, we use motion capture to measure the position of the robot and the canvas, and a robust control algorithm to command the robot to fly to different stipple positions to make contact with the canvas using an ink soaked sponge. We describe a collection of important details and challenges that must be addressed for successful control in our implementation, including robot model estimation, Kalman filtering for state estimation, latency between motion capture and control, radio communication interference, and control parameter tuning. We use a centroidal Voronoi diagram to generate stipple drawings, and compute a greedy approximation of the traveling salesman problem to draw as many stipples per flight as possible, while accounting for desired stipple size and dynamically adjusting future stipples based on past errors. An exponential function models the natural decay of stipple sizes as ink is used in a flight. Stipples per second and variance of stipple placement are presented to evaluate our physical prints and robot control performance. For fully autonmous flight we power our quadrotor using a wired tether. We compensate for the tether in our control of the robot by assuming a static catenary curve of fixed length between the robot and the power source. We evaluate accuracy of hovering and flight on simple paths, and compare the results to untethered flight.

ABRÉGÉ

Nous décrivons une méthode pour créer des impressions pointillées en utilisant un robot quadrirotor volant. À un niveau bas, nous utilisons la capture de mouvement pour mesurer la position du robot et la toile, et un algorithme de commande robuste pour commander le robot de voler à différentes positions pointillées pour prendre contact avec la toile à l'aide d'une éponge d'encre trempée. Nous décrivons une collection de détails importants et les défis qui doivent être relevés pour le contrôle avec succès dans notre mise en oeuvre, y compris le modèle de robot estimation, filtrage de Kalman pour l'estimation de l'état, la latence entre la capture de mouvement et de contrôle, des interférences de communication radio, et le paramètre de commande de réglage. Nous utilisons un diagramme de Voronoi centroïde pour générer des dessins pointillées, et calculer une approximation gourmande du problème du voyageur de commerce pour attirer autant de pointillés par vol que possible, tout en tenant compte de la taille de stipple souhaitée et en ajustant dynamiquement pointillés futurs basés sur les erreurs du passé. Un des modèles de fonctions exponentielles la désintégration naturelle des tailles pointillées que l'encre est utilisée dans un vol. Pointillés par seconde et de la variance du placement de stipple sont présentés pour évaluer nos impressions physiques et les performances de contrôle du robot. Pour le vol entièrement Autonome en nous alimenter nos quadrirotor en utilisant une attache filaire. On compense pour l'ancrage dans la commande du robot, en supposant une courbe caténaire statique de longueur fixe entre le robot et la source d'alimentation. Nous évaluons la précision de vol stationnaire et vol sur des chemins simples, et comparer les résultats à vol untethered.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

ABSTRACT iii

ABRÉGÉ iv

LIST OF TABLES vii

LIST OF FIGURES viii

1 Introduction 1

2 Related work 5

3 Hardware and software overview 9

3.1 Quadrotor 9
3.2 Radio 10
3.3 On-board sensors 11
3.4 Optical tracking 11
3.5 System integration overview 12

4 Flight control 14

4.1 Software based speed controller 14
4.2 Quadrotor model 15
4.3 Hover controller 16
4.4 Yaw controller 17
4.5 Stipple controller 17
4.6 End to end latency measurement 20
4.7 Kalman filtering and state prediction 21
4.8 Radio communication improvements 21

5 Stipple generation and planning 24

5.1 Weighted centroidal Voronoi diagrams 24
5.2 Brush, ink usage, and stipple sizes 25
5.3 Stipple order and dynamic updates 26

6 Fully autonomous stippling 29

6.1 Voltage drop 29
6.2 Tether model 30
6.3 Autonomous ink refill 32
6.4 Torque model 33

7 Results and discussion 37

7.1 Implementation details 37
7.2 Physical prints 37
7.3 Tethered flight comparison 39
7.4 Discussion and limitations 42

8 Conclusion 47

8.1 Future work 47

References 50

LIST OF TABLES

Table page

7–1 Model parameters table 38

7–2 Print statistics summary 39

7–3 Tethered flight hover comparison 40

7–4 Torque modeled comparison 41

7–5 Tethered stippling comparison 42

LIST OF FIGURES

Figure		page
1–1	Teaser	2
3–1	The crazyradio	10
3–2	Motive tracking software	11
3–3	Systems integration overview	12
4–1	Stipple controller tuning tests	19
4–2	Kalman filter velocity estimation and smoothing	22
5–1	Stipple size decay model	26
6–1	Catenary model approximation	31
6–2	Autonomous refilling	32
6–3	Quadrotor body frame diagram	33
6–4	Torque model fit	35
7–1	Errors when maintaining hover over time	43
7–2	Physical prints comparison	45
7–3	Grace kelly 2000 stipples	46
8–1	Colored print	48
8–2	Long exposure photo of light painting of wire frame cube.	49

CHAPTER 1
Introduction

Pen plotters, fax machines, and modern laser printers are all highly specialized robots that permit the reproduction of images. Over many decades, these machines have been relied upon to produce physical copies of computer generated images. In contrast, it is interesting to consider how general purpose robots can be used to apply ink to paper. Notable recent examples of this alternative approach have used industrial robot arms and humanoid robots to draw and paint [14, 26]. We explore the benefits and challenges of using aerial robots for stippling, that is, the creation of images with many small dots.

Flying robots present interesting new possibilities for painting because they can easily get to hard to reach places. Equipped with a brush, a flying robot can make strokes at the top of a wall, and can likewise apply paint or ink to curved surfaces. We focus exclusively on stipples because this lets us avoid the hard problem of controlling contact between an airborne robot and the canvas during continuous strokes. The simpler problem of controlling the robot's trajectory with intermittent contact still remains an interesting challenge.

The aerial robots we use are quadrotors, which are special because they can efficiently put all power into torque free lift, and are simple and reliable thanks to inertial measurement units and stability control. Advances in hardware and miniaturization have made these flying robots very affordable and popular for both serious and leisure applications. In our work, we use Crazyflie quadrotors (see Figure 1–1 left) because they are

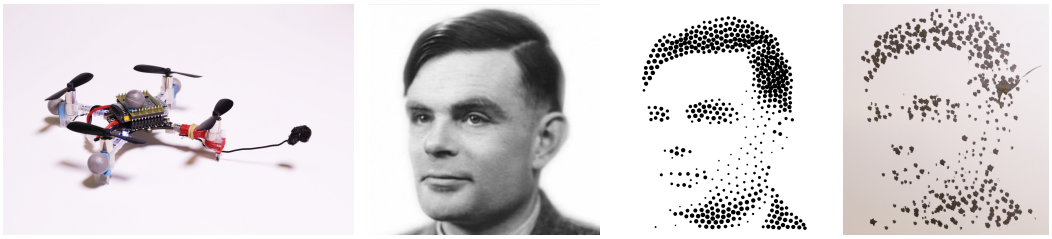

Figure 1–1: Our stippling aerial robot uses motion capture markers for positioning, and an ink sponge on a small arm to create stipples. Also shown is an example result: from left to right, a well known image of Alan Turing, a stippled version of the image with 500 dots, and a finished print with 500 stipples, drawn using a dynamic correction for the placement of stipples.

a particularly nice platform for research and development due to the open hardware and software design and well organized development environment. They are also small and light, taking up 9 cm^2 and weighing about 30 grams, which makes them much safer than larger quadrotors in an indoor environment.

There are a number of unique challenges to using quadrotors for stippling. The control task is all about putting a dot in the right place, with the time of placement being unimportant. At a high level, there are important computational problems, such as path planning with the constraints of limited battery life, dynamic adjustment of stipples to accommodate errors in placement, and the variability of stipple sizes as ink on the brush gets used up. At a lower level, there is a critical need for robust and stable control. Trajectory control of position is difficult because the robot is under-actuated. With four motors, the robot can only adjust its thrust, roll, pitch, and yaw. Therefore, control of horizontal position can only be achieved by rolling and pitching to let thrust produce accelerations in directions orthogonal to gravity. Under-actuation, combined with the small size and weight of our robot, makes absolute accuracy in position challenging as the robot is easily

perturbed by air currents. State estimation is also challenging for mobile robots. Larger flying robots often include cameras and GPS systems that allow absolute position estimates for both indoor and outdoor flight. Because of the small size of the Crazyflie, it has a limited payload for cameras or additional sensors, and likewise has limited compute power for performing on-board localization. We instead use a motion capture system to track position and orientation easily and accurately.

Flying robots have been used within a variety of art projects for the creation of light paintings, for example demos by Ascending Technologies, and Spaxels at Ars Electronica. Quadrotors have likewise recently been used for producing rim illumination for photography [24]. However, to the best of our knowledge, we are the first to address the computational issues of painting with autonomous aerial robots.

In the pursuit of fully autonomous flight and stippling we power our quadrotor using a wired tether. To account for the force and torque caused by the tether on our quadrotor, we model the tether in our control by assuming it takes the shape of a static catenary curve of fixed length between the robot and the power source. We compare the accuracy and control fidelity of using a tether to power the quadrotor opposed to the results of untethered flight.

The remainder of this thesis is organized as follows. We first acknowledge previous work on robots being used in the creation of artwork, generation of stipple placements from source images, and quadrotor control problems in Chapter 2 Related Work. Chapter 3 provides a high level look at the platforms we use and how they interact with each other. We describe our model and controllers for flight control in detail in Chapter 4. Chapter 5 describes the process of stipple generation and how feedback of the placement error can be used to dynamically update optimal placement of future stipples. Chapter 6 presents

the model used for the wired tether and how it informs the controller to compensate for the additional tension force and torque for fully autonomous stippling. We present and discuss our results in Chapter 7. We suggest possible future extensions that can be made using our control foundation in Chapter 8.

CHAPTER 2
Related work

The creation of art by robots is a topic that spans several fields. It involves aesthetic choices in the placement of brush strokes, selection and tuning of state estimation and control parameters to make the robot execute these strokes, and computational aspects to efficiently plan robot trajectories and dynamically adjust for errors.

There are a number of examples where robots have been used in the creation of art and drawings. One example is the sketches and portrait drawing of Paul the robot [25, 26]. In earlier work, Lin et al. [13] use a camera and a humanoid robot to draw a line drawing portrait of the person in view. In similar work, Lu et al. [16] use cameras and visual feedback to create images with hatching patterns that capture both texture and tone of the original image. Indeed, feedback is a critical aspect in robot drawing and painting systems. Other computational approaches to painting with a robots address feedback guided stroke placement [4], image stylization with semantic hints [15], and dynamic adjustment of layered strokes [14]. In our work, the challenge of stippling with flying robots is significant because of how hard it is to control the position of the robot and the brush, and thus, dynamic adjustment of stipples is critical.

Understanding the shapes of strokes is useful in the analysis and creation of robot or computer art. Berlio et al. [1] design a curve representation suitable for creating and analyzing graffiti tags. Lehni developed a system called Hektor [11], a graffiti robot positioned by cables. It is small, light, and can work on a large surface, but that surface must

be flat and there must be places where the cable pulleys can be mounted. By using a flying robot we are not limited to planar surface, but this comes at the cost of losing precision in position control. By using stipples to create images, we avoid the problem of modeling and drawing more complex strokes or curves.

Following the work of Secord [22], we use a centroidal Voronoi diagram to compute stipple positions. When working with a small number of stipples, it can also be advantageous to encourage stipple placements that reveal important image features such as edges [19]. In more recent work, Li and Mould describe how error diffusion allows for reduced stipple counts while preserving structure of an original image [12]. While our results would benefit from these recent advances, we use Secord's method because it is simple to implement, fast to compute for small stipple counts, and easy to update during the drawing process to account for errors in the placement of stipples.

Stippling robots can be found within the maker community, specifically the *eggbot* (available as a kit from distributors such as Adafruit and SparkFun). The eggbot is a pen plotter that is designed for drawing on the surface of an egg. This is a nice example of stippling on a non-flat surface. By using an aerial robot, we see the advantage that future versions of our robot will be able to apply ink to a wide variety of hard to reach non-flat surfaces.

Finding optimal paths is an important problem for a stippling robot. Optimal paths have been used in the construction of labyrinths and mazes [20]. Similarly, approximate solutions to the traveling salesman problem have been used to produce continuous single-stroke drawings [2, 8]. In our case, we have a problem of finding a path that takes the robot between a subset of the stipples before returning to a landing pad for a fresh battery

and an ink refill. This is related to a traveling salesman problem as we would like to draw as many stipples as possible on a single charge, but there there are additional challenges and complexity involved. Specifically, the time and distance to fly between the stipple positions is not the only cost because there is also the time cost of stabilizing the robot before creating a stipple, and constraints related to the desired stipple sizes.

Optimal path planning for robots has received a vast amount of attention for robotic manipulators, vehicles, and flying robots [7, 9]. Furthermore, many control problems specific to quadrotors have been investigated, such as methods to produce aggressive maneuvers [18] and flips [17]. The robot control algorithms we implement in this work is largely inspired by that of Mellinger et al. [18], as well as the PhD thesis of Landry [10].

For continuous flight not bounded by battery capacity, we power the quadrotor using a wire tether. We model the shape the hanging wire takes and include the forces it generates into our controller. While these forces may be negligible for larger robots or disregarded in other applications where absolute position accuracy is less important [24, 27], in our application it is critical because of the small robot size and our desire for the accurate placement of stipples. A related problem to adding a tether to the robot is control and planning while taking into account the dynamics of a slung load [5, 23]. When the slung load is also allowed to become slack, the problem becomes even more challenging [3]. We take a simpler approach of assuming the dynamics of the cable are small, given air resistance on the very light weight cable we use, and we show that accounting for a static tether can make a significant improvement. Tethers are often used for safety, and for testing new control algorithms, but they are also common for providing power or for communication. We note the work of Zikou et al. [27] being similar to ours in tether modeling, though they focus

on controlling a spool to provide additional length or take up slack as necessary. Given that they use a larger robot, they let the existing quadrotor control handle the disturbance introduced by the weight of the tether.

CHAPTER 3
Hardware and software overview

With the interaction of measurement devices, computation, and communication we are able to precisely control a quadrotor for the purpose of creating printed works. To accomplish this we make use of various hardware and software systems. We present the hardware used and the integration with the systems developed and how they communicate. We make use of optical motion tracking to measure the position and orientation of the quadrotor. We implement flight control software that uses the motion tracking information to compute the flight commands to be sent to the quadrotor over radio. The flight control system also interacts with the higher level stipple server that ultimately takes a source image and converts it into an ordered set of stipple locations.

3.1 Quadrotor

We use the Crazyflie 2.0 Nano Quadcopter development platform as a launching point for a stippling quadrotor. The Crazyflie is a versatile, lightweight quadrotor with an open development environment. This development environment is easy to use and provides full access to the various sensors and measurement devices on the quadrotor, as well the possibility to manipulate the controlling firmware. The firmware can be flashed using only a radio connection, making updates convenient. It is one of the smallest quadrotors available on the market. Its small size and durable construction is ideal for flying indoors and near

Figure 3–1: The Crazyradio from Bitcraze.se

people without requiring other safety precautions as with larger quadrotors. It has two on-board microcontrollers. The first microcontroller is an ARM Cortex-M4 embedded processor (STM32F405) with a floating point unit that supports all ARM single-precision data operations and types. This microcontroller is responsible for running the main firmware on the Crazyflie. The second microcontroller handles power management and the radio.

3.2 Radio

The Crazyflie supports communication over a 2.4 GHz radio using a Nordic Semiconductor nRF51822. The control software communicates with the chip using a USB dongle that integrates a Nordic Semiconductor nRF24LU1+ chip, known as the Crazyradio as can be seen in figure 3–1. The communication software for the Crazyflie using the Crazyradio is also completely open source. Typically a Crazyradio and Crazyflie communicate on a one-to-one basis, however work has been done to allow a Crazyradio to communicate with multiple Crazyflie silmultaneously through the use of PC-sided time slicing [6].

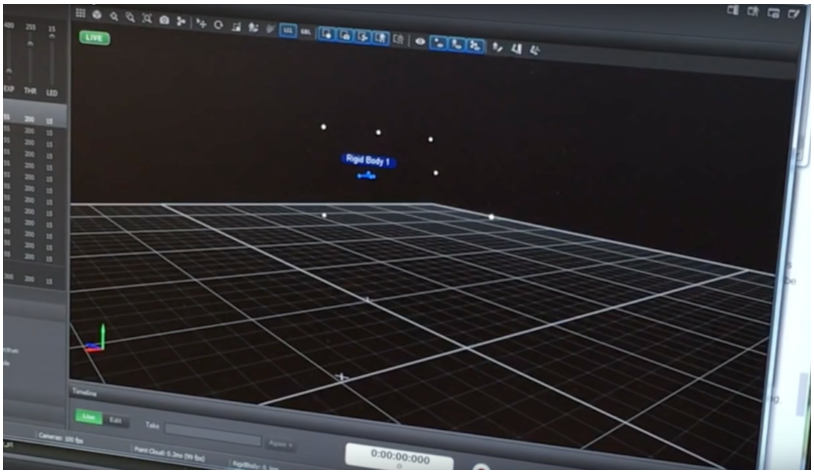

Figure 3–2: The motive software tracking the quadrotor in flight. The extra markers track the location of the canvas and the mounting location of the end of the tether.

3.3 On-board sensors

The Crazyflie 2.0 is equipped with an Inertial Measurement Unit (IMU), that contains a 3-axis gyroscope and 3-axis accelerometer. The on-board controller uses these sensors to achieve desired orientations. Measurements from these sensors can be provided to the offboard controller if required. However we elect to use only information obtained from optical tracking. First, because the radio bandwidth has limited capacity, and sending control information to the quadrotor as frequently as possible with little latency is paramount. Second, consolidating optical tracking and IMU measurements from different sources and latencies would be difficult.

3.4 Optical tracking

The control algorithms we present rely on accurate real-time position information of the quadrotor. Methods that rely on the on-board gyroscope and accelerometer are not sufficient for providing the millimeter accuracy in measurement required for stippling. We

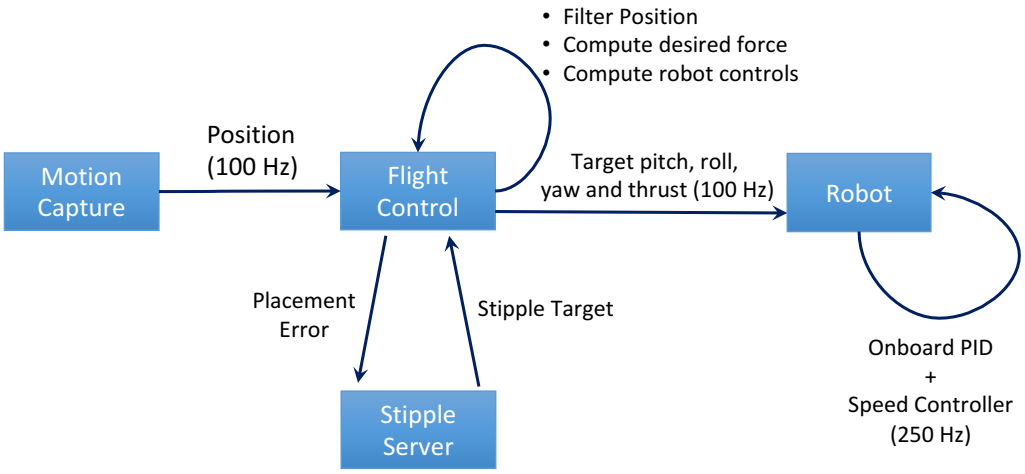

Figure 3–3: An overview of the the control systems and how they interact with each other.

use OptiTrack Flex 3 infrared cameras and the accompanying Motive software to optically measure the position of the quadrotor during flight. Optical tracking is used to measure the location of the quadrotor, the canvas and the tether endpoint as can be seen in figure 3–2. The quadrotor is tracked using 4 markers placed directly on the quadrotor. We stream measurements of the position and orientation of the quadrotor to our control algorithms at a consistent frequency of 100Hz using the NatNet SDK from OptiTrack.

3.5 System integration overview

The entire system can be viewed as four independent systems interacting with each other asynchronously through message passing. Figure 3–3 shows an overview of the entire system and the direction information travels between the sub-systems. The flight control system receives position and orientation information from the motion capture system at a rate of 100 Hz. Using this information it computes the orientation and thrust values to achieve the desired target position and sends a message to the quadrotor. The on-board

proportional-integral-derivative controller (PID controller) running on the quadrotor updates at a rate of 250 Hz using measurements from its gyroscope to achieve the target orientation and thrust sent from the flight control system. The flight control system receives the target stipple locations from the stipple server. Following a stipple being placed, the flight control system sends an estimated placement error to the stipple server and requests the location for the next stipple. The stipple server uses the placement error to dynamically adjust the location of future stipples before providing the next stipple location to the flight control system.

CHAPTER 4
Flight control

Much of what we would like to accomplish involves having the quadrotor approach a specific point or maintain its position. In some cases, based on perturbations in the air or contact with a canvas, we will need to abruptly change the current control plan. In this section we describe two simple methods that we use to control the flight for the purpose of being able to draw points on a canvas. These methods both rely on having high precision measurements of the location of the quadrotor obtained from the motion capture system. We acknowledge that the model we use has been simplified to avoid a full system identification of the quadrotor in use, while recognizing that much higher fidelity control can be obtained with better models, learning, and by controlling the motor torques directly instead of relying on the Crazyflie's internal PID control.

4.1 Software based speed controller

Due to its small form factor, the motors of the Crazyflie 2.0 are not brushless (unlike some larger quadrotors) and are powered by an unregulated power supply. This has the disadvantage that the torques produced by the motors do not always reflect the commands being sent. We correct for this in software by using feedback from the measured battery voltage [10]. The duty cycles sent to the motors can be treated as a function of the measured battery voltage and the desired angular velocities of the motors,

$$u = \frac{V_{\text{max}}}{V_{\text{actual}}} \left(\sqrt{\omega^2 - \beta} \right) + \alpha, \tag{4.1}$$

where u is the command input to the motors, V_{max} is the battery's rated voltage, V_{actual} is the most recent measured battery voltage, and α can be interpreted as the minimum duty cycle that must be sent to the hardware in order to get any angular velocity ω at the motors. Parameter β then accounts for the fact that the propellers start out at a certain non-zero velocity. We find that this software based speed controller proposed by Landry is critical for obtaining reliable thrust control with our Crazyflies.

4.2 Quadrotor model

The simplified model we use represents the quadrotor as a point mass that can align its pitch and roll instantly. This has the benefit of requiring only two forms of off-line system identification, and we can accomplish both using an inexpensive scale. First, we must know (approximately) the mass of the quadrotor. Second, we require a mapping between the commands sent to the motors and the total force generated by the motors.

The duty cycle commands that are sent to the motors of the quadrotor are represented arbitrarily by 16 bit unsigned integers. We experimentally determined a mapping between these values and the actual forces produced by inverting the quadrotor and measuring the force produced while varying this value. With the software based speed controller in place, this relationship can be approximated with a linear function.

Absolute accuracy in positioning is difficult because quadrotors are inherently under-actuated. Any change in position is dependent on the current orientation, therefore errors in position are acted on indirectly by controlling orientation. The controllers described in the following sections compute the desired pitch, roll, yaw rate, and thrust and rely on the Crazyflie's internal PID to achieve the desired orientation.

4.3 Hover controller

A PID controller is used to reach and maintain a desired position with zero velocity. This is accomplished by using the pitch and roll angles of the quadrotor to control its position. PID feedback on the position error is first used to compute the desired net force acting on the quadrotor. This is computed as a force,

$$F_{\text{net}} = k_P(x_t - x) + k_I \int (x_t - x) + k_D(\dot{x}_t - \dot{x}), \tag{4.2}$$

where x_t is the target position, $\dot{x}_t$ is the target velocity, and x is the current position of the quadrotor in world space. In situations where a stable hover at a point is desired, the target velocity should be zero. Once the desired net force has been computed, the steering force can be computed by subtracting all other body forces from the net force. For all intents and purposes, this simply involves subtracting the force due to gravity. Due to physical limitations, not all steering forces can be realized by the quadrotor. The steering force is therefore clamped to a maximum force, (the maximum measured force that the quadrotor produced during system identification).

Once the desired steering force has been computed it is transformed into the intermediate body frame using the most recent reading for the yaw from the motion capture system. This is accomplished with a simple rotational transform about the y_W axis (i.e., world vertical). Once in this frame, the desired pitch and roll are computed so that the y_B vector (i.e., robot body vertical) will be pointing in the same direction as the steering force. The value for thrust that is sent to the quadrotor is computed using the linear mapping experimentally determined with the magnitude of the steering force as input, (see

Section 4.2). Note that we control the yaw of the quadrotor with a separate P controller that calculates the yaw-rate.

4.4 Yaw controller

The yaw of the quadrotor is controlled by sending a desired yaw-rate to the quadrotor. We use a P controller that acts on the error between the target yaw and the current yaw. The target yaw is calculated using the normal of the canvas at the target stipple location. For a flat canvas the target yaw is constant regardless of the stipple location, however for curved surfaces, such as a column, the target yaw would vary across the surface.

To decrease stabilization time, we look at the yaw of the quadrotor 400 ms after performing a stipple. Ideally, following a contact, the impulse from the sponge contacting the canvas would not affect the yaw of the quadrotor. However, in practice, we observe small consistent changes in the yaw following impact. These errors can be reduced by manually re-positioning the brush tip relative to the quadrotor. A better solution is to use feedback from the measured error following a stipple. We do this by adding a modified I term to the yaw controller. For this I term we update the integral value only during a period of 400 ms following contact with the canvas. This PI controller results in improvements for the rate of stippling as the quadrotor is able to more quickly stabilize and move on to performing the next stipple. This also improves the reliability of the controller, since the impulse acts to push the quadrotor directly away from the canvas, which prevents accidental collisions with the canvas following stippling.

4.5 Stipple controller

The stipple controller is a PD controller that is used to control the position of the robot in the plane parallel to the canvas. It is used in combination with the hover controller while

the quadrotor is completing a stipple action. First, the quadrotor's position is projected into the plane of the canvas. Then the error between the projected position and the target stipple location is calculated and used as input to the PD controller for computing the desired net force. This force of the stipple controller is always in a direction parallel to the canvas. The quadrotor's desired net force is the sum of the forces computed by the hover PID controller and the stipple PD controller.

The act of stippling consists of three stages, preparation, stippling and recovery. During the preparation stage, a target hover position is computed at a distance of 20 cm away from the desired stipple position along the normal of the canvas, (or 12 cm between the tip of the brush to canvas). The quadrotor stabilizes around this point before attempting to draw the stipple by using the hover controller. The quadrotor is said to be stable if it can maintain an error of less than 2.8 cm for 350 ms between its position and the target hover position. Additionally, during this period its velocity must not exceed 4.2 cm/s. These values were determined experimentally. We optimized first for accuracy, by trying to achieve the minimal standard deviation of stipple placement error. Once this was found, we minimized time between stipple placements until we began to see a degradation in stipple placement accuracy. Only when the quadrotor is stable may it proceed to the stippling stage.

During the stippling stage, the quadrotor uses the sum of the desired net forces computed by the hover controller and the stippling controller. The target location for the hover controller is set to the stipple location. In addition, the hover controller is set to have a target velocity in the direction of the canvas to increase the momentum of the quadrotor at the point of impact. The act of completing a stipple is determined when the distance

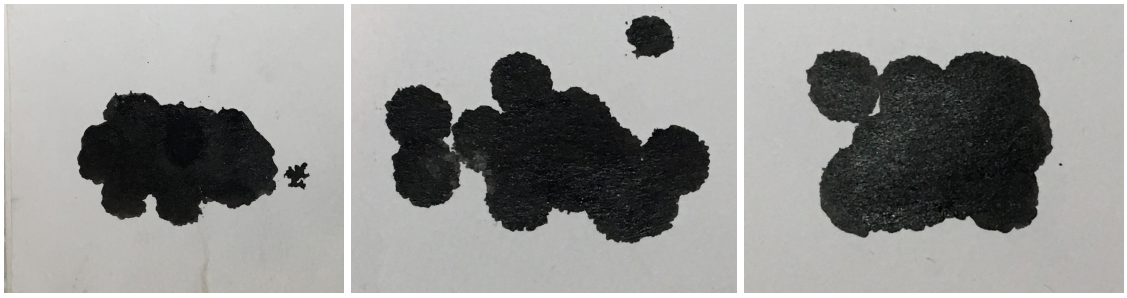

Figure 4–1: Stipple tests for the purpose of tuning the controllers. The quadrotor is instructed to stipple the same location repeatedly 32 times. The location of each stipple and the rate of stippling is recorded. The standard deviation of the placement error for the above images and average stipple rate: from left to right, σ_h = 6.0 mm, σ_v = 3.1 mm, 7.5 seconds per stipple, σ_h = 7.0 mm, σ_v = 4.0 mm, 3.3 seconds per stipple, and σ_h = 5.9 mm, σ_v = 4.3 mm, 4.0 seconds per stipple.

between the quadrotor and the canvas is less than a fixed threshold. When this is detected, control proceeds to the recovery stage. The location of the placed stipple is computed using the location and orientation of the quadrotor measured by motion capture, using the known position of the sponge in the quadrotor's reference frame.

The recovery stage is responsible for controlling the robot as it moves away from the canvas, and preventing any accidental collisions following the stipple. It uses the hover controller with a set target velocity away from the canvas. The target location of the hover controller is set to the expected location of the next preparation stage. This puts the quadrotor in a position nearby to the next target as the dynamic stipple planner typically makes only small adjustments to future stipple locations between each stipple. Control transitions to the next preparation stage once 500 ms have past and the location of the next stipple is known.

4.6 End to end latency measurement

The amount of latency from motion capture to quadrotor response has an important effect on the quality of the control. We experimentally measure the latency in the system and account for it to improve control. To measure the latency we attach an LED to the quadrotor and turn it on upon receiving a command. The command is sent from the computer upon receiving input from the motion capture revealing that a tracked object was moved (the threshold to trigger the command is set to 0.4 mm). We measured the end to end latency using a high-speed 1200 fps Nikon camera. The video records the LED and tracked object being struck, and thus we count the number of frames between the time of impact and the illumination of the LED. We find the latency to be $\Delta T = 49.6$ ms $\pm$ 11.6 ms after repeating the experiment 20 times.

There are multiple sources that introduce a variable amount of latency into the system, such as the amount of processing required to identify the position of the quadrotor by the motion capture system. This is dependent on the number of markers in the scene, the visibility of the markers relative to the cameras, and performance can be degraded when there are reflective surfaces in the capture volume. To minimize potential problems, reflective surfaces were covered whenever possible and all unnecessary markers were removed from the capture volume.

Additionally, radio interference may require a command to be resent multiple times before reaching it's destination. Depending on how many times the command must be sent, there may be up to 10 ms of additional latency before the command is simply dropped.

4.7 Kalman filtering and state prediction

We use a Kalman filter to filter the position obtained from the motion capture system. The filter also provides estimates for the velocity and acceleration. To counter the latency in the system, we predict what the quadrotor's position and velocity will be at the time it receives the command and use those values when computing the desired net force. Specifically, we compute

$$\begin{aligned} \dot{x} &= \dot{x}_F + \Delta T \ddot{x}_F \\ x &= x_F + \Delta T \dot{x}_F + \Delta T^2 \ddot{x}_F \end{aligned} \tag{4.3}$$

where x_F, $\dot{x}_F$, and $\ddot{x}_F$ are the position, velocity and acceleration estimated by the Kalman filter, and ΔT is the average latency of the system, which we previously determined experimentally as described in Section 4.6. Figure 4–2 shows how the filter performs for estimating velocity when the crazyflie is hovering, compared to a simple numerical differentiation of the motion capture position trajectory.

4.8 Radio communication improvements

The default communication protocol provided by the Crazyflie is not optimal for real-time control in environments with interference. The Crazyflie communicates using a 2.4 GHz radio which is the same range of frequencies used by WIFI enabled devices. A crowded radio spectrum can interfere with the Crazyflie's ability to send and receive control packets. The existing protocol maintains a queue of all commands, and sending a command from the computer to the quadrotor requires an ACK to be sent back before sending the next command in the queue. If no ACK is received, the computer must retry sending the same command up to a maximum of 10 times before forcing a disconnect.

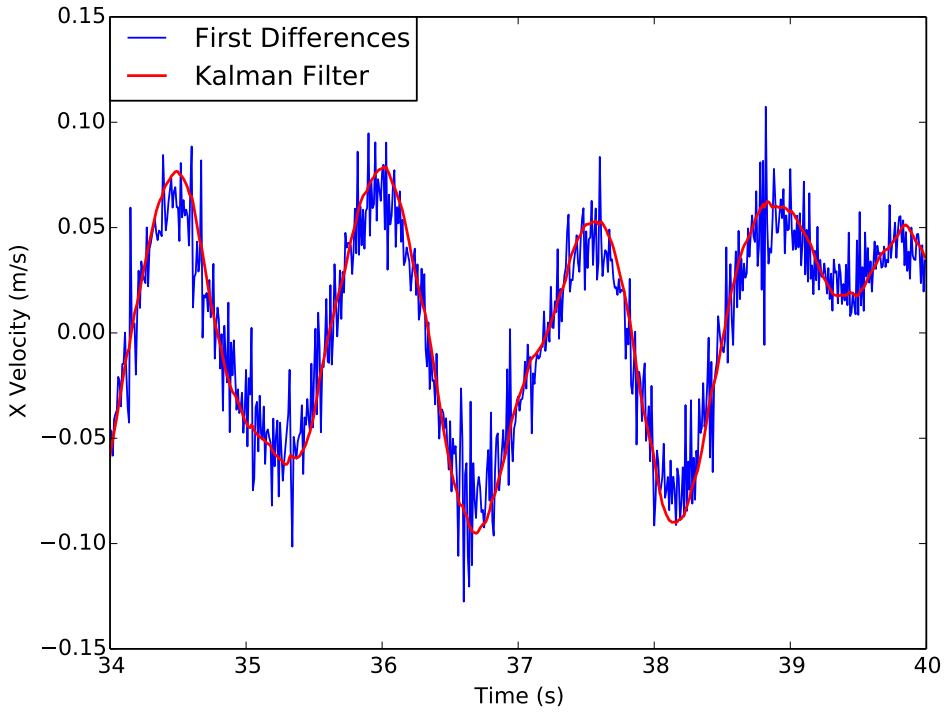

Figure 4–2: The Kalman filter provides a smooth estimation of the quadrotor's velocity

This implementation has the benefit, barring a disconnect, that all commands are ensured to reach the quadrotor in the order they were sent. However, in the case of interference, we often found that commands from the computer were successfully received, but it was only the ACKs being returned that were not received. In this situation the computer would continue to repeat the same command, even when new commands were waiting in the queue. When the interference was high, the computer would force a complete disconnect despite the quadrotor successfully receiving all commands being sent.

The protocol we implement relaxes the requirement that all commands be received. If a new command enters the queue, we stop trying to resend an old command and send the most up to date command instead. By allowing the possibility that some commands

may never be received, we can focus on sending commands using the most up to date position information from the motion capture. This is a more appropriate for a real-time scenario since commands computed using delayed position information are no longer valuable. This protocol also prevents duplicate messages being received in the case where the original command was received but the computer simply did not receive the ACK.

CHAPTER 5
Stipple generation and planning

To provide our aerial robot with something to draw, we convert a selected image into a set of stipples given a set of constraints such as the number of stipples and a range of stipple sizes. Because the robot will make small errors when placing stipples, it is important that we be able to quickly update the positions of the remaining pixels and to adjust the order in which stipples are drawn. In this section, we review how we compute a set of stipples for an image, and discuss the problems of stipple ordering and dynamic updates.

5.1 Weighted centroidal Voronoi diagrams

The core of our stippling algorithm is based on weighted centroidal Voronoi diagrams (CVD), as described by Secord [22]. The main idea is to start with a random set of points and to compute a Voronoi diagram. Then, we compute centroids for each region in the Voronoi diagram by integrating over the pixels of the target image using the brightness as weights. We then shift each point to the centroid of its region and repeat these steps until we reach a stable configuration.

To select the size r_i of a stipple i, we use the average brightness of its Voronoi region. We linearly map the average pixel brightness $\rho_i \in [0,1]$ to a stipple size in the available range,

$$r_i = \rho_i r_{\min} + (1-\rho_i) r_{\max}, \tag{5.1}$$

where $r_{\min}$ and $r_{\max}$ are the minimum and maximum radii. Because of the limited range of stipple sizes and our desire to use small stipple counts to minimize print time, a printed result will generally have a lighter tone than the original image. Furthermore, large light areas in the image result in a sparse collection of tiny stipples that would be poorly drawn given our minimum stipple size. Therefore, we prune these points by removing any stipples that fall below a given threshold.

5.2 Brush, ink usage, and stipple sizes

The brush we use to draw stipples is a small spherical sponge mounted at the end of a stiff wire arm. At the beginning of each flight, we soak the sponge with a black liquid acrylic ink. While the size and shape of the sponge is an important factor in determining the size of stipples that we will draw, the amount of ink remaining in the sponge is also an important factor. The inset image in Figure 5–1 shows several sequences of how stipple sizes decrease as drawing progresses from left to right and top to bottom. Using a set of six sequences, we measure the area of the stipples and build an approximate ink decay model by fitting the exponential function shown in Figure 5–1. This allows us to predict the size of the next dot the robot will draw. Note that the velocity of the robot at impact will influence the deformation of the sponge, and will allow for some additional control of the stipple size. However, to maintain good accuracy of stipple placement we use a consistent velocity and control strategy for all stipples.

Note that the maximum stipple $r_{\max}$ size comes directly from the exponential function of our ink model. We set the minimum stipple size by evaluating the function at the maximum number of stipples that we can draw in a flight (typically no more than 70).

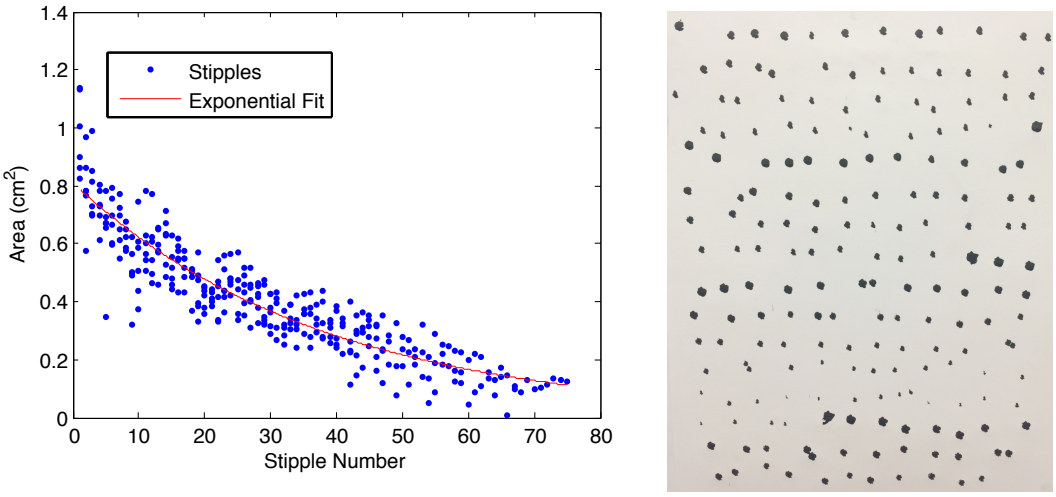

Figure 5–1: Stipple size decay model fit to six sequences of ink transfer tests. The area of each stipple is measured in the source images, from which we fit a two parameter exponential model.

5.3 Stipple order and dynamic updates

While the static set of stipples generated by the algorithm in Section 5.1 generates good results, the quadrotor will ultimately make errors in the placement of each stipple. To try and mitigate this error, we use a dynamic update to the remaining stipple positions to accommodate errors as they happen.

Our dynamic update happens on-line using a server-client architecture. The quadrotor controller requests the next point to draw from the server and reports back the position it ended up hitting. The server then sets the position of this point and marks it as unmovable. The optimal position for the remaining points is then adjusted by running a few iterations of the CVD algorithm. This is fast because we only constrain one point to a new position, and we start from a stable configuration. For a image consisting of 2000 stipples, the

updated positions can be computed in less than 500 ms. Thus, the next point is always available by the time the quadrotor is ready to start its flight to the next stipple location.

There are three main factors that influence the order in which we want to draw stipples. First, we want to minimize the distance traveled by the robot between stipples throughout a drawing session. This will increase the rate at which stipples are drawn and maximize the number that can be drawn on each flight with a fully charged battery. Second, we want the sizes of a sequence of stipples to be drawn to match as best as possible the ink usage model estimated in Section 5.2. Thus each flight should start with large stipples and end with the smallest ones before the battery is depleted. Third, if dynamic updates are to have a benefit on the final result, it is important that completed regions be grown progressively. For instance, consider the process of drawing equally spaced stipples to form a line. If we draw from left to right, then we can always adjust the next point to the left or right to account for the error. In contrast, if the order is random, then we will have stipples that need to be placed between two others and we must compromise on minimizing the error to both.

Originally, we implemented an approximate solution to the traveling salesman problem using a generic algorithm. However for our application we had more constraints then simply computing the minimum path. Our path is broken up into drawing disjoint segments at a time, as well as taking into account the desired size of the stipple. Furthermore, our robot draws an unpredictable number of stipples before the battery is drained (typically between 50 and 70) at which point the battery is swapped and the ink reloaded. To handle the shifting stipple positions and irregular session length, we instead design a dynamic greedy strategy for stipple ordering. At any given time, the next optimal stipple is

selected as closest using a metric that combines distance and stipple area. This allows us to partition the stipples into as many flights as necessary, with no prior knowledge, and can easily accommodate our dynamic adjustment of stipple positions and sizes. We adjust the weighting of distance to stipple area in the metric by hand on a per image basis by observing synthetic results (i.e., a simulation that takes into account canvas size, stipple sizes, and placement error). A good weighting will produce regions that grow relatively continuously while also matching stipple sizes.

CHAPTER 6
Fully autonomous stippling

The previous chapters outline our method for semi-autonomous stippling, with the limitations being short flight times requiring frequent battery changes and refilling the ink supply. We present here our methods for achieving fully autonomous stippling.

For fully autonomous stippling an entire drawing containing thousands of stipples should require no user interaction following start to finish. Considering the duration of the quadrotor's battery is typically 6 minutes, this would require an impossibly fast rate of stippling. Wireless charging of the battery by having the quadrotor accurately land on a charging platform was considered as a possible solution. A disadvantage to this approach is the brush drying out during charging. Instead we get rid of the battery, and power the quadrotor directly from a wired power supply. Such a solution has accompanying problems that must be overcome.

6.1 Voltage drop

Stippling requires very accurate position control. In order for the wire not to affect flight control its weight must be small compared to that of the quadrotor but also sufficiently long to not restrict stippling larger canvases. We use 30AWG wire 180 cm in length weighing 2.4 grams. The quadrotor requires up to 3 A of current in the range of 3.0 V to 4.2 V to run properly. The resistance of the wire is not negligible at these values, roughly 0.6 Ω, and there would be a significant voltage drop across the wire. The resistance of the quadrotor is variable and dependent on the thrust, so increasing the supplied

voltage is not a viable solution as the quadrotor would receive an irregular voltage as the thrust varies. To counteract this we attach a small buck converter to the quadrotor which drops the 12 V coming from the power supply to a consistent 3.3 V before powering the quadrotor.

6.2 Tether model

Even at only 2.4 grams, the force and torque from the wire acting on the quadrotor has a significant negative impact on the control. We account for this by modeling the wire and estimating the force and torque acting on the quadrotor. Similarly to how the flight controller accounts for gravity, tension is treated as an additional external force. However unlike gravity which does not induce torque, connecting the wire directly to the center of mass of the quadrotor is not feasible and therefore the wire's tension induces torque.

We model the hanging wire using a catenary curve, which is the shape an idealized hanging cable assumes under its own weight when supported only at its ends. One endpoint of the cable is fixed to a stand (90 cm high) while the other is attached to the base of the quadrotor. The motion capture runs at a frequency of 100 Hz. In each time step, we numerically solve for the unknown parameters a, b, and c in

$$F(x) = a + \frac{1}{b}\cosh\left(b(x-c)\right), \tag{6.1}$$

obtaining the catenary curve that a cable of fixed length would assume by the two endpoints. We can find the direction of the tension forces by computing the derivative of the curve for each endpoint. We can then analytically solve for an approximation of the tension forces with the assumption that the system is at equilibrium.

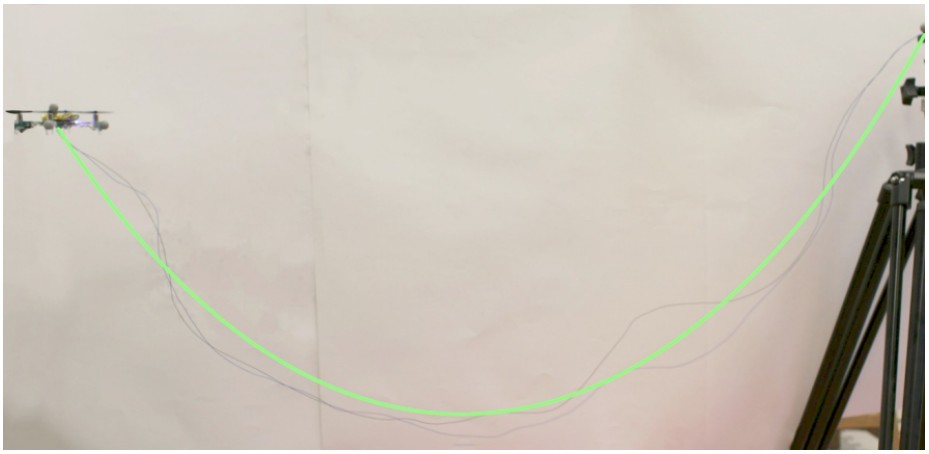

Figure 6–1: The quadrotor in flight attached to the tether with the overlaid green line showing the curve our model predicts.

For the case of accurate stippling, the ability to hover accurately at a point in preparation of stippling is paramount; it is only when our system detects that the position error is below a desired threshold that the quadrotor will perform the stipple. Therefore, the assumption that the system is at rest is not too restrictive in regards to stippling because the system will ideally be at rest when accuracy is most required.

One problem with the tether that we faced early on was oscillations of the sagging cable reducing the quadrotor's ability to maintain a stable hover, which we discuss further in Section 7.3. This is most pronounced when the endpoints of the cable are relatively close together compared to the cable's length. We dampen these oscillations by tying a lightweight thread to the midpoint of the cable, which does not have a significant impact on the mass or shape of the cable but dampens swinging motion by the frictional forces of it dragging on the floor.

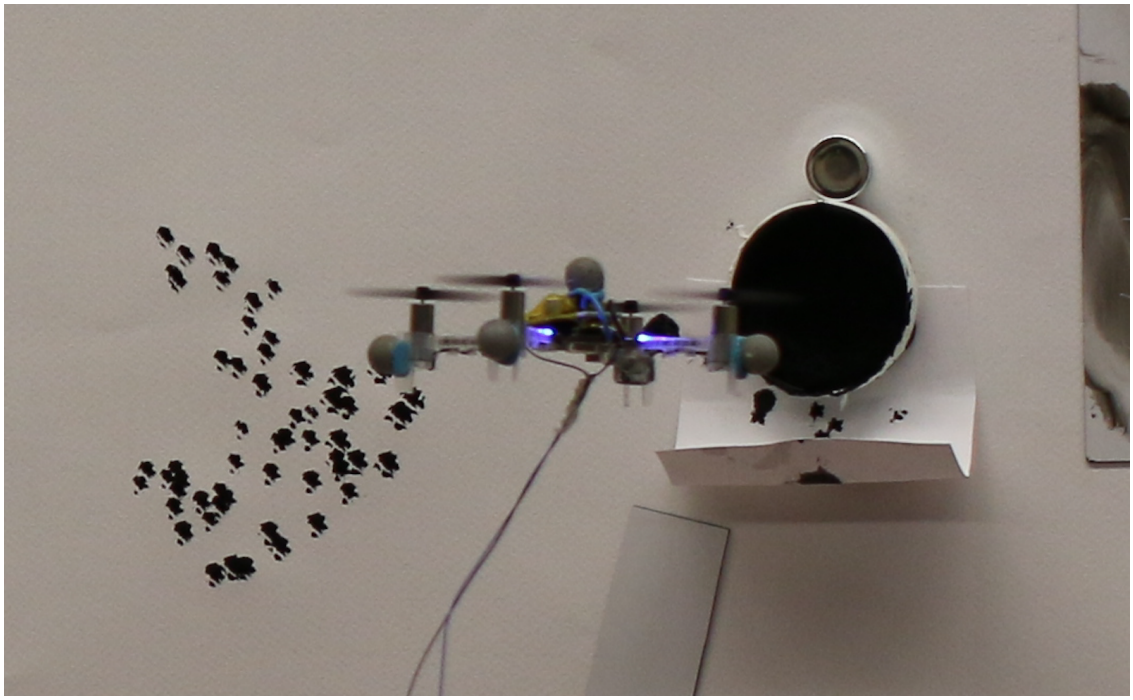

Figure 6–2: Quadrotor is preparing to refill mid-flight using the ink well. To construct the ink well, we cut out the bottom of a coffee cup. We then cut a sponge to fit and glue it to the base of the cup. Before starting a drawing, we fill the sponge of the ink well to capacity, such that it is fully saturated but will not drip. We finally attach it to the canvas and record its location so the stipple server knows where to send the quadrotor for refilling.

6.3 Autonomous ink refill

The final hurdle for fully autonomous flight is the ability for the quadrotor to replenish its own ink while stippling. Our solution is elegant in that it uses the existing controller with no need for modification. We make ink wells and attach them next to the canvas. When the ink model predicts the ink is running low, we set the quadrotor's target stipple location to the center of the ink well. The quadrotor stipples the ink well repeatedly to refill its ink. All of this is controlled by the stipple server. As far as the quadrotor controller is aware, it is performing a stipple no different than it would on the canvas. We

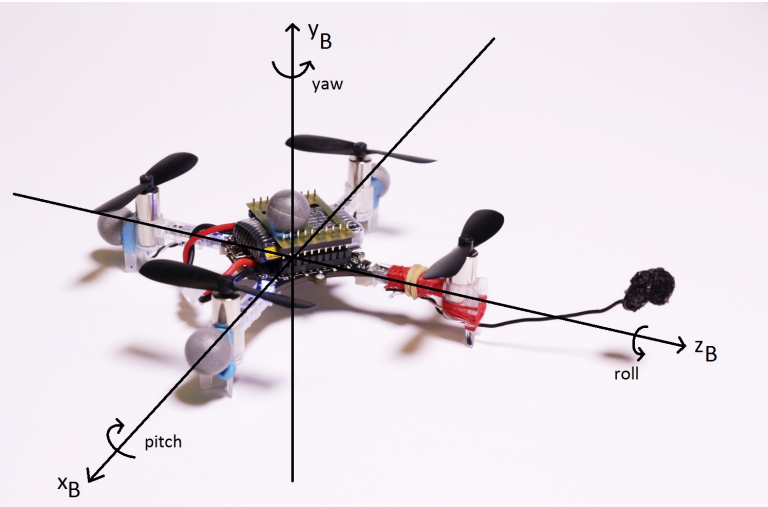

Figure 6–3: The quadrotor with a diagram of its body frame overlaid.

use an exponential decaying ink model constructed as outlined in section 5.2. Through experimentation we find that 10 refill stipples results in a consistent maximum capacity regardless of the amount of ink already in the quadrotor's sponge. However for longer drawings, to keep a consistent range of stipple sizes the ink well should be refilled roughly every 500 stipples.

6.4 Torque model

The catenary model provides us with an estimate of the force of tension caused by the wire, but not the torque acting on the quadrotor. Our simplified model of the quadrotor does not provide an inertia tensor. Also the lever arm is not known, since there is no estimate of the center of mass. For these reasons an experiment was performed to learn the torque induced on the quadrotor from the tension force using a simplified model.

For our model we assume that the principal axes run along the arms of the quadrotor as can be seen in Figure 6–3. The x_B, y_B, and z_B axes correspond to the pitch, yaw, and

roll of the quadrotor respectively. Let F be the force of tension exerted by the tether on the quadrotor and r be the lever arm from the quadrotor's center of mass to the attachment point of the tether. Through experimentation we found the yaw induced by the tether to be negligible, so we only care about the torque acting on the pitch and roll of the quadrotor, which we approximate using

$$T_p = r_y F_z - r_z F_y \tag{6.2}$$

$$T_r = r_x F_y - r_y F_x \tag{6.3}$$

respectively. Both Equations 6.2 and 6.3 are linear with respect to the tension force. To construct the model to approximate the torque we identify the two unknown parameters for each equation using the following experiment.

First the quadrotor is flown to a variety of positions relative to the mounting point and allowed to stabilize. The integral term of the quadrotor's on-board PID captures the torque induced by the tension force from the wire. This is because when in a stable hover, the net torque acting on the quadrotor is zero. The tension force from the catenary model, as well as the integral term of the quadrotor is recorded. The tension force is mapped into the quadrotor's coordinate frame and a multi-linear model is used to fit. For one of our quadrotor setups we obtained

$$T_p = 2177.69F_z - 642.87F_y - 16.89 \tag{6.4}$$

$$T_r = 740.48F_y + 1963.97F_x + 6.51. \tag{6.5}$$

The intercepts for equations 6.4 and 6.5 correspond to the trim of the quadrotor, a small constant offset applied to a control in order to make an aircraft fly correctly. The trim

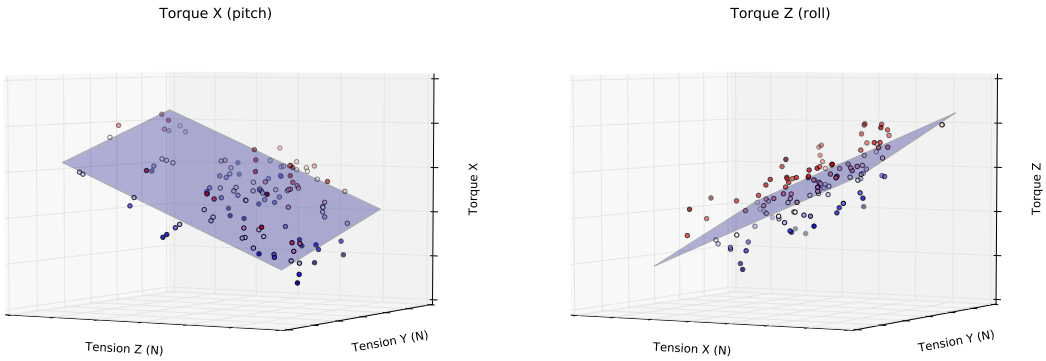

Figure 6–4: Torque Model trained for one of the quadrotors. Prediction errors are colour coded, with darker red corresponding to larger positive errors, and darker blue larger negative errors.

encapsulates many unknowns such as differences in thrust generated by the motors and asymmetries in the distribution of the mass. If it were possible to use a massless tether, the integral of the on-board PID should be constant and approach these values. The brush attached to the quadrotor for stippling is responsible for the intercept in Equation 6.4 for having a large negative value of -16.89.

With the multilinear torque model, the torque acting on the quadrotor can be predicted from the tension force and sent to the on-board controller. The multilinear model does not account for all the variability, the coefficient of determination is only 0.613 and 0.764 for roll and pitch respectively. However we use the model in combination with the on-board PID which results in faster convergence for the integral term and overall better control. The improvement in control is most noticeable when quickly traveling between two far away points, since the torque acting on the quadrotor will vary drastically.

The main disadvantage of this method is that it is specific to the current setup of the quadrotor. For different configurations, such as changing the brush length, marker

placement, or different quadrotor, the process of learning the linear torque model must be repeated. However, predicting the torque and including a feed-forward term to the on-board PID drastically improves the stabilization time of the on-board PID resulting in improvements in overall control.

In addition to the usual control commands, the torque computed for the pitch and roll are sent to the on-board controller. The on-board controller is then modified to make use of these values when computing the commands to send to the motors. The integral term of the on-board PID controller for the pitch and roll is modified and computed as

$$k_I \left(T + \int_0^t e(t)dt \right), \tag{6.6}$$

where k_I is the integral coefficient, T is the torque, and $\int_0^t e(t)dt$ is the accumulated error in orientation over the course of the flight.

CHAPTER 7
Results and discussion

We have created a number of prints using our system, as well compared experimentally the differences between tethered and untethered flight. We present these results in this chapter, along with details on the implementation of our system, its limitations, and possible improvements.

7.1 Implementation details

We perform real time motion capture with Optitrack cameras and Motive software. Motive tracks the position and orientation of objects tagged with retro-reflective markers, specifically, the canvas and the robot. We use 12 cameras attached to the ceiling in a square pattern and facing the middle of the room, providing a cube-shaped capture volume of about 2 meters in each direction. A Python program reads motion capture data and sends commands to the robot, but also communicates to a second Python process which computes and updates the planned stipple points. The software running on the Crazyflie is modified to improve control of the brushed motors and improve radio communication in the presence of dropped packets. Table 7–1 provides a list of parameters we use in our implementation.

7.2 Physical prints

Figure 7–2 shows examples of physical prints compared to their source images and planned stipples (see also the result shown in Figure 1–1). The accompanying video shows the process of creating these prints, including high speed video of a single stipple, the

Parameter	Value
Robot mass	29.4 g
Battery mass	5.8 g
Tether mass	2.4 g
Tether length	184 cm
Max thrust	41.0 g
Speed control α	3300
Speed control β	2.8
Horizontal PID gains (k_P, k_I, k_D)	$(0.36, 0.03, 0.11)$
Vertical PID gains (k_P, k_I, k_D)	$(0.34, 0.05, 0.14)$
Stipple Controller horizontal PD gains (k_P, k_D)	$(0.4, 0.2)$
Stipple Controller vertical PD gains (k_P, k_D)	$(0.2, 0.1)$
Yaw rate PI gains (k_P, k_I)	$(3.0, 0.01)$
Control rate	100 Hz
Ink model	$1.033e^{-0.0263n}$
Max stipple area	0.8 cm^2
Min stipple area	0.18 cm^2
Canvas size	45 cm $\times$ 60 cm

Table 7–1: Values of parameters used in our aerial robot stippling implementation.

process of swapping batteries and reloading ink, and a time laps of progress in making a complete print. Currently, on average it takes several seconds to accurately place a single stipple. Flight time on a single charge can be as long as 6 minutes. For the earlier prints during this time we can draw up to 70 stipples. Conveniently this is roughly the maximum number of stipples producible before requiring a refill. Altogether, the time to create a print varies from about 10 minutes for the sphere, to approximately an hour for Che, Turing, and the teapot. Table 7–2 shows a summary of statistics related to the creation of prints. Finally, Figure 7–3 shows an example of a larger print in the process of stippled.

Image	t	μ_h	μ_v	σ_h	σ_v	n_f	$\bar{n}_s$
Che	6.7	1.70	-2.1	6.9	4.6	10	40
Turing	6.7	0.99	-4.1	6.6	3.8	8	62
Teapot	6.4	0.23	-3.1	7.9	4.2	10	50

Table 7–2: Print information for selected images, where t is the average time in seconds per stipple, μ_h and μ_v are the stipple error means in the horizontal and vertical directions in mm, σ_h and σ_v are stipple error standard deviations in the horizontal and vertical directions in mm, n_f is the number of flights, and $\bar{n}_s$ is the average number of stipples per flight.

7.3 Tethered flight comparison

Flight tests were performed to measure the quadrotor's ability to maintain a fixed position in order to understand better the variability of the hover controller position when flying with the tether. Each hover test was performed twice, once with the target position at a distance of 40 cm from the fixed endpoint of the tether, and once again but at a distance of 140 cm. In all tests the target position was set to be at an equal height to the fixed endpoint (90 cm). The x direction measures the horizontal error in the plane orthogonal to the plane the catenary curve lies, while the y and z directions indicate the vertical and horizontal error in the plane of the curve.

It is clear that modeling the tension of the tether results in improvements in control, as can be seen in Table 7–3, the standard deviation of the errors is smaller in almost all instances compared to the tests where tension is not modeled. For the results in Table 7–3, the σ_x error is reflective of errors introduced by cable oscillation. The more the cable sags, the worse the oscillation, which can be seen in the large error of the not dampened near compared to far tests. However dampening the tether mostly eliminates this discrepancy.

Finally comparing the σ_z illustrates the necessity of providing an estimate of the torque to the controller. When hovering far away, σ_z errors are larger. This has to do

Hover Tests	μ_x	μ_y	μ_z	σ_x	σ_y	σ_z
Control	-0.10	2.80	2.68	1.58	0.66	1.35
Near	-0.60	2.49	0.26	3.60	1.11	2.31
Near with Model	0.05	-0.18	1.07	3.79	0.44	1.75
Near and dampened	-1.02	3.14	1.76	2.77	0.45	2.04
Near with Model and Dampened	-0.67	-0.04	1.00	1.27	0.42	1.56
Far	0.59	1.86	3.80	2.10	1.13	5.87
Far with Model	0.41	-1.03	0.21	1.25	1.02	2.06
Far and Dampened	0.48	3.91	3.50	2.51	1.13	4.19
Far with Model and Dampened	0.12	-0.86	0.51	1.56	1.71	3.19

Table 7–3: Hover test results, where μ is the mean error and σ is the error standard deviation in cm. In these tests the controller is only accounting for the tension force of the catenary model, and ignoring the torque caused by the tension. The near and far tests were performed at a fixed height, 40 cm and 140 cm away from the endpoint respectively, in the z direction. The dampened tests uses a single thread tied to the midpoint of the tether to dampen oscillations.

with the torques being produced by the tension force. When the tension is pointing mostly down, little torque is produced since the cable is attached to the quadrotor at a point below its center of mass. When the robot is far away from the fixed endpoint, the horizontal tension component is larger and produces a greater torque on the quadrotor.

Table 7–4 shows the improvements in maintaining a stable hover that modeling the torque provides. The hover test was repeated at the distance of 140 cm away in the z direction. The hovering results of the quadrotor when using the torque model are comparable to those of the untethered control; it even out-performs the control with regards to the σ_x. This may be due to the dampening thread providing a stabilization effect on the side to side motion of the quadrotor. Most notable is the improvement of the σ_z error in comparison to the tethered test without modeling the torque. Almost all of the additional error introduced by the large torque the tether is exerting on the quadrotor has been accounted for.

Torque Hover Test	μ_x	μ_y	μ_z	σ_x	σ_y	σ_z
Control	-0.10	2.80	2.68	1.58	0.66	1.35
Tethered	0.12	-0.86	0.51	1.56	1.71	3.19
Torque	0.10	1.84	1.40	1.48	0.78	1.49

Table 7–4: The final hovering tests results. The control test is performed using battery powered flight rather than the tether. The tethered test is using a dampened tether and the controller is modeling the tension but not the torque. The torque test is also using a dampened tether, but is modeling torque as well as tension caused by the tether using the learned parameters as described by section 6.4.

In general, the results of tethered stippling is comparable to that of untethered. For a small trade-off in the accuracy of stipple placement and rate of stipples, what can be a very involved process of constantly refilling ink levels and swapping batteries becomes fully automated. Following initial setup of the canvas and the ink well, the quadrotor has completed over 800 stipples with no interference. Tethered stippling also seems to be no less reliable then untethered. The most frequent cause of failure throughout our experimentation is the radio disconnecting which occurs equally often in both cases. Tethered flight works best when the operating volume of the quadrotor is small. This works well for stippling on smaller fixed size canvases. For other applications, tethered flight may be a more significant restriction. For example, the navigation of objects as in work by Landry [10] would be impossible.

As a final evaluation metric a stippling test to compare the tethered model to the untethered was performed. The stippling test consists of the quadrotor repeatedly colliding with the canvas at set target locations a total of 120 times. Errors are computed as the difference from the target location and the projection of the quadrotor's sponge onto the

Tethered Stippling	t	μ_h	μ_v	σ_h	σ_v
Control	3.8	0.97	-1.1	7.2	4.2
Tethered	4.5	1.05	0.3	7.2	4.5

Table 7–5: Stippling test results, where t is the average time in seconds per stipple, μ_h and μ_v are the stipple error means in the horizontal and vertical directions in mm, and σ_h and σ_v are stipple error standard deviations in the horizontal and vertical directions.

canvas at the time of collision. For the tethered test, the tether was dampened and both tension and torque estimates were provided by the catenary model to the quadrotor. Tethered tests without the model were also performed, but due to the lower fidelity control, only a few stipples at most could be performed before the quadrotor would crash; or that reason these results are not included in Table 7–5. As can be seen in Table 7–5, tethered stippling performs very similarly to untethered stippling, with the only prominent difference being a slightly slower stippling rate. This is likely because the catenary model does not account for the dynamics of the tether as the quadrotor travels between points.

7.4 Discussion and limitations

Figure 7–1 demonstrates the hover controllers ability to maintain a target position. We measure an average position error of just over 2 cm when controlling the robot to hover at a point in space. We find the quality of the hover controller to be acceptable, but the magnitude of the error is not ideal for stippling. Fortunately our stipple controller can reliably produce stipples with a much lower error. As reported in Table 7–2, the standard deviation in horizontal and vertical directions is typically closer to half a centimeter. This is possible because the quadrotor only proceeds to the stippling stage when the error in the hover controller is below a certain threshold.

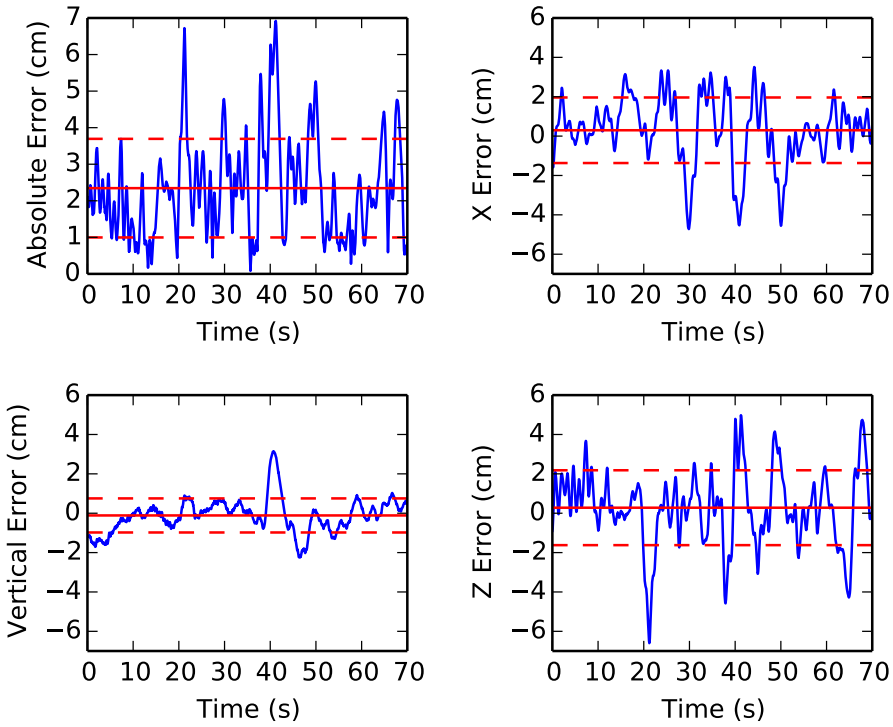

Figure 7–1: Error in the quadrotor's position while trying to maintain a stable hover. From left to right, the error magnitude and the x (orthogonal to tether plane), y (vertical), and z errors. Solid and dotted red lines show the mean and standard deviation respectively.

While we do not adjust stipple positions to better represent edges or salient features as proposed by previous work [12, 19], we briefly experimented with adjusting our robot control to improve accuracy for certain stipples, such as those that make up the eyes of a face or those that lie along a sharp edge. However, this has little benefit because it involves an increase in flight time to wait for the robot to match the necessary position and velocity to initiate the placement of a high precision stipple with fractional improvement in the accuracy of the placement.

All of our results are printed on flat surfaces, but we note that it would be straightforward to stipple on curved surfaces. For the robot to successfully draw stipples, the curvature of the surface would need to be limited. The current position of the stippling brush is best suited to vertical surfaces, so it would be necessary to change the orientation of the brush to apply stipples on non-vertical surfaces (e.g., on an overhang or ceiling). Finally, computing stipple positions on a manifold rather than a plane is an interesting problem to explore in future work.

While the previous work on robot drawing and painting makes use of visual feedback with cameras, we rely solely on motion capture and our ink model to estimate the position and size of stipples. We expect that our results could be improved by using visual feedback. Visual feedback would not only provide more accurate feedback on the placement of the stipples, but on the size of the stipple as well. The dynamic update for stipple placement could then use both placement feedback and size feedback when computing future positions. We also expect the results would improve by controlling the orientation of the Crazyflie at the point of contact. Other obvious improvements would be to perform a more sophisticated system identification of the Crazyflie, as done by Landry [10]. Finally, note that some of our final results have artifacts from running ink when too much is applied to a given area. Improvements to our ink model, the shape of the brush, and velocity control could all help better control stipple sizes and prevent these artifacts.

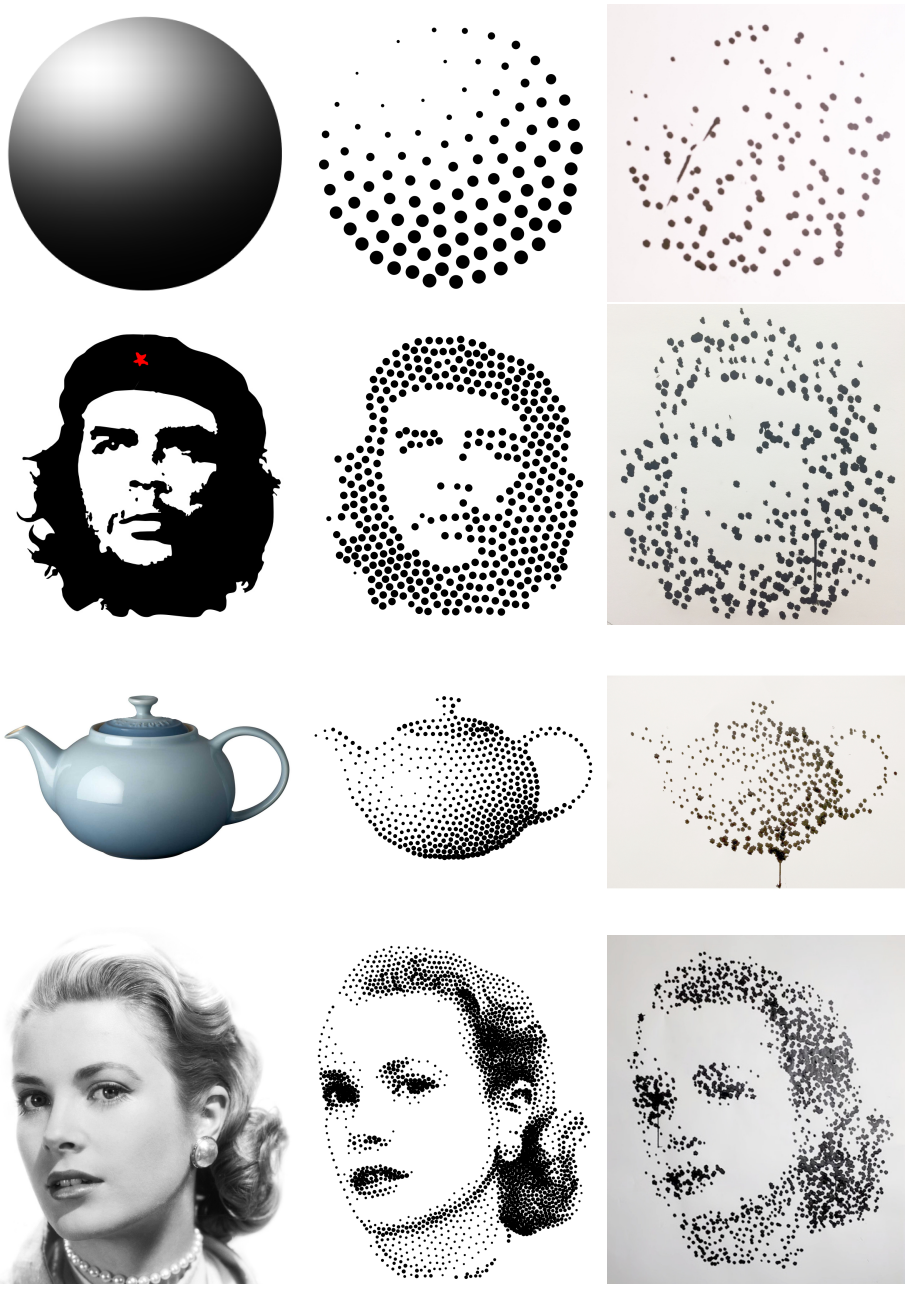

Figure 7–2: Example results, comparing original image, planned stipple positions, and the result of stippling with the flying robot. Stipple counts for the sphere, Che, teapot, and Grace are 100, 400, 500, and 2000 respectively.

Figure 7–3: A photo of our largest print with 2000 stipples in the process of being drawn on a 100 cm $\times$ 70 cm canvas.

CHAPTER 8
Conclusion

We present a technique for creating stippled prints with a flying quadrotor robot. This involves commanding the under-actuated robot to fly to different positions, and control in the presence of a contact. We describe in detail the low level details, including state estimation, latency issues, PID control, radio communication, and parameter tuning. We also describe the high level algorithmic aspects involved in creating a set of stipples for an image, adjusting their positions in the presence of errors, and the issues in computing a good order for stipple creation. We present a method for fully autonomous stippling using a tether to provide power in order to not be limited by the capacity of batteries. We describe a model for the tether and how the existing controller can be easily extended to account for additional forces and torques acting on the quadrotor.

8.1 Future work

While the results chapter discusses limitations and some simple extensions to our work, there are a variety of other exciting related avenues for future work. Although we have eight robots in our fleet, we only use one at a time for stippling. When creating a larger print it would be advantageous to coordinate multiple robots to reduce the total printing time. The acrylic ink we use is available in dozens of colors, which opens up interesting computational challenges to adjust for color in addition to stipple positions, as well as color blending. Figure 8.1 shows an early result experimenting with multiple drawing passes, each using a different color ink. Moving beyond stipples, for instance

Figure 8–1: A colored print of the joker, done in 3 drawing passes each using a different colored ink. From left to right, the source image, the physical print, and a close up of two over-lapping stipples illustrating subtractive color blending.

using an airbrush, would be very interesting and could exploit existing work on optimizing ink transfer in creating murals [21].

By extending the methods outlined and applying them to larger quadrotors, stippling of surfaces outdoors would be possible. While the crazyflie quadrotor can fly outside, due to its light weight the accuracy of its stipple placement would be decreased by any breeze. This opens up a range of possibilities from stippling hard to reach locations, to creating large murals.

Light painting with aerial robots also has a collection of unique and interesting computational challenges. Figure 8–2 shows an experimental result where we directly apply the robot control algorithms that we use for stippling to light painting. In this example, the cube is drawn with a Neopixel, with the total flight path partially revealed by the additional light trails left by the robot's battery and communication status lights. Light painting with flying robots can benefit form dynamic updates similar to those we present here, while new control strategies can be designed to relax flight trajectories in directions that project to the same point in the image.

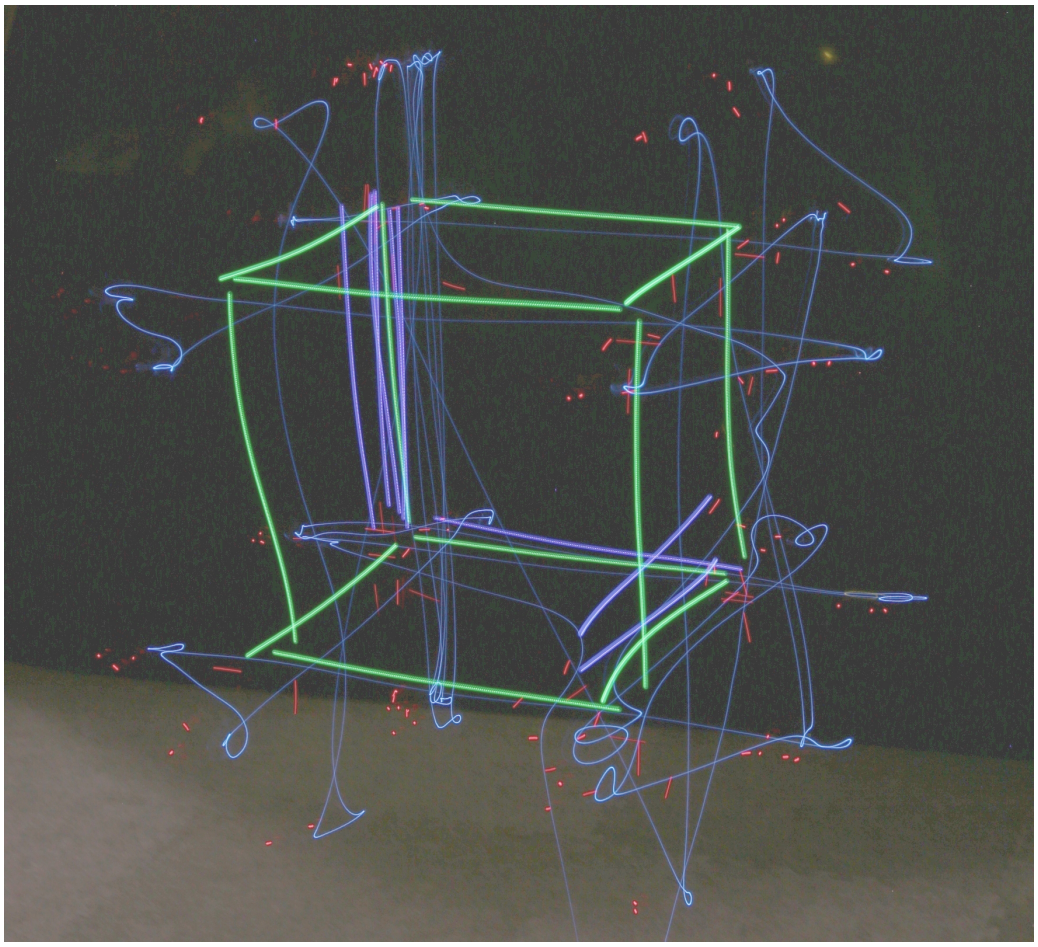

Figure 8–2: Long exposure photo of light painting of wire frame cube.

References

[1] Daniel Berio and Frederic Fol Leymarie. Computational models for the analysis and synthesis of graffiti tag strokes. In *Proceedings of the Workshop on Computational Aesthetics*, CAE '15, pages 35–47, 2015.

[2] Robert Bosch and Adrianne Herman. Continuous line drawings via the traveling salesman problem. *Operations Research Letters*, 32(4):302–303, July 2004.

[3] C. de Crousaz, F. Farshidian, M. Neunert, and J. Buchli. Unified motion control for dynamic quadrotor maneuvers demonstrated on slung load and rotor failure tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2223–2229, May 2015.

[4] Oliver Deussen, Thomas Lindemeier, Sören Pirk, and Mark Tautzenberger. Feedback-guided stroke placement for a painting machine. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, CAe '12, pages 25–33, 2012.

[5] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia. Learning swing-free trajectories for uavs with a suspended load. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4902–4909, May 2013.

[6] Wolfgang Hoenig. Towards crazyswarms, April 2015. https://www.bitcraze.io/2015/04/towards-crazyswarms-2/.

[7] S. Hota and D. Ghose. Optimal path planning for an aerial vehicle in 3d space. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 4902–4907, Dec 2010.

[8] Craig S Kaplan, Robert Bosch, et al. TSP art. In *Renaissance Banff: Mathematics, Music, Art, Culture*, pages 301–308. Canadian Mathematical Society, 2005.

[9] James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings.*

ICRA'00. IEEE International Conference on, volume 2, pages 995–1001. IEEE, 2000.

[10] Benoit Landry. *Planning and control for quadrotor flight through cluttered environments*. PhD thesis, Massachusetts Institute of Technology, 2015.

[11] Jurg Lehni and Uli Franke. Hektor. Diploma project at ECAL (Ecole cantonale d'art de Lausanne), 2002.

[12] Hua Li and David Mould. Structure-preserving stippling by priority-based error diffusion. In *Proceedings of Graphics Interface 2011*, GI '11, pages 127–134, 2011.

[13] Chyi-Yeu Lin, Li-Wen Chuang, and Thi Thoa Mac. Human portrait generation system for robot arm drawing. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pages 1757–1762, July 2009.

[14] Thomas Lindemeier, Jens Metzner, Lena Pollak, and Oliver Deussen. Hardware-based non-photorealistic rendering using a painting robot. *Computer Graphics Forum (Eurographics)*, 34(2), 2015.

[15] Thomas Lindemeier, SÃűren Pirk, and Oliver Deussen. Image stylization with a painting machine using semantic hints. *Computers & Graphics*, 37(5):293 – 301, 2013.

[16] Yan Lu, Josh HM Lam, and Yeung Yam. Preliminary study on vision-based pen-and-ink drawing by a robotic manipulator. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pages 578–583. IEEE, 2009.

[17] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648. IEEE, 2010.

[18] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):0278364911434236, 2012.

[19] David Mould. Stipple placement using distance in a weighted graph. In *Proceedings of the Third Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'07, pages 45–52, 2007.

[20] Hans Pedersen and Karan Singh. Organic labyrinths and mazes. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 79–86, 2006.

[21] Romain Prévost, Alec Jacobson, Wojciech Jarosz, and Olga Sorkine-Hornung. Large-scale painting of photographs by interactive optimization. *Computers & Graphics*, 2015.

[22] Adrian Secord. Weighted Voronoi stippling. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 37–43, 2002.

[23] K. Sreenath, N. Michael, and V. Kumar. Trajectory generation and control of a quadrotor with a cable-suspended load - a differentially-flat hybrid system. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4888–4895, May 2013.

[24] Manohar Srikanth, Kavita Bala, and Fredo Durand. Computational rim illumination with aerial robots. In *Computational Aesthetics (Expressive 2014)*, 2014.

[25] P. A. Tresset and F. Fol Leymarie. Sketches by Paul the robot. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, CAe '12, pages 17–24, 2012.

[26] Patrick Tresset and Frederic Fol Leymarie. Portrait drawing by Paul the robot. *Computers & Graphics*, 37(5):348 – 363, 2013.

[27] L. Zikou, C. Papachristos, and A. Tzes. The power-over-tether system for powering small uavs: Tethering-line tension control synthesis. In *Control and Automation (MED), 2015 23th Mediterranean Conference on*, pages 681–687, June 2015.