Leonardo Da Robot

Authors: Eric Chang: Electrical and Computer Engineering, Carnegie Mellon University
Christopher Bayley: Electrical and Computer Engineering, Carnegie Mellon University
Harsh Yallapantula: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of taking a digital image and painting a watercolor version on a physical canvas. The aim of the project is to create an image which looks naturally painted, not a replica of the source image. At minimum, this product will be able to represent simple images or match the shape of the given image. The resulting image should be similar in terms of color gradients and general appearance and contrast. The system will consist of a 2-dimensional gantry system to physically draw the picture along with a software component that will handle image processing.**

***Index Terms*— Gantry, Image Processing, Painting, Robot**

I. Introduction

One of the biggest goals in robotics is to create systems that behave more human-like. A large amount of research today is focused on creating robotic systems that replicate human tasks such as driving, speech comprehension and vision. Our intent is to break down this barrier between humans and robots in art. Our design is inspired by the gantry systems of 3D printers, which we combine with image processing and control systems to make a robot that can receive an image and paint it on a canvas. The most difficult part of this is to make the painting look natural and not robotic, while also maintaining accuracy and speed. This project does not aim to replicate how a printer makes an image pixel-by-pixel. To accomplish this task, we paint the picture using a paint brush and human-like strokes. Our approach takes the proven efficiency and accuracy of 3D printers to allow us to paint a pre-processed image using smooth and natural strokes.

Critical to our design is the ability to paint an image which is an accurate representation of the source image. This is measured using the structural similarity (SSIM) index, where we aim for a score of at least 0.2. Additionally, the total time for painting an image should be reasonable to the scale and complexity of the image, limited to 8 hours in the worst case. These metrics are key for our project to meet our goals of connecting robotic systems with art through a design which produces a painted image of good quality and can do so in a reasonable amount of time.

II. Design Requirements

The first requirement is that a digital input image of any size is capable of being rendered as the target image to paint and displayed back to the user. This step allows for an image to be rejected if the rendered painting is not of acceptable quality. This requirement is purely digital in nature and therefore can be tested using an image bank of 20 images which vary in image size and complexity. The successful design will be able to create renders of consistent quality across all scales and complexities, and will scale images which exceed the bounds of the painting space. This requirement will simply be measured by the program's success at processing the given input image.

Our next requirement is that we effectively use the full range of the palette which is integrated into the system. When painting an image the palette color with the lowest difference to the desired color should be chosen, measured using the HSV (hue, saturation, value) color model. Specifically, this color difference will be measured using the following equation:

$$\Delta = (|H_0 - H_r| + |S_0 - S_r| + |V_0 - V_r|) \quad (1)$$

In this equation, Δ is the color difference, H_0 is the hue of the original color from the image, H_r is the hue of the color chosen by the robot, S_0 is the saturation of the original color from the image, S_r is the saturation of the color chosen by the robot, V_0 is the value of the original color from the image, and V_r is the value of the color chosen by the robot. This requirement can be tested using an image of our own palette; the pigment in the palette image should always be painted with that pigment. Other images that contain basic color samples, such as the one shown in Fig. 1, can also be used, in which case (1) will be utilized. Together, these images will be used to verify that the correct color is used.

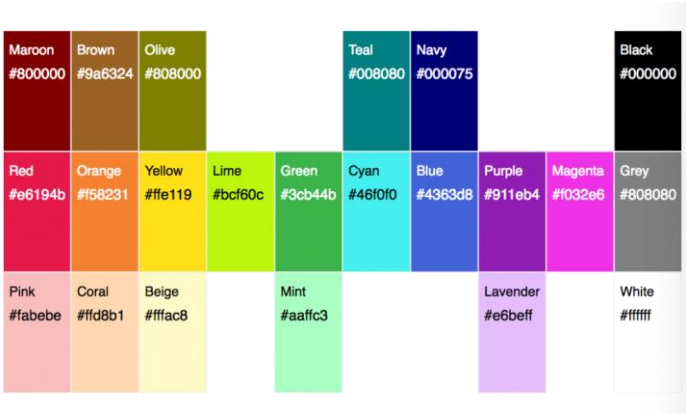


Fig. 1. An example of a basic color sample image which will be used to test our robot's color selection.

We require that our output painted image has a score of at least 0.2 according to the structural similarity (SSIM) index. This is a metric which is used to measure the perceived similarity of an image to the source and is commonly used for film and television **[4]**. We use this metric for this reason as it corresponds closely to the goal of our project. After testing the SSIM for various images, we found that professionally done water painting images of customer supplied images was roughly 0.4 on average. We will test our performance using an image bank of 10 images, shown in Fig. 2, which features images which grow in complexity. Using this bank, we will paint the images and test that the SSIM score is at least 0.2 for the first 8 images. The final two images are more complicated and are our stretch goals once we reach the scores for the first 8.

Our final requirement is that our design operates in a reasonable amount of time as a function of image size and complexity, which we define using the following equation:

$$t = (1 + \boldsymbol{\alpha}) * (\boldsymbol{\beta} * s) \quad (2)$$

In this equation, t is the estimated time, **α** is a measure of image complexity, measured as 4 * (uncompressed image size / JPEG compressed size), s is the size of the painted image in square inches, and **β** is a constant scaling factor of 4 (translating to 4 minutes to paint each square inch). The equation for **α** follows the logic that JPEG compression uses DCT-II coefficients to compress the image from a raw uncompressed size of the total image size and dimensions, meaning that the JPEG compressed size of an image gives rough estimate of how much information is contained in an image **[5]**. The image size painted can be variable, but is limited to 7.5x10 inches, and from experimental testing **α** can be up to 0.70 for a very complex landscape image, and as low as 0.04 for a single line. Equation (2) gives a rough estimate of the upper limit of the testing time, and is designed to have a limit of roughly 8 hours for the largest and most complex image. We will test our design using the same image bank from Fig. 2, which contains variably sized images as well, and will ensure that they all print in a time corresponding to (2).

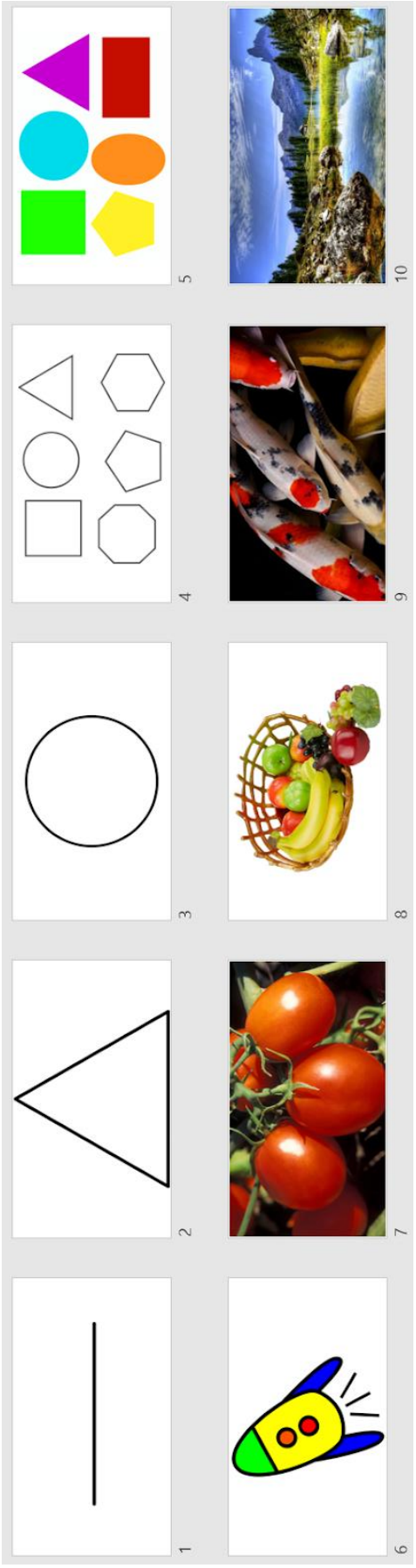


Fig. 2. The test image bank we will use in testing.

III. Architecture and/or Principle of Operation

Our project will primarily be split into a physical hardware component and a software image processing component. The overall architecture of our project is represented in the block diagram depicted in Fig. 3. The physical component will be the robot itself, which includes the base and frame that supports the 2D gantry system. The gantry system is supported by a frame that lies on the base of the robot. The gantry system's purpose is to move a painting head around the XY coordinate space above the paper, water, and palette which are lying on the base. The painting head will control a paintbrush which is attached by a servomotor, and the head will be moved around by two stepper motors. These motors will be controlled by a Raspberry Pi. The servomotor is connected to the Raspberry Pi through a GPIO pin, while the stepper motors are controlled using the I^2C interface which is provided by a Raspberry Pi motor shield.

The software component of our project centers on using image processing to convert a digital input image into instructions for our motors in order to successfully paint an image. A monitor will be connected to the Raspberry Pi which will allow the user to submit the digital image they wish to be painted. The monitor will also be used to display the result of the image segmentation process to the user, in order to show the user what the approximate final result of the painting will be. The input image will go through our image processing algorithm, which will first modify the image to be easier to paint, in the process of image segmentation. This process will reduce details and cluster colors together to create a simpler image for painting, which is what will be displayed to the user on the monitor. The image segmentation process will output data and information such as edges and colors, which will then be used in our stroke creation algorithm. This process will generate the sequence of strokes that our robot will need to perform in order to paint the image, with information such as color, length, and direction encoded.

This sequence of strokes will be given to the stroke routine, which will control the motors. The motor shield for the Raspberry Pi that we are using comes with a library of motor controls, which we will use in our stroke routine algorithm. A list of strokes with a specified color and location are provided to the stroke routine, and the routine will instruct the motors of our robot to dip the brush into the water, paint, and finally onto the paper. Our motor control routines are the bridge between the software containing information of what to paint and the hardware which is capable of painting. The interface from the lowest level of motor drivers allow the stepper motors to rotate a set number of steps in a set delay time between each step. The servomotor is controlled through PWM, and its driver has an interface that allows the motor to be set to a certain angle. These interfaces are used by the control routines to abstract to higher level concepts such as dipping the brush to grab pigment or performing a stroke at set locations. These control routines receive input from the painting and stroke generation algorithms.

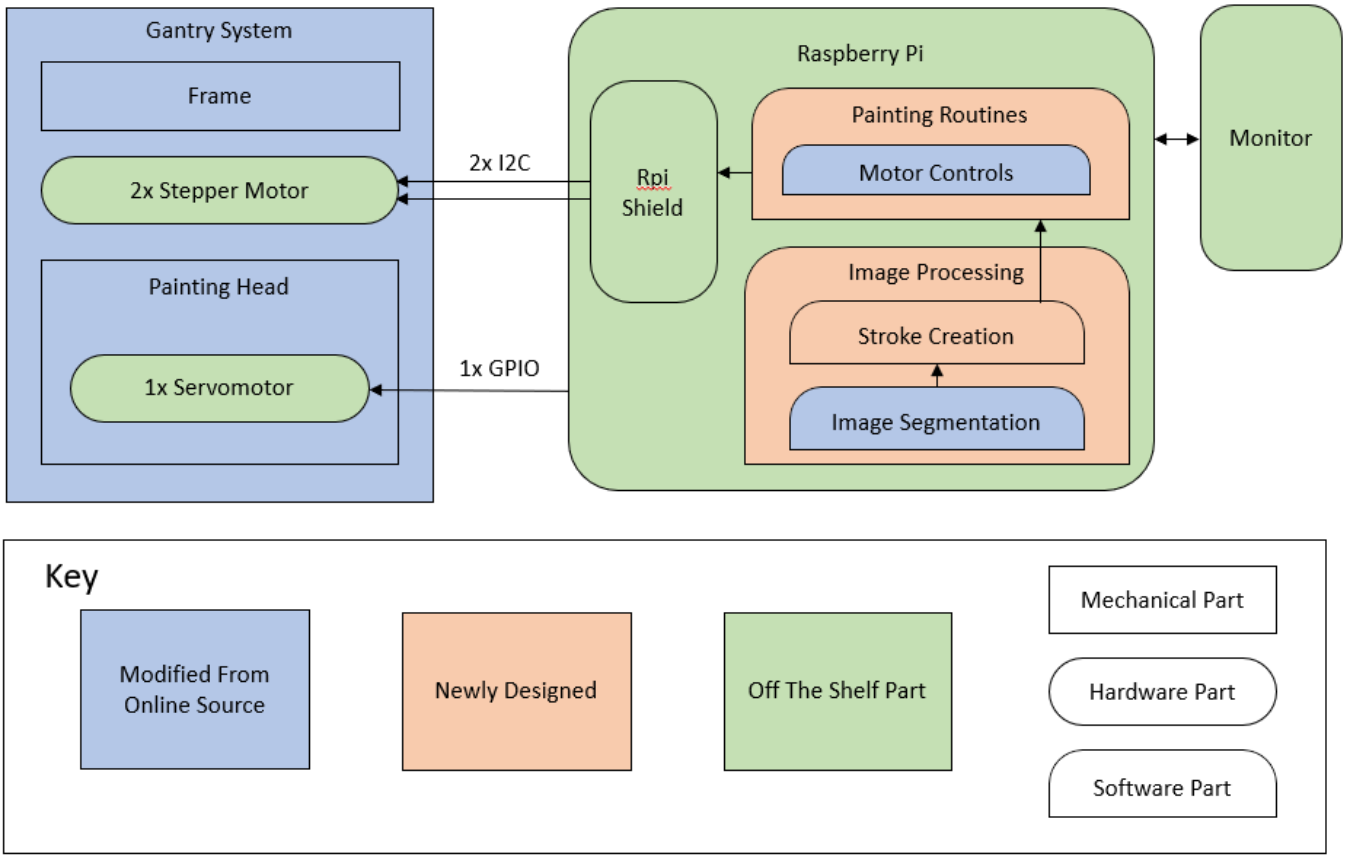


Fig. 3. The block diagram showing our project's archiecture and subsystems.

IV. Design Trade Studies

In designing our project, there were numerous trade-offs and options to consider when finalizing decisions. Two of the most important aspects of our design were the 2D gantry system, which is the primary physical and hardware component of our robot, and the image segmentation algorithm, which is the first step of our software component. A significant amount of testing, research, and experimentation was conducted in order to choose the best option to proceed with in these two subsystems.

A. 2D Gantry System

There were several possible implementation paths available for designing the 2D gantry, and all would meet the major constraints of our project which are having a large enough workspace and a fine enough granularity. Because of this, the main considerations made when choosing the gantry design were complexity, cost, and risk. Our original plan was to use threaded rods which would support the carriage, allowing rotation of the threads to carry the device. We found that these designs were generally used for much larger constructions, as it can support a much heavier payload. For this reason, we chose to use a belt-based design as is commonly used in 3D printers, as these designs are commonly used for lighter loads and there are plenty of reference implementations available in the form of 3D printers.

When considering possible belt-based gantry designs, we found some of the most common designs are the Hbot system (depicted in Fig. 4), CoreXY (depicted in Fig. 5), and the Ultimaker style **[2]**. The Hbot uses a single very long belt arranged in the shape of an H, but the force applied is uneven and results in a moment created on the edge of the print head. The CoreXY system solves this problem using two belts arranged in an H shape which cross near the top. This design is fairly complicated, and the exact arrangement of the belts and pulleys required leaves a lot of room for error. Based on these factors, we chose the design used by the Ultimaker printers. This design uses several sets of belts which run in one direction and is almost entirely symmetric, making it simpler to implement. Additionally, Ultimaker is open source, allowing us

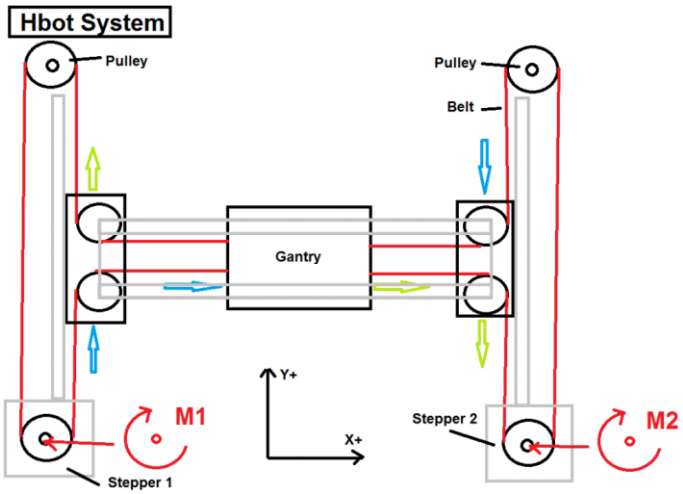


Fig. 4. The Hbot gantry design.

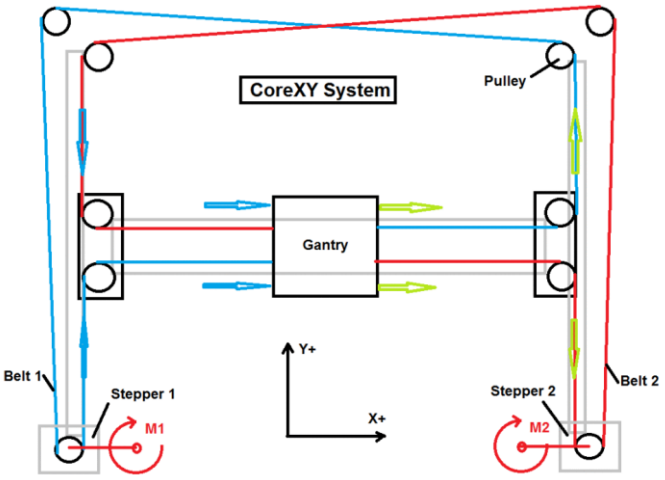


Fig. 5. The CoreXY gantry design.

to leverage their proven system with small modifications to work in our own design. We selected the gantry specification based on the simplicity of the Ultimaker style and the low risk due to the open-source nature of Ultimaker.

The final aspect of our design decision with respect to the gantry was to use existing parts or design our own. By designing and 3D printing our own parts, we allow for a more custom design, fast part availability, and an overall lower cost than purchasing the Ultimaker parts. The drawbacks are that this allows room for error by modifying an existing design which is known to work well. The Ultimaker parts were not available from any major retailers in the US, and additionally the replacement part packs were expensive and contained many parts we wouldn't need. For these reasons we chose to design our own parts and have them 3D printed. This design choice has allowed us to save over $100 of our budget, and allowed us to expand the working space of the head considerably. Inevitably the print head would have to be a custom part as well, and by modifying all of the other parts we reduced the number of constraints on the geometry of the print head which allowed us to pursue a clean and simple design.

Overall, our gantry design decision diverged first between a threaded rod or belt design, between several different belt designs, and finally on using custom or premade parts. We made the choice of a belt-based design for its prevalence in an area which is very similar to our goal, allowing us to keep risk low and manage complexity by using well documented existing designs. The Ultimaker design was chosen among other belt-based designs again for its simpler design and proven effectiveness, which greatly lower our risk. Finally, the decision to modify the parts came at the price of a small increase in risk for the benefit of a significantly less constrained design space and lower cost.

B. Image Segmentation Algorithm

There were a few methods we considered for implementing an algorithm to pre-process the image. The reason that we needed to pre-process was because of the limitations of the physical system. The brush width must stay constant throughout the entire painting process, and thus we cannot draw something

thinner than this width. The second limitation is that we have 24 colors available. This means that all colors have to be approximated to those 24 colors. The third limitation is time. Since this is a physical system where motors have to move everything around, there are time constraints. To tackle these constraints, we considered the following pre-processing algorithms: blurring, edge drawing, k-means image segmentation, and mean shift segmentation. After some consideration, we decided to discard the idea of blurring. The original image used to test all of these image segmentation algorithms, depicting a fruit basket, is shown in Fig. 6. The result of blurring it is shown in Fig. 7.

Although blurring removes the detailed parts of the image, it still has gradients, which are very difficult to produce. It also becomes difficult to see where one object ends and another begins. Therefore, we decided to further explore drawing only the edges and image segmentation. To choose between these three methods, we calculated the accuracy of the output along with how complex the output image was. For accuracy, we looked at the output image and used the Structural Similarity Index to classify the methods. For complexity, we took the output JPEG file and compared it with the original JPEG file in terms of file size. As stated earlier, the JPEG file format compression is correlated to how complex an image is, so the more compressed the output file is, the less complex it is **[5]**. Table 1 shows the complexity and similarity for the three methods.

The conclusions we can draw are that the k-means clustering and mean shift clustering are far better than edge drawing in both accuracy and complexity. Mean shift clustering is 27% less complex than k-means clustering but still manages to beat out k-means in terms of accuracy. It also returns objects of defined edges and uniform color, which is much easier for the physical system to draw than what k-means returns. The following images show the output of edge detection, k-means clustering, and mean shift segmentation. Fig. 8 shows the output of the original image after running edge detection, Fig. 9 shows the result of running k-means, and Fig. 10 shows the result of running mean shift segmentation. All these images use the original fruit basket image in Fig. 6 as input.

TABLE I. STRUCTURAL SIMILARITY AND COMPLEXITY RATIO OF DIFFERENT IMAGE SEGMENTATION ALGORITHMS

	SSIM	Complexity Ratio
Edge detection	0.0335	1.8982
Mean shift	0.9432	0.7654
k-means	0.9352	1.0471

Fig. 6. The original image of a fruit basket used in our image processing experimentation.

Fig. 7. The result of blurring the image in Fig. 6.

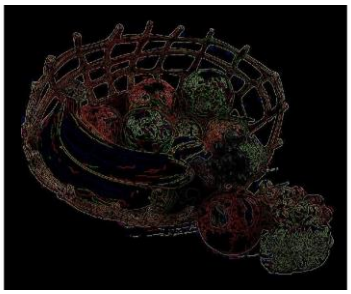


Fig. 8. The result of running edge detection on the image in Fig. 6.

Fig. 9. The result of running k-means on the image in Fig. 6.

Fig. 10. The result of running mean shift segmentation on the image in Fig. 6.

V. System Description

Our project is split into three main subsystems: the 2D gantry system, which is the physical component that controls movement of the brush; the gantry control layer, which controls the motors of the gantry; and image processing and stroke generation, which is the software component of our project.

A. 2D Gantry System

The gantry system forms the physical portion of the project. This is a system of shafts, pulleys, and motors which allow the robot to dip a paintbrush in water, collect pigment from a palette, and perform a stroke on the paper. The design for our gantry follows the gantry design used in the Ultimaker line of 3D printers, which is proven to be effective and accurate. Ultimaker is also entirely open source, and we were able to modify their CAD files as well as design our own which match our design more precisely. Our design does not use the Ultimaker parts, although they are available for purchase online through 3D printer repair sites. This is because by modifying the Ultimaker designs or creating our own we are able to keep costs low by 3D printing these parts, as well as make custom parts which match our design requirements more precisely. For example, the carriages were redesigned to fit an axes arrangement which was simpler for us to execute, as well as to carry the print head with the crossbar shafts in a different arrangement than the original design.

The core of the gantry design is outlined in Fig. 11. Two stepper motors are located at one corner of the gantry and can be positioned anywhere in the z plane below the axes. Two sets of parallel axes are positioned to form a square, mounted in a bearing allowing the shafts to rotate easily. Both the stepper motors and the axes have a pulley around them, and are connected by a timing belt, labelled purple. This translates the rotation of the stepper motor to the axes. Each set of parallel axes are connected by two belts as well, labelled red and blue, which coordinate their rotations. A carriage rests on the axes and is attached to the belts, which allows the rotational motion of the opposite axes to be transferred into translational motion on the carriage. The carriages carry shafts which support a paint head in the center space, allowing the head to be coordinated in two dimensions. The paint head carries a servomotor which has a brush connected, allowing the brush to be raised from the page or lowered to make contact. This is suspended via a frame of T-slotted aluminum, allowing the paint head to travel a fixed distance above the base. The implementation of our design is shown in Fig. 12.

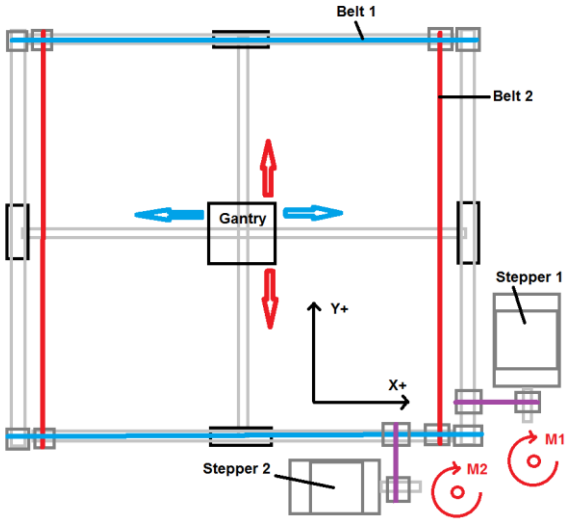


Fig. 11. The Ultimaker gantry design.

The major constraints on this system are that the brush head have a movable space which is large enough and that the brush can be controlled with sufficient precision. The working space must fit a standard sheet of paper, 8.5x11 inches, our watercolor palette, about 3.5x8.5 inches, and a cup of water, about 2x2 inches. This totals a working space requirement of 12x12 inches. Our design consists of a frame of aluminum with inner dimensions of 16x16 inches. The entire inner dimension is not usable however, due to the space required within the frame for the axes to run uninterrupted. Considering the space required for the axes, as well as the pulleys and carriages mounted on the axes, the dimensions of the working space are 14x14 inches. Then this meets our requirement for the brush to be able to move over a sufficiently large area. Additionally, the mounting height of the entire gantry is adjustable, but fixed during operation. This allows us to modify the height to fine tune the amount of contact the brush makes with the paper as it is rotated by the servomotor. This is the advantage of using T-slotted aluminum as well as our own designed mounts for the axes.

To address the precision allowed by this design, we must consider the precision of the stepper motors and the dimensions of the pulleys and timing belts. The stepper motors step size translates to a rotation of the axes by the same amount, as they are coupled by a timing belt. The rotation of the axes will result in a translational movement of the belt which carries the carriage according to the degree change and the radius of the pulley. From this we can form the equation $x = 2\pi r * (\theta / 360)$, where x is the translation of the carriage, r is the radius of the pulley, and θ is the change in angle of the axes. The pulley used in the Ultimaker, which we use in our design with a small change to the inner radius, has a radius of 0.25 in. Our stepper motors have 200 steps / revolution, which translates to a 1.8

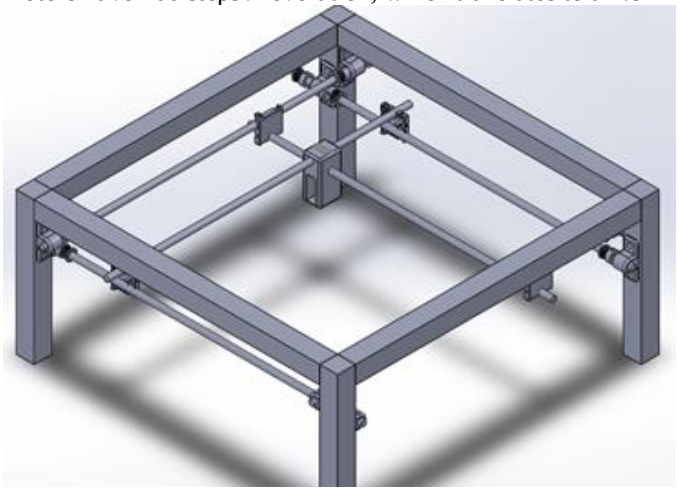


Fig. 12. A CAD model of our gantry system.

degree step size. Of all the cost-effective stepper motors we considered, this step size was standard. This gives us a minimum horizontal translation $x = 2\pi * (0.25 \text{ in}) * (1.8 / 360) \approx 0.008$ in. or 0.2 mm. This means we can operate the paint head with a granularity of 0.008 inches, which is more than sufficient to carry out all the requisite strokes and operations.

The integration of the gantry into our overall design is at the motors. The two stepper motors are connected to a motor shield mounted on the Pi, and the servomotor are connected directly to the Raspberry Pi. Low level drivers are used for basic control of these devices, allowing their rotations to be orchestrated into an operating gantry system.

B. Gantry Control Layer

The gantry control layer consists of the motors that will control the 2D gantry system as well as the software in the Raspberry Pi that controls the motors. The code for this subsystem will be written in Python, and will use libraries to help interface the motors with the Pi. The servomotor, which controls the painting head, will be wired to the Raspberry Pi through a GPIO pin. We will be using the library gpiozero to interface with the servomotor, specifically the AngularServo class, which extends the Servo class **[6]**. This class allows control of a rotational PWM-controlled servomotor, and gives us the ability to set it to specific angles. For the library to work, we must set the servo to its maximum position and its minimum position and measure its angles, and input these angles as the constructor for an instance of the class. This setup will let us move the servomotor to any angle in between.

The two stepper motors will be connected through the motor shield on the Pi using the I^2C interface. The motor shield comes with its own library to control the motors, which is available through GitHub **[7]**. The library allows us to control a stepper motor by moving it either forward or backward, and defining the number of steps it moves and the time interval between each step. These functions will be used to manipulate the two stepper motors into moving the belts, which will rotate the rods that will translate the painting head around the 2D axis system. The library comes with the feature of letting the user define what the "forward" and "reverse" directions are on the motor without any rewiring.

Using these two libraries, an interface will be created for use by the painting routines. The gantry control layer will act as the intermediary between the software and hardware components of our project, allowing our code to easily call functions that will move the motors how we desire. This layer exposes a single interface which is the stroke routine, which can receive a list of xy coordinates which define a series of straight line segments and a number corresponding to one of the available colors in the palette. This function will then paint the entire stroke and return when completed. Doing so will require the paint head be moved to wash the brush, grab pigment from the palette, and trace the line segments on the page. It may be necessary to gather more water and more pigment while painting a single stroke, which is the responsibility of this layer to control. The layer above provides no information on how frequently to collect pigment.

In creating the stroke routine which is exposed to the layer above, routines for washing the brush and collecting a specific pigment are required. These routines are also a part of the motor control layer but are not exposed to the layer above. These are used internally by the stroke routine when it is necessary to wet the brush and collect pigment. As the locations of all of the needed objects to carry out these routines are fixed, much of the control routines will be moving to hard-coded locations, not operating by some feedback informing the head where the objects are. For this to work this layer must also always keep track of the precise location of where the head is. An additional challenge is that the coordinate systems used by the image must match the coordinate system used by the control layer. To address this the layer also exposes an initialization routine which will receive information from the above layer regarding the image size and desired output size, in inches. This information allows the future routines to normalize the pixels received and translate to its own coordinate system which is rooted in the physical space rather than a digital space.

This layer bridges the gap between the hardware and software in our design. Built up from the provided device drivers for the motors, this layer offers a single control routine for painting a desired stroke with a specified color. This abstraction allows the painting to be easily performed from the above layer after a list of strokes representing the image has been created.

C. Image Processing and Stroke Generation

The Image Processing and Stroke Generation algorithms form the software component of this project. The first part is the Image Processing algorithm. This converts the original digital image into something that the physical apparatus can draw. The reason this is required is that the physical part of the robot has 3 main constraints: brush width, number of colors, and time. The width of the brush is constant from the beginning to the end of the painting process, since we won't swap out brushes during a painting. We are using 24 colors, so the robot will have to select the closest color to the 24 colors. There is also a time limitation. Therefore an image clustering algorithm called Mean Shift Segmentation was used. This algorithm takes regions of similar color that are close together and turns them into segments of uniform color. Three parameters are used to toggle the properties of the segmentation algorithm. The first parameter is the distance from the center that a point can be such that it is still able to be part of that segment. This is called the spatial radius. The second parameter is the range radius, which specifies the range of colors that can be in the segment. The third parameter is the minimum density of points, which indicates the number of pixels that can be inside one segment, thereby dictating the size of the largest segment. These parameters ensure that the objects are wide enough to be drawn by the brush. This algorithm reduces the complexity of the image significantly (about 24%) while keeping the accuracy close to the original. The second benefit is that the algorithm outputs a map with the labeled regions for each pixel, which allows us to have objects with defined edges and uniform color. This makes it simple for the next part of the software

component, namely the Stroke Generation algorithm.

The Mean Shift Segmentation algorithm is an open source function implemented in the pymeanshift library **[8]**. By changing the parameters, we are able to segment an image in whichever way the problem requires. Fig. 13 shows the result of running light segmentation while Fig. 14 shows the result of running stronger segmentation.

The second part of the software component is Stroke Generation. Once the segmented image has been created, the segments are separated into lists of strokes. There are two different kinds of strokes: perimeter strokes and fill-in strokes. Each object in the segmented image is made up of one perimeter stroke and one fill-in stroke. The perimeter strokes trace the outline of an object with the color of that object. The fill-in strokes fill in the object with horizontal straight lines of that color. First the perimeter stroke is drawn for one object, and then that object is filled in with fill-in strokes. The order of objects to be drawn is based on the length of the perimeter stroke, to make sure that the lowest detailed objects are drawn first. A perimeter stroke is made up of several very small line segments which trace the outline of the object. A fill-in stroke is made up of several horizontal line segments which go from the top of the object to the bottom.

Once the strokes have been created and ordered, they are broken up into line segments which are defined by a starting coordinate, ending coordinate, and color. The color is chosen to be the closest color from the palette in terms of HSV values to the original color. This list of coordinates and colors is sent to the control layer.

Fig. 13. The result of light segmentation on the image in Fig. 6.

Fig. 14. The result of stronger segmentation on the image in Fig. 6.

VI. Project Management

A. Schedule

The schedule for the project is shown on page 10. Each task is color coded based on which team member primarily worked on it, and lines between tasks indicate which tasks depended on the completion of others.

B. Team Member Responsibilities

So far, we have all been focusing on the physical portion of the design. Since none of us have had any mechanical design experience before, we have decided to frontload that part. Chris has taken the lead on the mechanical design and has done most of the designing and ordering of the parts. Harsh has taken charge of the Software portion of the project. Eric has taken charge of the control layer of the robot. Chris will be helping out with the software and control layer since the mechanical portion of the construction will be completed first. As the schedule shows, we will all be sharing some responsibilities at the end.

C. Budget

The budget consisted of ordered parts and 3D printed parts. Refer to the Bill of Materials on page 11.

D. Risk Management

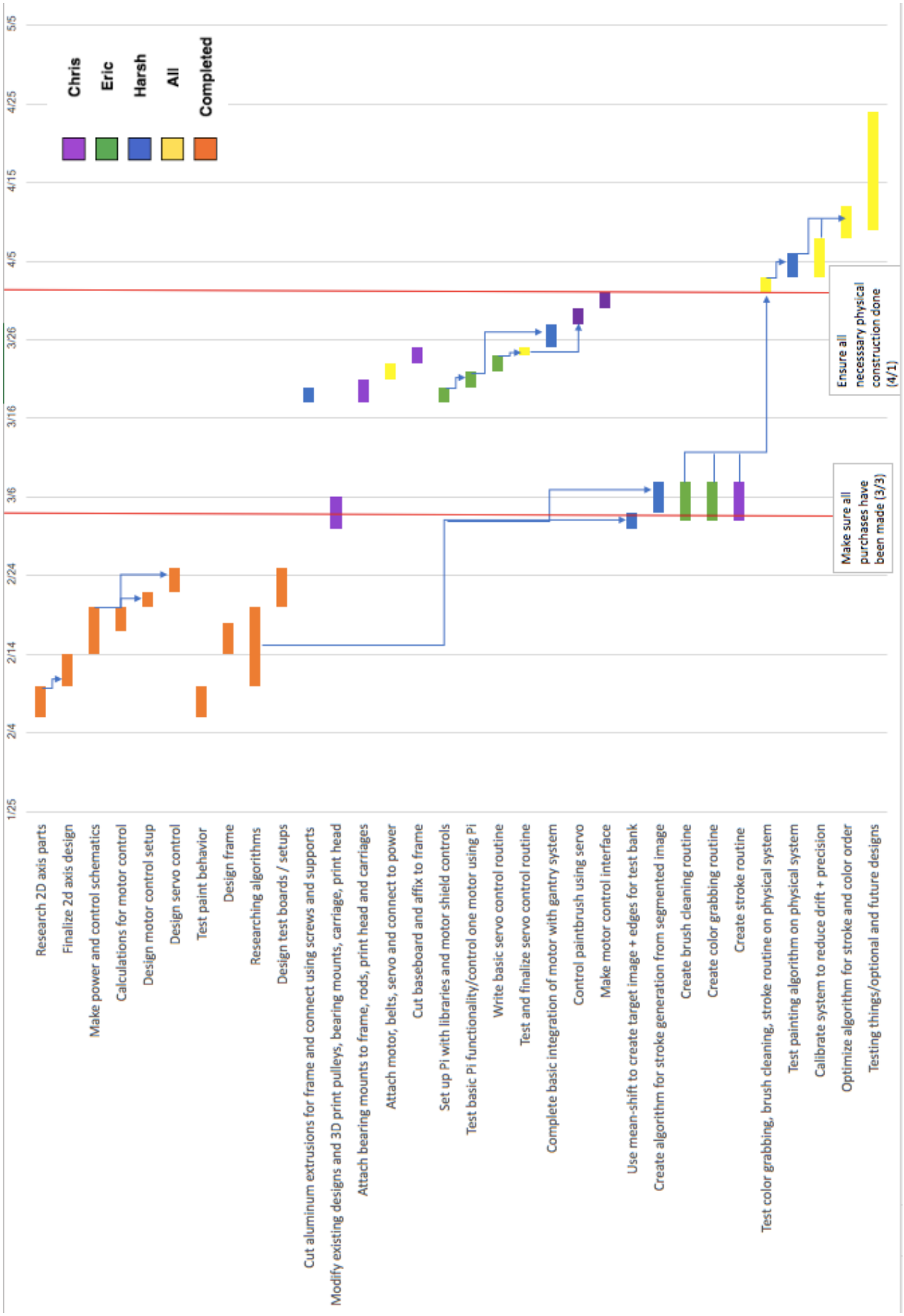
There are a few risks associated with this project. The first one is that the robot may not be able to paint the images accurately. We cannot test how accurate the robot will be able to paint until the physical construction and the control layer have been finished. In case the accuracy is undesirable and the image cannot be replicated, we have put in a fail-safe. The painting head can hold any type of writing or drawing instrument, so if water color painting is not accurate, we can always switch over to sketching with markers or pencil. Another risk is that the colors may not be accurate. In this case as well, we may switch over to sketching with a pencil. A final major risk factor is that the paintings take too much time. Either the motors may move the painting head too slowly or the paintings may just be too complex. We will make sure that the motors can move as fast as possible, and this shouldn't be very difficult since we already know that the similar design in the Ultimaker moves the head at 70 mm per second. If the painting is too complex, we will refine our image segmentation parameters to make the image less complex.

VII. Related Work

The goal of having an independent robot independently paint an image is not unique to our project; this is the same goal of the Robot Art Competition and Exhibition **[1]**. This is a competition held annually for robotics enthusiasts to submit images which were painted by their creations. From this competition and the gallery of designs they provide we were able to see how many of the best robots performed and how they were designed. A stand out in the competition is a man named Pindar Van Arman and his CloudPainter, which is an AI driven robot which has both a robotic arm based design and a gantry based design. His design is so effective that he has built a brand and sells the paintings his robot creates for a large sum. Most other designs featured on the Robot Art site use gantry-based designs, including another watercolor painting robot. These designs were our main source of confirmation that our idea for a gantry-based painting robot could be a success.

References

[1] https://robotart.org/
[2] https://maxdesign1990.wordpress.com/2016/05/22/gmtech-printer-motion-platform-research/ - gantry designs
[3] https://imiloainf.wordpress.com/2012/06/13/mean-shift-segmentation/ - Mean Shift Segmenttio
[4] http://www.imatest.com/docs/ssim/ - SSIM info
[5] https://en.wikipedia.org/wiki/JPEG - JPEG Compressions algo
[6] https://gpiozero.readthedocs.io/en/stable/api_output.html - gpiozero servo library
[7] https://github.com/sbcshop/MotorShield - MotorShield library
[8] https://github.com/fjean/pymeanshift - Pymeanshift library


Chris
Eric
Harsh
All
Completed
1/25
2/4
2/14
2/24
3/6
3/16
3/26
4/5
4/15
4/25
5/5
Research 2D axis parts
Finalize 2d axis design
Make power and control schematics
Calculations for motor control
Design motor control setup
Design servo control
Test paint behavior
Design frame
Researching algorithms
Design test boards / setups
Cut aluminum extrusions for frame and connect using screws and supports
Modify existing designs and 3D print pulleys, bearing mounts, carriage, print head
Attach bearing mounts to frame, rods, print head and carriages
Attach motor, belts, servo and connect to power
Cut baseboard and affix to frame
Set up Pi with libraries and motor shield controls
Test basic Pi functionality/control one motor using Pi
Write basic servo control routine
Test and finalize servo control routine
Complete basic integration of motor with gantry system
Control paintbrush using servo
Make motor control interface
Use mean-shift to create target image + edges for test bank
Create algorithm for stroke generation from segmented image
Create brush cleaning routine
Create color grabbing routine
Create stroke routine
Test color grabbing, brush cleaning, stroke routine on physical system
Test painting algorithm on physical system
Calibrate system to reduce drift + precision
Optimize algorithm for stroke and color order
Testing things/optional and future designs
Make sure all purchases have been made (3/3)
Ensure all necesssary physical construction done (4/1)

Bill of Materials

Part name	Quantity	Extra Quantity	Total Quantity	Price/unit	Base Price	Shipping/ Tax	Total Price	Notes
RPi	1	0	1	$34.49	$34.49	$0.00	$34.49	
RPi Shield	1	1	2	$14.99	$29.98	$0.00	$29.98	
RPi Power Adapter	1	0	1	$9.99	$9.99	$0.00	$9.99	
MicroSD with Adapter	1	0	1	$7.99	$7.99	$0.00	$7.99	
T-slotted Aluminum Beams	96	32	128	$0.23	$29.44	$7.80	$37.24	Price per inch, shipping/tax price is for 4 cuts at $1.95 each
Beam Caps	8	2	10	$1.20	$12.00		$12.00	Shipping/tax included in Beam T-Nut and Screw
Cap Push-In Fastener	8	2	10	$0.13	$1.30		$1.30	Shipping/tax included in Beam T-Nut and Screw
Beam Corner Brackets	8	2	10	$2.75	$27.50		$27.50	Shipping/tax included in Beam T-Nut and Screw
Beam T-Nut and Screw	8	2	10	$0.46	$4.60	$22.57	$27.17	Shipping/tax is for all of the 80/20 parts
Beam T-Nut and Screw (for bearing mounts)	16	0	16	$0.46	$7.36	$0.00	$7.36	Identical as above part, ordered separately
Stepper Motor	2	1	3	$9.63	$28.88	$0.00	$28.88	Nema 17 (from Ultimaker 2) (pack of 3 + cables)
Servomotor	1	1	2	$3.95	$7.89	$0.00	$7.89	Pack of 2
Servomotor extension cable	1	0	1	$9.99	$9.99	$0.00	$9.99	
Jumper Cables	1	0	1	$5.99	$5.99		$5.99	
12v 1A DC Power Supply	1	0	1	$8.95	$8.95	$8.94	$17.89	Shipping and tax includes cost for following part
Female DC Power Adapter	1	0	1	$2.00	$2.00		$2.00	
Paint Set	1	0	1	$10.29	$10.29	$0.00	$10.29	
Base Board	1	0	1	$10.69	$10.69	$4.69	$15.38	
Shafts	3	1	4	$4.67	$18.68	$7.12	$25.80	Each shaft will be cut into 2
Pack of Bearings	1	0	1	$24.95	$24.95	$4.98	$29.93	Contains 10 Bearings, 8 needed 2 extra
Motor Belts	1	0	1	$6.99	$6.99	$0.00	$6.99	2 belts per pack
Belts	1	0	1	$8.88	$8.88	$0.00	$8.88	5m of belt
Total Cost							$364.93	

Cost of 3D Printed Parts

Part Name	Quantity	Mass (g)	Total Mass (g)	Notes
.325 diameter pulley	1	2	2	This is a test part
.32 diameter pulley	10	2	20	
Stepper Motor .32 diameter pulley	2	2	4	
Bearing Mount	8	6	48	
Print Head	1	23	23	
Carriage Half	8	2	16	
Total Mass (g)			113	

Total Cost

	Cost
Hardware Parts	$364.93
3D Printed Parts	$34.50
Total Cost	$399.43
Budget Remaining	$200.57