Computational Aesthetics in Graphics, Visualization, and Imaging
Angus Forbes and Lyn Bartram (Editors)

Stippling with aerial robots

B. Galea and E. Kia and N. Aird and P. G. Kry

School of Computer Science, McGill University, Canada

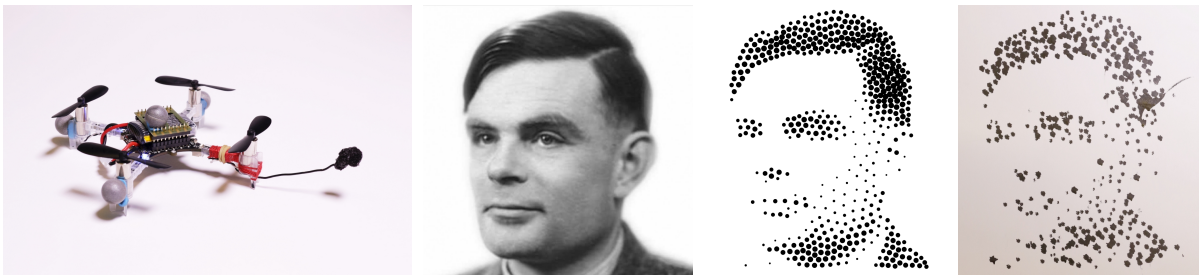

Figure 1: *Our stippling aerial robot uses motion capture markers for positioning, and an ink sponge on a small arm to create stipples. Also shown is an example result: from left to right, a well known image of Alan Turing, a stippled version of the image with 500 dots, and a finished print with 500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

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. We evaluate our dynamic adjustment of stipple locations with synthetic experiments. Stipples per second and variance of stipple placement are presented to evaluate our physical prints and robot control performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.2.9 [Artificial Intelligence]: Robotics—Commercial robots and applications

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 [TL13,LMPD15]. In this paper, 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 left) because they are 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, which makes them much safer than larger quadrotors.

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 underactuated. 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 (e.g., demos by Ascending Technologies, and Spaxels at Ars Electronica). Quadrotors have likewise recently been used for producing rim illumination for photography [SBD14]. However, to the best of our knowledge, we are the first to address the computational issues of painting with autonomous aerial robots.

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 [TL12, TL13]. In earlier work, Lin et al. [LCM09] use a camera and a humanoid robot to draw a line drawing portrait of the person in view. In similar work, Lu et al. [LLY09] 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 [DLPT12], image stylization with semantic hints [LPD13], and dynamic adjustment of layered strokes [LMPD15]. 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. [BL15] design a curve representation suitable for creating and analyzing graffiti tags. Similarly, Evan Roth and colleagues at the Free Art and Technology Lab have created a *Graffiti Markup Language* for use in analysis of graffiti, and for programing industrial robot arms to create tags. Lehni developed a system called Hektor [LF02], 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 [Sec02], 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 [Mou07]. In more recent work, Li and Mould describe how error diffusion allows for reduced stipple counts while preserving structure of an original image [LM11]. 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, as we do in this paper, 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 [PS06]. Similarly, approximate solutions to the traveling salesman problem have been used to produce continuous single-stroke drawings [BH04, KB*05]. 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 [KL00, HG10]. Furthermore, many control problems specific to quadrotors have been investigated, such as methods to produce ag-

gressive maneuvers [MMK12] and flips [LSSD10]. The robot control algorithms we implement in this work is largely inspired by that of Mellinger et al. [MMK12], as well as the PhD thesis of Landry [Lan15].

3. 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. Both of these methods rely on feedback on the position 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.

3.1. 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 [Lan15]. 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,

$$u = \frac{V_{\max}}{V_{\text{actual}}}\left(\sqrt{\omega^2 - \beta}\right) + \alpha, \tag{1}$$

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.

3.2. 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. 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 following controllers compute the desired pitch, roll, yaw rate, and thrust and rely on the Crazyflie's internal PID to achieve the desired orientation.

3.3. 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,

$$F_{\text{net}} = k_P(x_t - x) + k_I \int (x_t - x) + k_D(\dot{x}_t - \dot{x}), \tag{2}$$

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 (i.e., 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. Note that we control the yaw of the quadrotor with a separate P controller that calculates the yaw-rate.

When preparing to draw a stipple, 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.

3.4. Stipple controller

The stipple controller is a PD controller that 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 is always in a direction parallel to the canvas.

The act of stippling consists of three stages, preparation, stippling and recovery. During the preparation stage, the quadrotor attempts to stabilize around a point a set distance away from the canvas, such that it is normal to the target stipple location. The preparation stage uses the hover controller to approach and maintain its target location. 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. 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 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. Control transitions to the next preparation stage once 500 ms have past and the location of the next stipple is known.

3.5. 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.

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.

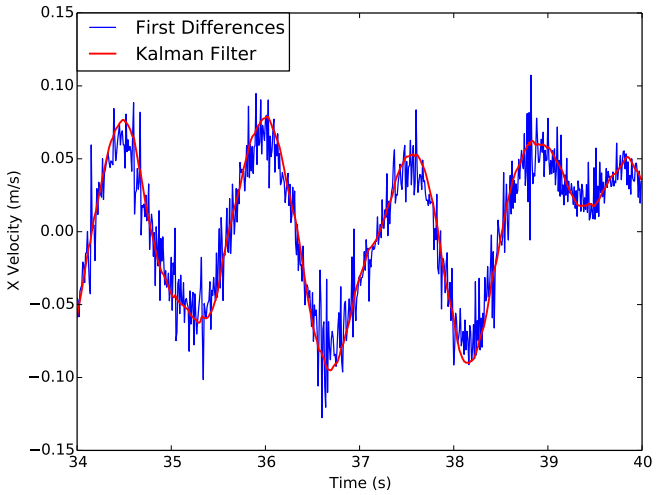

Figure 2: *The Kalman filter provides a smooth estimation of the quadrotor's velocity*

3.6. 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

$$\begin{aligned} \dot{x} &= \dot{x}_F + \Delta T \ddot{x}_F \\ x &= x_F + \Delta T \dot{x} \end{aligned} \tag{3}$$

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 3.5. Figure 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.

3.7. Radio communication improvements

The default communication protocol provided by the Crazyflie is not optimal for real-time control in environments with interference. 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. 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.

4. 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.

4.1. Weighted centroidal Voronoi diagrams

The core of our stippling algorithm is based on weighted centroidal Voronoi diagrams (CVD), as described by Secord [Sec02]. 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,

$$r_i = \rho_i r_{\min} + (1-\rho_i) r_{\max}, \tag{4}$$

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.

4.2. 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 3 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 3. 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.

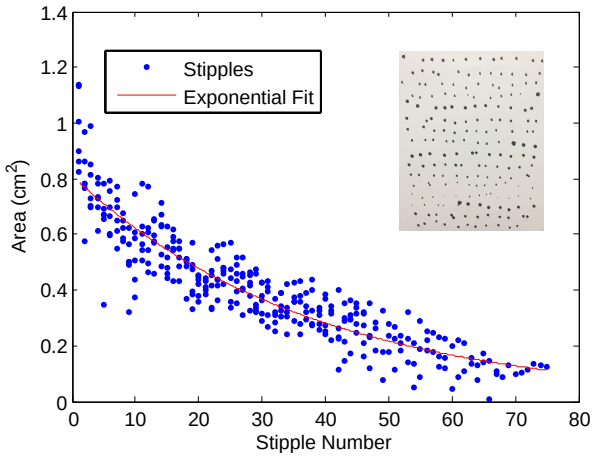

Figure 3: *Stipple size decay model fit to six sequences of ink transfer tests (example shown as inset image). The area of each stipple is measured in source images, from which we fit a two parameter exponential model.*

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).

4.3. Stipple order and dynamic updates

While the static set of stipples generated by the algorithm in Section 4.1 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. 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 4.2. 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 (see Section 5.2 for examples and evaluation of our dynamic update in synthetic examples).

Originally, we implemented an approximate solution to the traveling salesman problem using a generic algorithm. This was slow and did not work well with the adjustment of point positions as required for dynamic updates for errors. 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.

5. Results and discussion

We have created a number of prints using our system, and we have evaluated our method with synthetic tests. We present these results in this section, along with details on the implementation of our system, its limitations, and possible improvements.

5.1. 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 1 provides a list of parameters we use in our implementation.

Parameter	Value
Robot mass	35.4 g
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)	$(0.34, 0.05, 0.14)$
P gain for yaw rate	3.0
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 1: *Values of parameters used in our aerial robot stippling implementation.*

Image	n	3‰	5‰	10‰	20‰
Sphere	100	18.2%	20.7%	13.3%	23.7%
Teapot	250	22.2%	33.3%	38.0%	31.4%
Che	500	26.7%	28.6%	31.9%	16.6%
Turing	750	54.5%	31.3%	33.8%	26.8%
Grace	1000	45.2%	33.6%	26.8%	4.1%

Table 2: *Percent difference between the mean inter-stipple distance of the static versus dynamic case compared to ground truth, for various images of different stipple counts n, and for different per-mil error sizes.*

5.2. Dynamic stipple adjustment evaluation

Dynamically updating points allows us to achieve better spacing on stipples under a Gaussian error model. To measure how well a given drawing performs, we plot histograms of the inter-stipple distance (Figure 4). In the histograms, we can see that the dynamic method give results which perform somewhere half way between the ground truth stipple statistics, and the distribution produced for a naive static stippling without dynamic correction for errors.

Table 2 shows how much the mean of the inter-stipple distance shifts between the static and the dynamic case across different images and error sizes. We examined other summary statistics, such as minimum, maximum, and standard deviation, but did not observe any discernible patterns. The best mean improvements appear to be achieved around the 5 to 10 ‰ error range, which is what the error on our quadrotor is on a 50 cm canvas. Note that the improvement of the mean from dynamic corrections appears to become less important when the errors are large, particularly for larger stipple counts (see Grace at 20 ‰ in Table 2). Figure 5 shows a synthetic comparison of how the dynamic correction effects the final result for a range of different placement error variances.

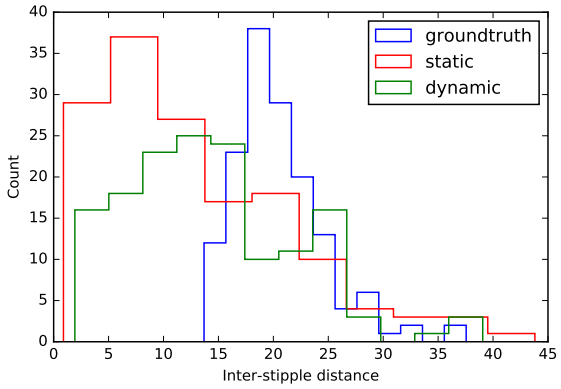

Figure 4: *Histogram of the inter-stipple distance for the original output without errors (blue), the static stippling with error (red) and the dynamic error adjusted stippling (green). The histogram is for a stippling of a teapot with 200 points and 1% standard deviation Gaussian error.*

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 3: *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.*

5.3. Physical prints

Figure 6 shows examples of physical prints compared to their source images and planned stipples (see also the result shown in Figure 1). The accompanying video shows the process of creating these prints, including high speed video of a single stipple, the 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, during which time we can draw up to 70 stipples. 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 3 shows a summary of statistics related to the creation of prints. Finally, Figure 7 shows an example of a larger print in the process of stippled.

5.4. Discussion and limitations

Figure 8 shows a plot of position error over time. 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 3, the

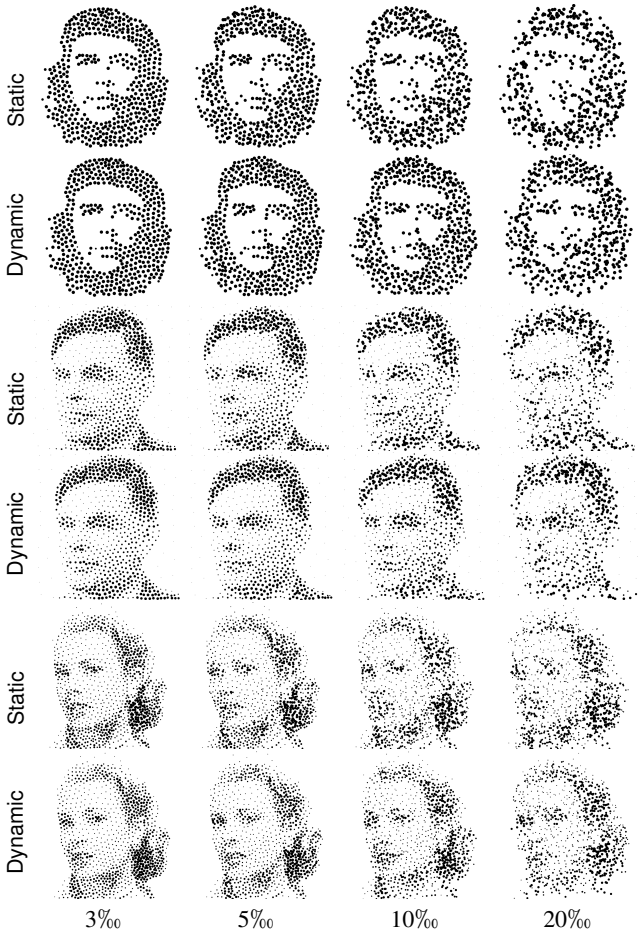

Figure 5: *Comparison of synthetic results showing stipples drawn with and without dynamic stipple placement correction for different amounts of error. Error standard deviation of stipple locations in different columns is shown as a per-mil of the canvas size.*

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.

While we do not adjust stipple positions to better represent edges or salient features as proposed by previous work [Mou07, LM11], 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 edges. However, this has little benefit because it involves a significant 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.

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 sur-

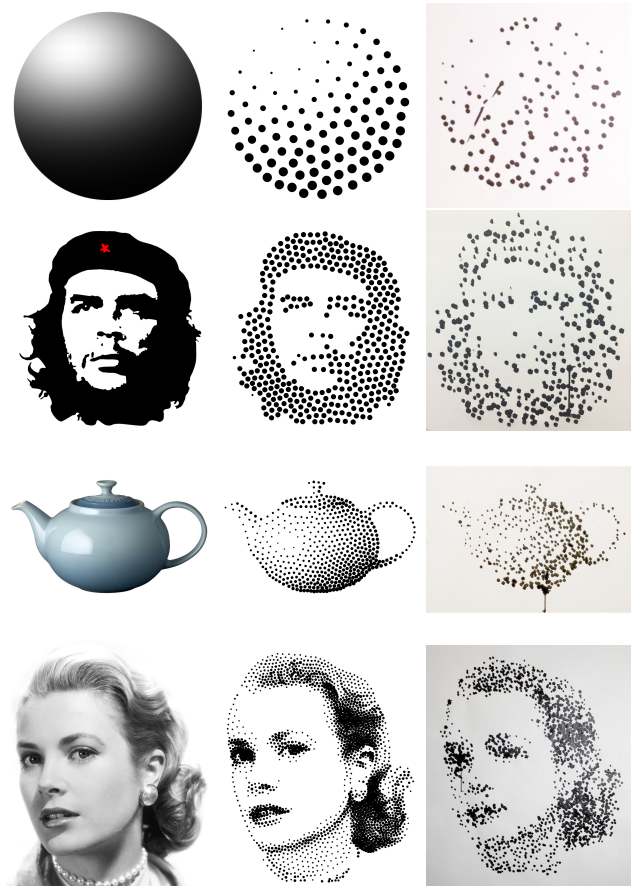

Figure 6: *Example results, comparing original image, planned stipple positions, and the result of stippling with the flying robot. Stipple counts for the sphere, Che, teapot, and Grace are 100, 400, 500, and 2000 respectively.*

Figure 7: *A photo of our largest print with 2000 stipples in the process of being drawn on a 100 cm × 70 cm canvas.*

faces (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. 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 in other work [Lan15]. 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.

6. Conclusions and future work

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 have presented printed results, and evaluated errors produced by our technique for a collection of synthetic results.

While the results section 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. We would also like to modify these robots to use inductive charging stations so that they can autonomously recharge and refill their ink sponges, allowing print creation without any human intervention. 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. Moving beyond stipples, for instance using an airbrush, would be very interesting and could exploit existing work on optimizing ink transfer in creating murals [PJJSH15].

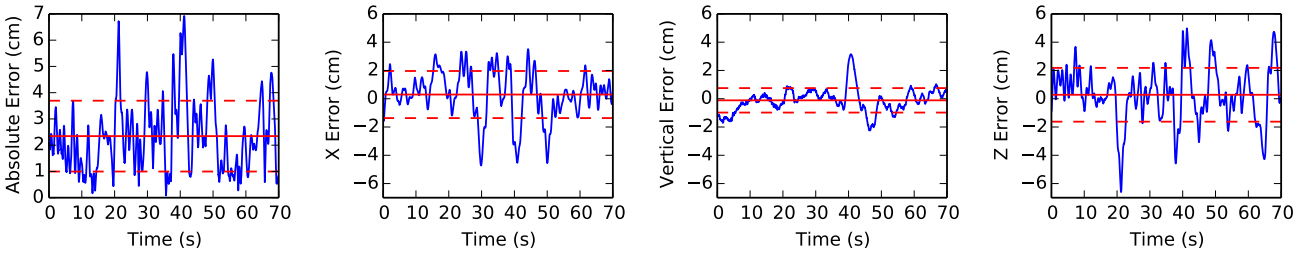

Figure 8: *Error in the quadrotor's position while trying to maintain a stable hover. From left to right, the error magnitude and the x, y (vertical), and z errors. Solid and dotted red lines show the mean and standard deviation respectively.*

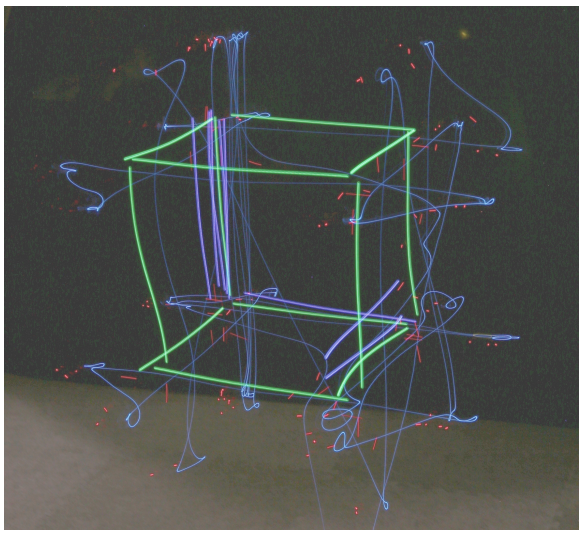

Figure 9: *Long exposure photo of light painting of wire frame cube.*

Light painting with aerial robots also has a collection of unique and interesting computational challenges. Figure 9 shows a preliminary 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.

Acknowledgements

This work was supported by funding from NSERC, FRQNT, and CFI. We also thank Karan Singh for useful discussions.

References

[BH04] BOSCH R., HERMAN A.: Continuous line drawings via the traveling salesman problem. *Operations Research Letters 32*, 4 (July 2004), 302–303. 2

[BL15] BERIO D., LEYMARIE F. F.: Computational models for the analysis and synthesis of graffiti tag strokes. In *Proceedings of the Workshop on Computational Aesthetics* (2015), CAE '15, pp. 35–47. 2

[DLPT12] DEUSSEN O., LINDEMEIER T., PIRK S., TAUTZENBERGER M.: Feedback-guided stroke placement for a painting machine. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (2012), CAe '12, pp. 25–33. 2

[HG10] HOTA S., GHOSE D.: Optimal path planning for an aerial vehicle in 3d space. In *Decision and Control (CDC), 2010 49th IEEE Conference on* (Dec 2010), pp. 4902–4907. 2

[KB*05] KAPLAN C. S., BOSCH R., ET AL.: TSP art. In *Renaissance Banff: Mathematics, Music, Art, Culture* (2005), Canadian Mathematical Society, pp. 301–308. 2

[KL00] KUFFNER J. J., LAVALLE S. M.: RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on* (2000), vol. 2, IEEE, pp. 995–1001. 2

[Lan15] LANDRY B.: *Planning and control for quadrotor flight through cluttered environments*. PhD thesis, Massachusetts Institute of Technology, 2015. 3, 8

[LCM09] LIN C.-Y., CHUANG L.-W., MAC T. T.: Human portrait generation system for robot arm drawing. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on* (July 2009), pp. 1757–1762. 2

[LF02] LEHNI J., FRANKE U.: Hektor. Diploma project at ECAL (Ecole cantonale d'art de Lausanne), 2002. 2

[LLY09] LU Y., LAM J. H., YAM Y.: 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* (2009), IEEE, pp. 578–583. 2

[LM11] LI H., MOULD D.: Structure-preserving stippling by priority-based error diffusion. In *Proceedings of Graphics Interface 2011* (2011), GI '11, pp. 127–134. 2, 7

[LMPD15] LINDEMEIER T., METZNER J., POLLAK L., DEUSSEN O.: Hardware-based non-photorealistic rendering using a painting robot. *Computer Graphics Forum (Eurographics) 34*, 2 (2015). 1, 2

[LPD13] LINDEMEIER T., PIRK S., DEUSSEN O.: Image stylization with a painting machine using semantic hints. *Computers & Graphics 37*, 5 (2013), 293 – 301. 2

[LSSD10] LUPASHIN S., SCHÖLLIG A., SHERBACK M., D'ANDREA R.: A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on* (2010), IEEE, pp. 1642–1648. 3

[MMK12] MELLINGER D., MICHAEL N., KUMAR V.: Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research 31*, 5 (2012), 0278364911434236. 3

[Mou07] MOULD D.: Stipple placement using distance in a weighted graph. In *Proceedings of the Third Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging* (2007), Computational Aesthetics'07, pp. 45–52. 2, 7

[PJJSH15] PRÉVOST R., JACOBSON A., JAROSZ W., SORKINE-HORNUNG O.: Large-scale painting of photographs by interactive optimization. *Computers & Graphics* (2015). 8

[PS06] PEDERSEN H., SINGH K.: Organic labyrinths and mazes. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering* (2006), NPAR '06, pp. 79–86. 2

[SBD14] SRIKANTH M., BALA K., DURAND F.: Computational rim illumination with aerial robots. In *Computational Aesthetics (Expressive 2014)* (2014). 2

[Sec02] SECORD A.: Weighted Voronoi stippling. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering* (2002), NPAR '02, pp. 37–43. 2, 5

[TL12] TRESSET P. A., LEYMARIE F. F.: Sketches by Paul the robot. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (2012), CAe '12, pp. 17–24. 2

[TL13] TRESSET P., LEYMARIE F. F.: Portrait drawing by Paul the robot. *Computers & Graphics 37*, 5 (2013), 348 – 363. 1, 2