

10/9/2013



PICASSAU

Ben Straub, David Toledo, Drew Kerr, Kayla Frost, Peter Gartland



## Executive Summary – Peter Gartland

PICASSAU is a robot designed to produce a simplified painting of a digital image. PICASSAU attempts to mimic a human artist by painting on an upright canvas. The paintbrush hangs from two motors at the upper corners of the canvas. By controlling these motors and servos attached to the paintbrush, PICASSAU can trace paths onto the canvas, dip the paintbrush into one of the paint reservoirs, and run a calibration routine to determine the paintbrush's position on the canvas. This level of functionality meets the Cycle 1 goal of being able to paint a single-color image from a supplied vector graphics file. PICASSAU makes use of several standards such as the SVG file format (page 9), XML (page 9), and an RS-232 serial communication protocol (page 18).

## Table of Contents

Introduction – Drew Kerr .....	1
Framing and Support – David Toledo .....	3
Motors – David Toledo.....	5
Motor Controllers – Ben Straub.....	6
Power Supply – Ben Straub.....	7
Graphics Type – Drew Kerr .....	9
Paintbrush Carriage Kinematics – Ben Straub .....	10
Software and Computing – Kayla Frost.....	14
Communication – Kayla Frost .....	18
Image Processing – Drew Kerr and Peter Gartland .....	20
Constraints – Peter Gartland .....	24
Budget – David Toledo.....	25
Timeline – Kayla Frost .....	26
Performance – Ben Straub.....	30
Conclusions – Kayla Frost.....	32
References .....	34
Appendix .....	35

## List of Acronyms

ADC	Analog to Digital Converter
FTDI	Future Technology Devices International
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
IR	Infrared
MDF	Medium-Density Fibreboard
OpenCV	Open Source Computer Vision
PVC	Polyvinyl Chloride
PWM	Pulse Width Modulation
RBBB	Really Bare Bones Board
RPi	Raspberry Pi
SVG	Scalable Vector Graphics
TTL	Transistor-Transistor Logic
UART	Universally Asynchronous Receiver Transmitter
USB	Universal Serial Bus
XML	Extensible Markup Language

## List of Figures

Figure 1	Diagram demonstrating the movement of the carriage during the calibration routine.	page 12
Figure 2	This figure outlines the communication process for sending a command from the RPi to the Arduino.	page 19
Figure 3	The original image.	page 20
Figure 4	The grayscale image.	page 21
Figure 5	The smoothed image.	page 21
Figure 6	Threshold system for assigning colors.	page 22
Figure 7	The thresholded image.	page 23
Figure 8	The final image that the robot will paint.	page 23
Figure 9	This Gantt chart outlines the timeline for the project.	pages 27-29

## List of Tables

Table 1	Pugh chart comparing the different framing material options.	page 3
Table 2	Pugh chart comparing the different motor options.	page 5
Table 3	Pugh chart detailing the advantages and disadvantages of the various power supply options.	page 8
Table 4	Pugh chart for determining which high-level programming language to use on the RPi.	page 15
Table 5	Pugh chart for determining which microcontroller to use to control hardware.	page 16
Table 6	Pugh chart for determining what kind of communication link to use between the RPi and Arduino.	page 18
Table 7	Budget, including estimated and actual costs.	page 25

## Introduction – Drew Kerr

For our project, we wanted to build a low cost robot that is able to paint pictures on a canvas taken by a webcam or uploaded onto a computer. We desired the painting process to be quick and efficient such that people could enjoy a quick self-portrait on the way to a football game or on an E-day tour without missing the game or falling behind on the tour. We also wanted the device to be easily visible during the painting process.

Therefore, we decided to create PICASSAU to meet these criteria. PICASSAU can accept an image by uploading it onto the computer connected to PICASSAU or by taking it with a webcam. The computer will then convert the uploaded or webcam image into a list of vectors and will send these vectors as a series of commands and coordinates to an Arduino. The Arduino moves the paintbrush by controlling two stepper motors at the top of the physical structure to match the coordinates and commands delivered by the computer. At the end of this process, PICASSAU will be able to generate a simplified yet magnificent painting from the original photograph.

The rest of this document describes the aforementioned process in more detail. It will detail the technical aspects of the project including the carriage kinematics, motor selection, frame structure, computer to Arduino communication, graphics file types and the power supply and microcontroller setup. In addition, the document discusses the standards and constraints involved in the project, the budget, the timeline and finally the performance of our robot thus far. The document will then conclude with our technical accomplishments and discuss the lessons learned in the first development cycle of PICASSAU. A complete outline of the remainder of the document follows.

*Outline*

- I. Technical Sections
  - a. Framing and Support
  - b. Motors
  - c. Motor Controllers
  - d. Power Supply
  - e. Graphics Type
  - f. Paintbrush Carriage Kinematics
    - i. General kinematics and motion control
    - ii. Calibration
    - iii. Curves
  - g. Software and Computing
    - i. Platform
    - ii. Computer Software
    - iii. Embedded Software
  - h. Communication
    - i. Communication Protocol
    - ii. Error Checking
  - i. Image Processing
- II. Constraints
- III. Budget
- IV. Timeline
  - a. Gantt Chart
- V. Performance
- VI. Conclusions
- VII. Appendix

## Framing and Support – David Toledo

The framing for the canvas and its support was constructed out of PVC. Our only other option we considered was to construct the framing out of 2X4 lumber. Table 1 shows the Pugh chart constructed to determine the better option. Using the Pugh chart we decided to construct the frame out of PVC. This also gave us the ability to use manufactured parts to help ensure the frame's corners and joints were square and level. The greatest plus from using PVC over lumber was the total weight of the structure, 2x4 lumber is more than twice the weight of PVC per unit length. The base of the frame was constructed to have a rectangular base, for stability while painting and moving, opposed to the general design of an easel, with four unconnected legs angled from the canvas to the floor.

Built into the frame are two platforms: one was built to support the paint containers and water and the second was built to support the power source, motor drivers, Arduino, and the RaspberryPi. The canvas support was constructed using MDF instead of plywood for two simple reasons: weight and cost. We also used MDF as the platforms to support the paint cups and the electronics.

Table 1: Pugh chart comparing the different framing material options.

		Framing Material options:	
Criteria:	Weight:	PVC	2X4 Lumber
Cost	2	++	0
Strength	3	+	++
Weight	4	+	-
Easy of use	2	++	+
+		15	8
0		0	2
-		0	4
Net Score		15	4

We decided to mount the motors to the frame above the canvas, as to have the painting carriage suspended parallel to the canvas. We decided on this design opposed to a horizontal design for two main reasons: one being the fact that a horizontal painter resembles a printer too much and the second being that we wanted the design to resemble a natural painter's posture. The motors are mounted approximately 1.5 feet above to canvas and approximately 1 foot away from the edges of the canvas horizontally.

## Motors – David Toledo

Our motor decision was a decision based on four variables: cost, holding torque, weight, and ease of use. Table 2 is a Pugh chart showing the pros and cons for five motor options we were considering. The HY100 1713 was a motor readily available to use and our first option. After we tested the HY100 motor we found that the holding torque and the moving torque for this motor was not sufficient for the constraints of our project. As shown in the Table 2, our best option was the 57BYGH420; this motor gave us a holding torque more than three times the holding torque of the HY100.

The 57B motor was also well within our projected motor budget of \$100, at only \$24 each. Another plus to the 57B was the fact that the motor driver needed for these motors, a 4988 stepper driver, has a potentiometer built-in enabling us to maximize current flow, which in turn maximizes our holding torque. The best attribute to this motor, however, is the number of steps per revolution; the HY100 has only 100 steps while the 57B motor has 200. Doubling the number of steps the motors have helped by increasing our painting resolution.

Table 2: Pugh chart comparing the different motor options.

		Motor options:				
Criteria:	Weight:	HY100 1713	SY57ST56-0606B	SY57STH56-1006A	SY57STH76-1006A	57BYGH420
Cost	4	++	0	0	-	+
Holding Torque	4	--	0	+	++	+
Weight	1	++	+	--	--	--
Easy of use	2	+	0	0	0	++
+		12	1	4	8	12
0		0	10	6	2	0
-		8	0	2	6	2
Net Score		4	1	2	2	10

## Motor Controllers – Ben Straub

In order to drive the motors, it is necessary to have motor controllers. These motor controllers must be able to handle the necessary current required by the motors without overheating.

At the beginning of the project, we used a pair of dual channel – one channel per stepper motor coil – H-bridge motor controllers based around the L298 [1]. These motor controllers are not specific for stepper motors. Also, they do not contain any active current-limiting. Active current-limiting allows you to maintain a steady average current independent of the supply voltage by pulsing the current with a PWM signal. While these motor controllers were suitable for the original motors, using them with the new motors required added difficulty. This is because the new motors ran on a lower voltage at a higher current; specifically, they required up to 2A at 3V per coil. Firstly, the higher current pushed the capabilities of our motor drivers. Secondly, due to the lack of current limiting, a 3V power source capable of supplying 8A would be needed for full power operation. Additionally, in order to adjust to a lower current level, the supply voltage would need to be lowered. While 5V and 12V supplies are relatively common, 3V and lower supplies are much more difficult to find.

To address these issues with our motor controllers, we considered switching to a pair of active current-limiting stepper motor drivers based around the A4988 [2]. These in particular were considered because they were readily available to us. These motor controllers allowed us to use any supply voltage between 8V and 35V. Additionally, an on-board potentiometer allowed us to easily adjust the current limit, a feature that proved useful in finding the current levels that produced an acceptably low level of heat in the motors and motor controllers. Another advantage of the A4988 was that it was a dedicated stepper motor driver, and was therefore easier to use for driving our stepper motors than the previous motor controller was. For these reasons, we chose to use the A4988-based motor controllers.

## Power Supply – Ben Straub

An essential but sometimes overlooked component of nearly any electronics-related project is the power supply. A poor power supply choice can cause many issues including random restarts, a lack of motor torque, or even permanent damage to other components.

The main consumers of power in PICASSAU are the motors. The stepper motors we selected can draw up to 2A per coil at 3V each. With two coils per motor, this comes out to 24W. To reduce heat in the motors as well as the motor controllers, we chose to run these at about 1.4A each, which was selected by testing various current levels and observing their levels of heat. 1.4A corresponds to a power level of about 3W per coil, or a total of 12W. It should be noted, though, that, because the motor coils are kept at the appropriate voltage and current using PWM, the power draw will not be a constant 12W, but will have peaks that go above 12W.

The next biggest consumers of power are the Raspberry Pi and its peripherals, including display, keyboard, and mouse. The RPi, keyboard, and mouse together, consume right around 0.5A on 5V, or 2.5W [3]. The display, which has not yet been purchased, is anticipated to draw an additional 2W on 12V.

Factoring in the other components, such as the microcontroller, the servo motors, and sensors, along with efficiency losses in the motor controllers and wiring, we estimate a bare minimum power requirement of 25W.

We considered the following options for our power supply:

- Initial Enercon power supply – We used this at the beginning of our project because it was on hand. This is an AC adapter, commonly referred to as a “wall wart”, with an adjustable voltage, and a maximum power output of 30W at 12V. However, in testing, it proved to be somewhat flaky at high power consumption, and would occasionally reset.
- Standard computer power supply – This is somewhat bulky but can be obtained cheaply and typically can supply plenty of power. Additionally, it has the convenience of having both 12V and 5V already, eliminating any need for regulating a 5V line from the 12V line.
- 12V AC adapter – This is similar to our initial power supply, except that it can supply 6A, or 72W. Additionally, instead of being an outlet-attached “wall wart”, it is more similar in appearance to a laptop power cable.

These options were analyzed using the Pugh chart in Table 3.

Table 3: Pugh chart detailing the advantages and disadvantages of the various power supply options.

		Power Supply Options:		
Criteria:	Weight:	Initial AC Adapter	Computer Power Supply	New AC Adapter
Cost	4	++	+	+
Power capability	3	--	++	+
Need for 5V regulator	1	-	+	-
Size	1	++	--	+
+		10	11	8
0		0	0	0
-		7	2	1
Net Score:		3	9	7

We therefore selected the computer power supply option.

## Graphics Type – Drew Kerr

For our project, we needed to convert an image uploaded to the computer or a webcam image into vectors that the computer parses and converts into sets of coordinates and instructions. Essentially, this would be converting other image types such as jpeg or jpg into a vector graphics file.

The industry standard for a vector graphics file is Scalable Vector Graphics (SVG), which is an XML-based vector image format for two-dimensional graphics [4]. The main functionality of SVG files falls within its path feature. This file feature stores movements and coordinates for drawing lines, Bézier curves and elliptical curves, which are the backbone for constructing any image using this format. In addition, these paths store information about the colors of the lines or curves defined in the path. Furthermore, the paths contain information about the height and length of these curves and lines that we scale to fit our canvas. Using these attributes, we are able to convert these paths into instructions that the Arduino uses to control the paintbrush carriage that paints the picture.

The SVG file type offers many advantages over other image formats for our project. For example, users have the ability to search and index SVG files. This is particularly important because it allows us to parse SVG files into coordinates and instructions that the Arduino can interpret as painting instructions. In addition, SVG images are scalable and printable with high quality at any resolution. This feature permits us to be able to receive any image size and scale that image to fit onto our canvas without losing resolution. Along those same lines, we can scale any image to fit any canvas size. Furthermore, SVG is an open standard and thus has many resources available to maximize our understanding and usability of the format.

## Paintbrush Carriage Kinematics – Ben Straub

### *General kinematics and motion control*

In order to have a painting robot, one of the key challenges is being able to accurately move the paintbrush, along with the carriage on which it rests, to the desired locations on the canvas. The control of the movement of the carriage was made particularly challenging due to the physical structure of the robot. For reasons discussed on page 4, the carriage is suspended from two motorized spools mounted at the top of the frame. To move the carriage, the spools are rotated to increase or decrease the length of the fishing line between the spool and the carriage. When moving upwards, the line is wrapped around the spools to pull the carriage up. When moving downward, the line is unspooled, and gravity pulls the carriage down.

The kinematics of such a system becomes complex because moving in a straight line is not a simple linear process. Consider, for example, a starting point A and a destination point B. Each of these points can be defined by two quantities: the distance from the left spool and the distance from the right spool. Let A and B have an equal distance from the left spool and different distances from the right spool. An initial attempt at a control scheme might be to keep the left spool still and simply move the right spool to get from point A to point B. While this would get the carriage to the correct location, it would travel in an arc with a constant radius from the left spool. In order to move in a straight line, a more complicated control scheme is required.

In order to help simplify the kinematics and the implementation of the controls, we normalize all distance and coordinates in terms of steps of the stepper motors. Thus moving a stepper motor one single step causes distance between the carriage and the corresponding spool to change by 1 unit. As another simplification, instead of storing all coordinates as their distances from the two spools, we store

them as rectangular x and y coordinates. Lastly, as a final simplification, all of the coordinate are relative to the top left spindle such that (0,0) is at the top left spindle, moving to the right increases the x coordinate, and moving down increases the y coordinate.

Our control scheme works by checking multiple possible moves and deciding which move is best. This is broken up into several steps. The first portion checks each motor and determines if moving it in the positive direction or negative direction moves the carriage closer to its destination. It also checks to see if moving both motors simultaneously gets it closer. Any moves that get the carriage closer to the destination is considered a valid move. If there are no valid moves, then the carriage has arrived as close to the destination as it can get and the movement is done.

If there are valid moves, then the system next checks how far from the line of travel – that is, the straight line from the starting point to the destination point – that each valid move will put the carriage. Whichever valid move gets the carriage closest to the line is the chosen move.

This process is iterated until there are no valid moves.

Several equations are required to make this system possible. This includes calculating the distance from a point to another point, calculating the position of the carriage based on the two distances from the spools, and calculating the distance from a point to a line defined by two points [5]. For a complete list of the equations used in this movement control system and where they come from, see the Appendix on page 35.

### *Calibration*

In order for the control scheme to work properly, it is essential to know the initial position of the carriage. In order to accurately achieve a known initial position, we developed a calibration routine. This routine starts the carriage in the lower half of its range. It then slowly brings the carriage up until it gets detected by a distance sensor that is mounted to the frame and facing horizontally across the carriage's range. The y value of the carriage's coordinates is then known to be the y value of distance sensor's coordinates. The x value is known to be the distance seen by the sensor added to the sensor's x value. Because distance sensors often have certain ranges at which the precision and accuracy are higher than others, this calibration can be further fine-tuned by then moving the carriage horizontally to this range. A simple diagram showing these movements can be seen in Fig. 1. The blue boxes represent the different positions of the carriage at different times. The red arrow represents the viewing beam of the distance sensor, which in this case is explicitly stated to be an IR distance sensor.

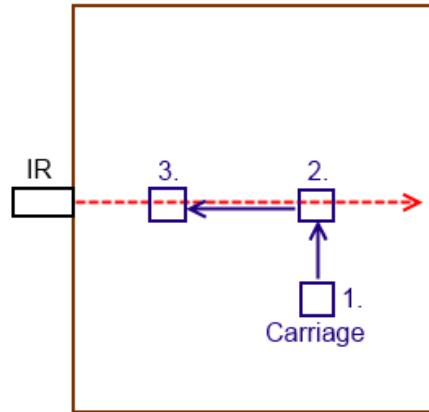


Figure 1: Diagram demonstrating the movement of the carriage during the calibration routine.

This calibration process relies on having a sensor to sense distance. We considered two options: an IR rangefinder and an ultrasonic rangefinder. Specifically, we looked at the Sharp GP2Y0A21YK IR rangefinder and the Parallax PING))) ultrasonic rangefinder, because of both of these sensors were immediately available and on-hand. Both sensors were fairly easy to use. The PING))) sensor was

slightly easier, as libraries were readily available for it, and it required little math to determine the distance. The IR sensor was only slightly more difficult, requiring an equation with floating point exponents. However, we selected the Sharp IR sensor because, unlike the PING)), it has a very narrow beam of detection, which was a desirable trait for our calibration sensor.

### *Curves*

In vector-based graphics, such as in SVG files (see page 9), it is often necessary to be able to define curves. The common way of doing this is by means of a cubic Bezier curve [6]. These are defined by four points: a start point, an end point, and two control points. In order to draw these, we considered two options. Firstly, we considered letting the Arduino's control system handle it, and, secondly, allowing the computer to handle it by breaking up the curve into straight line segments.

The only additional calculation required on the Arduino to handle curves would be a calculation of distance from a point to the curve. However, this problem yielded a quintic equation, which can be very difficult to solve programmatically. On the other hand, Bezier curves are particularly easy to split into linear segments because they are parametric equations, so simply evaluating it at evenly spaced intervals spanning the range of the parameter  $t$  yields evenly space points along the span of the curve. Thus, the second option would be much easier to implement than the first. The only tradeoff to the second option would be a loss of accuracy. Breaking a curve into linear segments can yield a poor approximation if it is not broken into enough segments. To counteract this, we simply picked a conservative number of linear segments to break the curves into and implemented this option.

## Software and Computing – Kayla Frost

In the Proposal Phase, we determined that we needed software capable of doing the following:

- Perform image processing
- Interface with a USB webcam
- Present a GUI to interact with images
- Parse vector image files for paths
- Convert paths to a series of motions for the brush
- Calibrate the stepper motors
- Perform calculations to determine how to travel from point to point
- Control stepper motors to move the brush carriage
- Control servo motors to create paint strokes
- Dip the brush at the appropriate interval into the correct container

### *Platform*

One option we investigated was running all of these functions on a microcontroller. We found that microcontrollers would not have the processing power necessary for the image processing we required. Also, implementing complex tasks such as a USB host interface or generating a GUI on a microcontroller would have been prohibitively difficult.

We considered exclusively using an inexpensive, small single board computer, such as the Raspberry Pi. Utilizing the GPIO of the Raspberry Pi, or RPi, to control motor functions would have been much more difficult compared to using a microcontroller. The RPi also does not have all of the ports necessary, such as an ADC [7].

We decided to combine these solutions and split the software tasks between the two platforms:

1. A small computer to handle computationally complex tasks and interface with users
2. A microcontroller to manage the embedded software responsibilities and interface with hardware

We used this configuration to take advantage of the benefits of each separate device without having to force a function onto a platform it does not belong. This solution divided the software into two isolated sections, which made debugging easier. Having both of the necessary components available at the beginning of the Proposal Phase also made this method a logical choice.

### *Computer Software*

The main decision to be made on the Computer Software side was to select which language to use to implement all of the functions. We discussed three different options, which are described in Table 4, and decided to use Python.

Table 4: Pugh chart for determining which high-level programming language to use on the RPi.

Criteria:	Weight:	Programming Language Options:		
		C/C++	Java	Python
Team's amount of past experience	5	++	0	+
Image processing utilities (OpenCV)	4	+	-	+
Efficiency	1	+	0	-
Ease of use	4	0	-	++
Readability	3	0	0	++
+		15	0	23
0		7	9	0
-		0	8	1
Net Score		15	-8	22

We produced a Python script which runs all of the computer software functions. At the end of Cycle 1, it runs on a laptop. The script accepts an SVG file and extracts the paths from the XML [4]. It sorts the

paths by color and reorganizes the paths into a series of either position movements or brush strokes for the paintbrush. Then the script sends these to the microcontroller to implement in hardware.

In Cycle 2, we will begin using the RPi to run the script. We also plan to interface with the webcam, vectorize images internally, and implement a GUI.

### *Embedded Software*

The main decision for embedded software was to decide what kind of microcontroller to use. We identified the Really Bare Bones Board (RBBB), an Arduino clone, as an option [8]. For simplicity, when we refer to the RBBB, we will call it the Arduino in this report. The other microcontrollers/boards we considered are: the DragonFly module used in ELEC 3040/3050, PIC microcontrollers, and an ARM-based microcontroller. When looking at these other microcontroller options, we found that they were very similar relative to our needs. Because of this, we evaluated them as a group against the Arduino, as shown in Table 5.

Table 5: Pugh chart for determining which microcontroller to use to control hardware.

Criteria:	Weight:	Microcontroller Options:	
		Arduino	All others
Team's amount of past experience	5	+	0
Cost	3	0	0
Efficiency	1	0	+
Ease of use	4	++	-
+		13	1
0		4	8
-		0	4
Net Score		13	-3

Note that the cost for the two boards is approximately the same. The Arduino is more expensive than the other microcontrollers, but the others would require buying a specific programmer. In contrast, the

can be programmed using an FTDI cable that connects directly to a computer via USB. We selected to use the Arduino because of its ease of use.

First, the Arduino is responsible for calibrating the motors using the IR sensor mounted on the frame.

Then, it accepts the instructions from the Python script and calculates how to move the various motors to accomplish the given task. Finally, it interfaces with the motors to carry out the instructions.

In Cycle 2, we plan to add support for multiple colors. We will also work on stabilization and optimization of the current methods. Additionally, we will interface with the servo on the brush carriage to implement brush rotation. This will result in more brush control and better output paintings.

## Communication – Kayla Frost

After deciding to split the software onto two different platforms, we needed a way to communicate between them. We looked at different options available to link the RPi and the Arduino. We decided using the Pugh chart in Table 6.

Table 6: Pugh chart for determining what kind of communication link to use between the RPi and Arduino.

Criteria:	Weight:	Microcontroller Options:		
		Bluetooth	GPIO on RPi	Serial
Team's amount of past experience	5	-	-	++
Cost	4	-	+	0
Efficiency	1	+	-	-
Ease of use	3	-	+	0
+		1	7	10
0		0	0	7
-		12	6	1
<b>Net Score</b>		<b>-11</b>	<b>1</b>	<b>9</b>

We selected to use the serial connection because it was much easier to implement. The serial interface we selected uses the serial communication protocol defined by the RS-232 standard, but at TTL voltage levels instead of the original RS-232 levels [9]. This TTL RS-232 interface is easily used by the Arduino's built-in UART. An FTDI USB-to-serial cable is used to convert the computer's USB interface to the Arduino's TTL RS-232 interface.

### *Communication Protocol*

We developed a communication protocol to send the commands one at a time from the Python script to the Arduino over the serial connection. This is detailed in Fig. 2 on the next page.

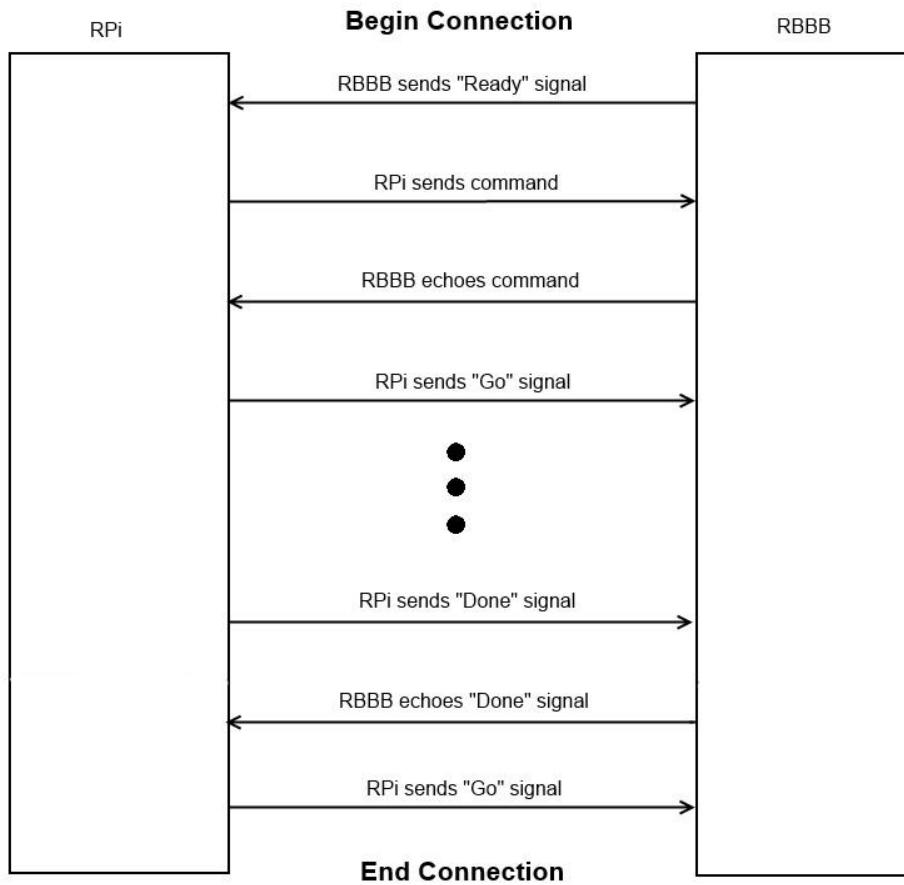


Figure 2: This figure outlines the communication process for sending a command from the RPi to the Arduino.

### *Error Checking*

For our communication link, we decided to use a serial connection, which can be unreliable. To overcome this, we implemented some error checking into our communication protocol in order to avoid data loss. Referring to Fig. 2, the RPi will not issue the “Go” command until the Arduino echoes back the exact statement the RPi sent. If the RPi does not receive this, it will reissue the command to try again. This means that if any part of the packet was corrupted to or from the Arduino, no action will take place. One disadvantage to this method is that it is less efficient and requires several more instruction cycles on both sides of the connection. It also took significant time to develop and debug this protocol. However, the reliable output that resulted is worth the sacrifice of efficiency and time.

## Image Processing – Drew Kerr and Peter Gartland

In order to give our paintings artistic flare, we decided to send the uploaded photo or webcam image through a filter that modifies the image to only have four colors. The team's Auburn background influenced the color selection to be blue, light blue, orange, and the white canvas background.

The filter that we use is based on the OpenCV library that contains many built-in filtering functions. The filtering process begins by converting the colored image shown in Fig. 3 to the grayscale image shown in Fig. 4.



Figure 3: The original image.



Figure 4: The grayscale image.

After we convert the image to grayscale, we smooth the image with a median filter that replaces each pixel with the median of its neighboring pixels [1]. We filter the image in this manner to reduce pixel noise and sharpen edges where there is a significant color contrast. One can see the smoothed image below in Fig. 5.



Figure 5: The smoothed image.

Next, we reduce the number of colors in our grayscale image to four. Each pixel in the grayscale, smoothed image has a value of 0 to 255 associated with it, depending on how light or dark the pixel is. For example, the circled point on the left side of Fig. 5 corresponds to a grayscale value of 25, while circle closer to the center of the image corresponds to a grayscale value of 240. Using each pixel's grayscale value and the user-defined color thresholds shown in Fig. 6, we assign every pixel one of four values: 0, 85, 170, and 255. One can adjust these thresholds to match the light settings of the original image or to create more contrast within the image. These values correspond to different shades of gray and convert the image into a new image composed of the four shades of gray. One can see the new image below in Fig. 7.

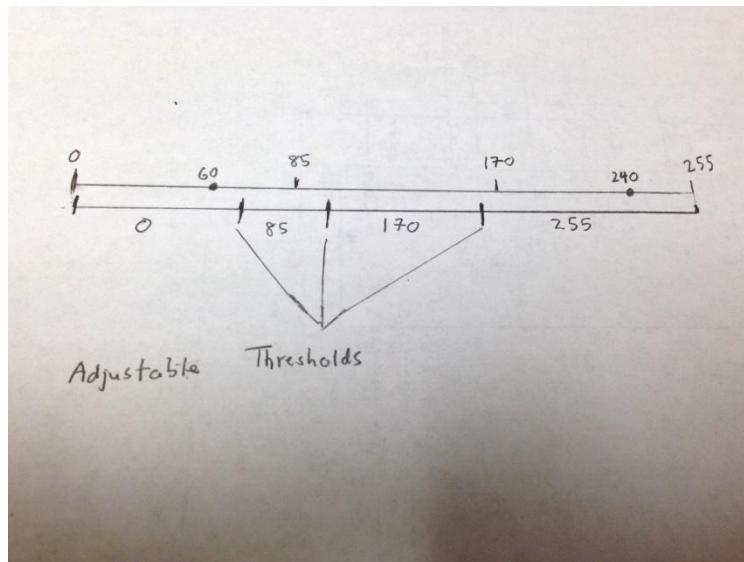


Figure 6: Threshold system for assigning colors.



Figure 7: The thresholded image.

After we assign the pixels to one of the four values mentioned above, we delegate each of the four values to a color: blue, light blue, orange and the canvas background. One can see this final image in Fig. 8.



Figure 8: The final image that the robot will paint.

## Constraints – Peter Gartland

In the first development cycle, we identified several constraints PICASSAU must meet or work around:

Cost:

- The total cost of PICASSAU should not exceed our budget of \$300. This was the primary consideration for much of PICASSAU's performance. The cost affected our choices of motors, processor, power supply, and building materials.

Performance:

- Currently, PICASSAU runs off of a power supply plugged into a wall outlet, which limits PICASSAU's area of operation to the length of extension cords.
- PICASSAU may not be able to provide a real-time image adjustment display without a faster (and therefore more expensive) processor.

Marketability:

- Because much of PICASSAU's value is in the novelty of watching a robot paint a painting, its consumption may be limited to those intending to display it in a public setting to attract business.

Sustainability:

- PICASSAU's many exposed moving parts have made it a relatively fragile product and may require expert maintenance to keep operational, especially in its current developmental state. A simple interface and clear documentation will help keep PICASSAU operational for E-Days to come.

## Budget – David Toledo

As shown in Table 7, our projected budget of \$265 was below our goal of no more than \$300. At the end of cycle one our actual budget is shaping up to be less than our projected budget, although this is partly due to the fact that we have not yet purchased a display for the Raspberry Pi. A couple of main points to bring up are that the selection of motors purchased for PICASSAU was 43% cheaper than projected, saving \$43 for possible later use; another point to notice was the fact that the projected budget for the framing and mounting hardware ended up being more expensive by \$27 dollars. With the interfacing display being the last expected necessary purchase, our budget is leaving us with a cushion of approximately \$54 for unexpected parts replacement for the remainder of the project.

Table 7: Budget, including estimated and actual costs.

Item	Purpose	Estimated Cost	Actual Cost
Stepper Motors	to replace the underpowered motors used in the prototype	\$100	\$57
Frame Hardware	to support the hardware	\$20	\$31
Mounting Hardware	to mount the spindles onto the motors	\$15	\$31
Painting Supplies	brushes, paint, and canvas/poster-board	\$15	\$13
Motor Controllers	to drive the stepper motors	\$20	\$20
Raspberry Pi	to run the software	\$35	\$40
Display	to interface with the raspberry pi	\$15	Not yet Purchased
Webcam	to take pictures to paint	\$15	\$4
Microcontroller	to interface with the hardware	\$10	\$20
Miscellaneous Electronics Hardware	wires, sensors, servos, etc.	\$20	\$16
<b>TOTAL</b>		<b>\$265</b>	<b>\$231</b>

## Timeline – Kayla Frost

We created a Gantt chart at the beginning of the Cycle 1. We scheduled the project objectives and grouped them by subsystem to guide our progress each week. According to the chart, we showed major improvements in each subsystem and successfully completed all Cycle 1 goals.

Additionally, we were already able to begin the following Cycle 2 tasks:

- Stabilize paintbrush platform
  - An objective that incorporates three different subsystems
- Setup Raspberry Pi
  - A major dependency in Cycle 2

Heading into Cycle 2, PICASSAU is slightly ahead of the original schedule. See the full Gantt chart in Fig. 9 for more detailed timeline information.

### Gantt Chart

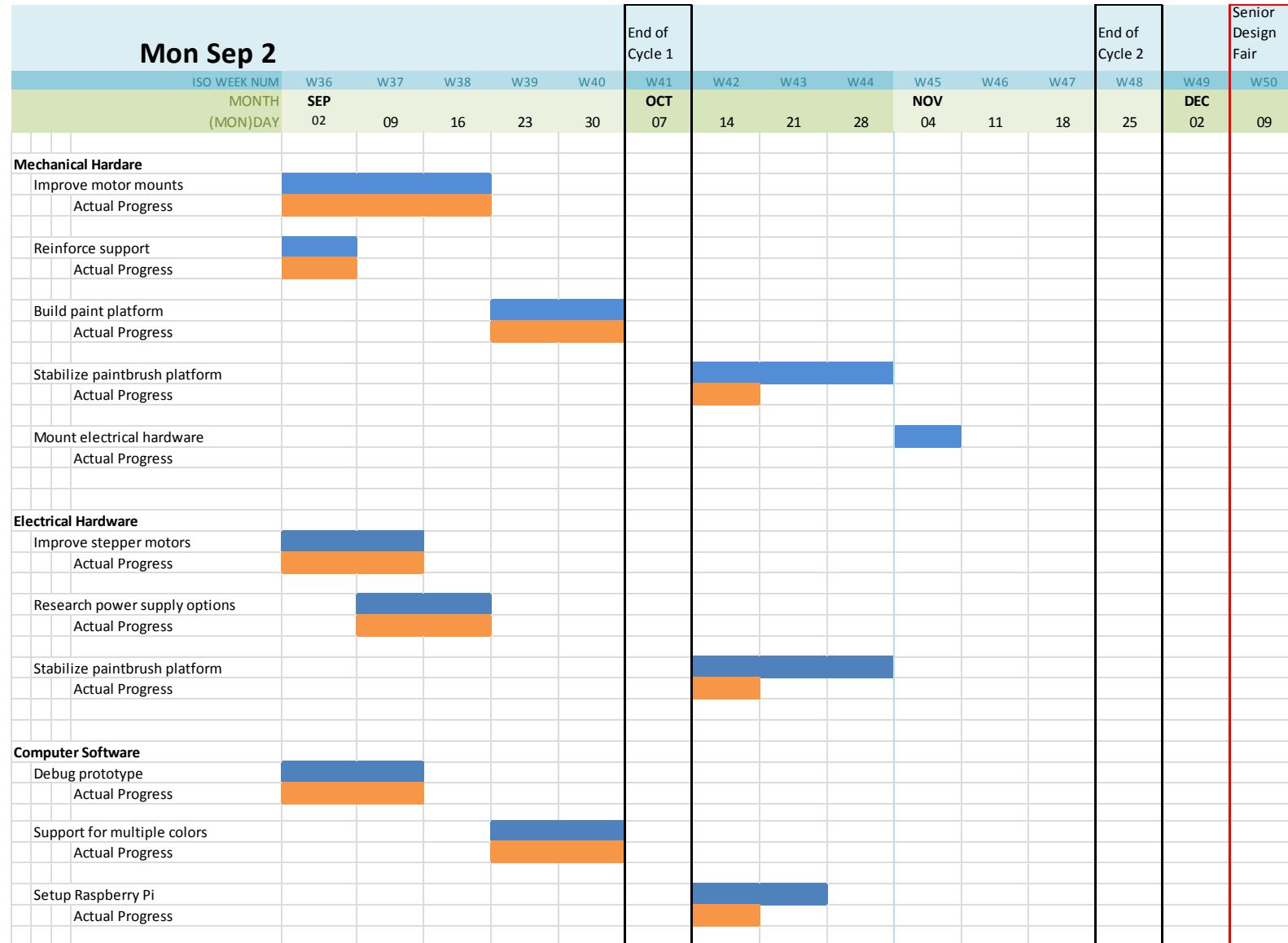


Figure 9: This Gantt chart outlines the timeline for the project.

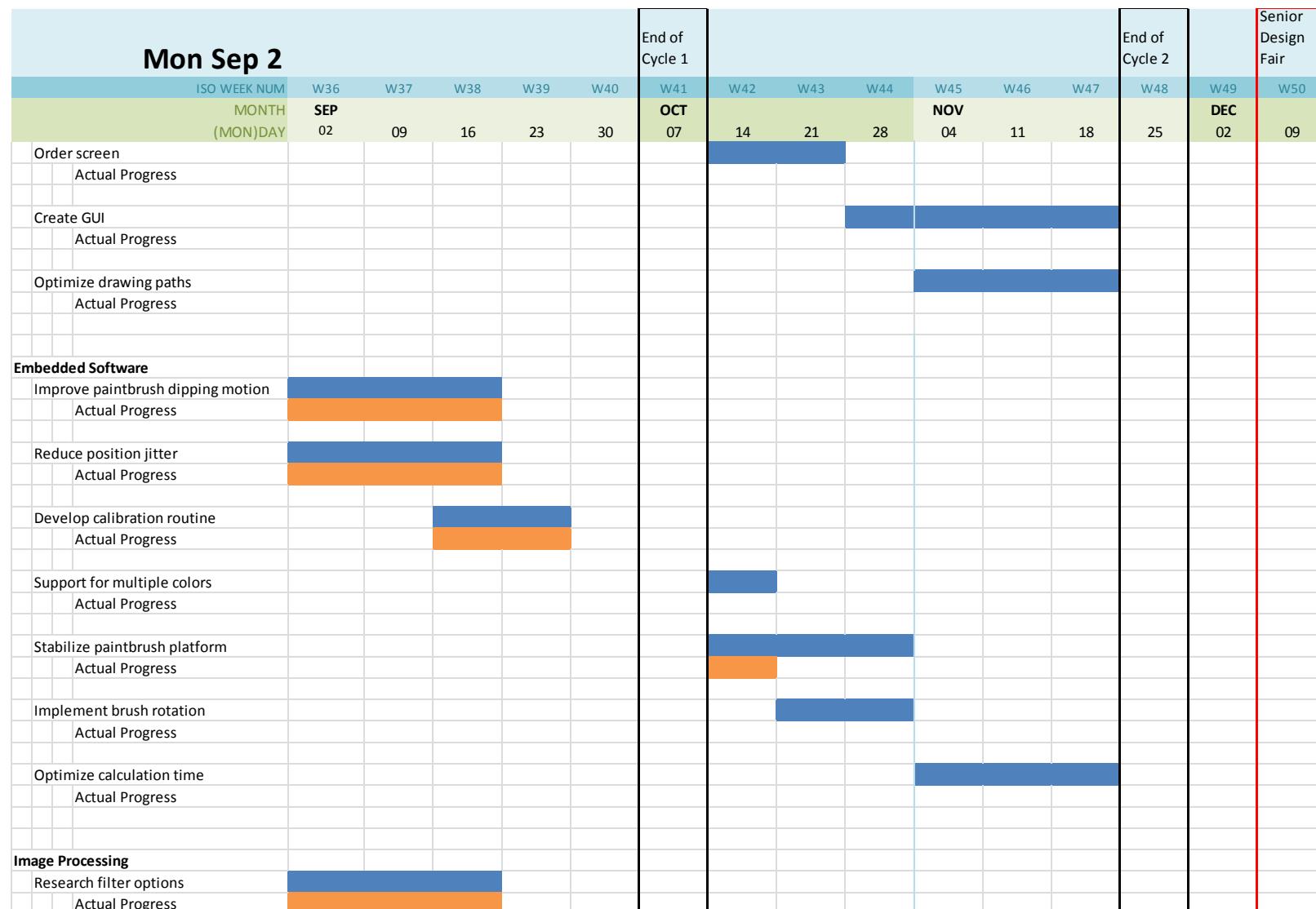


Figure 9 continued.

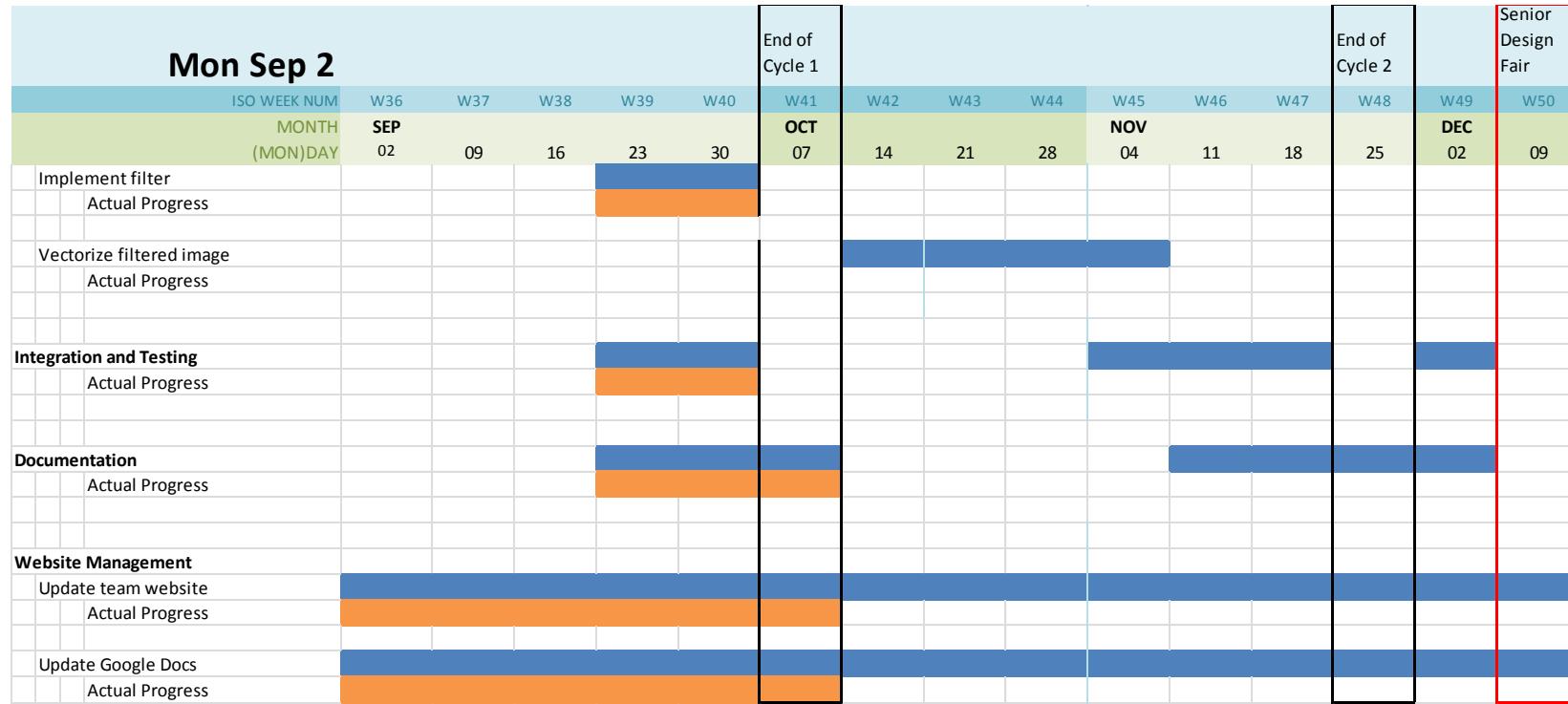


Figure 9 continued.

## Performance – Ben Straub

PICASSAU's performance at the end of Cycle 1 meets and surpasses most of our expectations. Our Cycle 1 goals were to have a sturdy robot that could take a supplied SVG file, use that to send coordinates of lines to be painted to the microcontroller, and paint along the commanded lines, dipping the paintbrush as necessary. Additionally, we wanted to have progress on our ability to filter an image down to a simple 4-color image that could then be vectorized.

In terms of the physical structure, the robot meets nearly all expectations. The only shortcoming is that a few of the mounts were made in a temporary manner, such as hot glue. This was done because at the time, we were simply testing different options. Now that the physical set up of the robot is decided, a better mounting method, such as screws or bolts, should be used instead. This will be addressed early in Cycle 2.

The actual performance of the computer software portion of the robot meets the projected performance for Cycle 1. It properly reads provided SVG files and transmits coordinates appropriately to the microcontroller. The only slight issue is that certain features in some SVG files cause problems in our software. Specifically, certain files cause the picture to be shifted from its intended location. While this may cause issues in our Cycle 1 performance, it should not matter for our Cycle 2 goals. This is because in Cycle 2, we will be creating our own SVG files from our filtered images, so we can be sure not to use the problematic features.

Lastly, our plotting and painting performance almost entirely meets our expected performance for Cycle 1.

1. One issue here is that during calibration, the distance sensor used for calibration sometimes erroneously reports sensing the carriage. We believe that this might be caused by the sensor

sometimes picking up the fishing line used to suspend the carriage. We plan to address this in Cycle 2 by repositioning the sensor such that the carriage will still be in its view but the fishing line will not.

The other issue with our painting performance is that after enough painting, the brush becomes saturated with paint and drips significantly, sometimes running down the canvas. To address this, we will test different thicknesses of paint. We will try shaking the brush out more, and potentially scraping off excess paint from the brush onto the lip of the paint reservoir.

## Conclusions – Kayla Frost

We now have a robotics platform that can accurately paint a given single-color SVG file. It performs all of the basic functionality we planned. Cycle 2 will focus on enhancing and improving this design.

Throughout this project so far, we have learned valuable skills that are relevant to the field of Electrical Engineering but that are not covered in our curriculum. Some examples of these include:

- *Mechanical Design*
  - Creating a stable frame, mounting motors, and stabilizing the carriage using weights were challenging problems to solve.
- *Motors and Motor Control*
  - We had used DC motors in the past, but this project exposed us to stepper motors. We learned that stepper motors are more useful in applications that require accuracy, like PICASSAU.
- *Communication Protocol*
  - We learned how to connect two separate devices to reliably communicate information in two directions.
- *Scripting Languages*
  - Programming without any hardware interaction taught us how to utilize the compiler tools to debug problems.
- *Image Processing*
  - We learned how to use several different filtering methods.

Our primary goals for Cycle 2 are to implement multiple color support, generate the vectors internally, and add a GUI to improve user experience. Our secondary goals are to optimize and stabilize the current system such as mounting loose hardware, minimize timing, and adjusting the brush carriage. We also plan to improve the aesthetics of the robot. By the end of this project, we anticipate a sophisticated and reliable final solution that can accept a picture and output a complex painting of the subject.

## References

- [1] "L298 2A 6V-50V Compact Dual Motor Driver Kit." *RobotShop*. [Online]. Available: <http://www.robotshop.com/solarbotics-l298-motor-driver-kit-3.html>
- [2] "A4988 Stepper Motor Driver Carrier." *Pololu*. [Online]. Available: <http://www.pololu.com/catalog/product/1182>
- [3] Forcie. "RasPi power usage measurements ~2 watts at idle." *Raspberry Pi*. [Online]. Available: <http://www.raspberrypi.org/phpBB3/viewtopic.php?f=63&t=6050&start=50>
- [4] "Introduction to SVG." *w3school*. [Online]. Available: [http://www.w3schools.com/svg/svg\\_intro.asp](http://www.w3schools.com/svg/svg_intro.asp)
- [5] E. Weisstein. "Point-Line Distance--2-Dimensional." *Mathworld – A Wolfram Alpha Web Resource*. [Online]. Available: <http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html>
- [6] "Bezier Curves." *T-Splines*. [Online]. Available: [http://www.tsplines.com/resources/class\\_notes/Bezier\\_curves.pdf](http://www.tsplines.com/resources/class_notes/Bezier_curves.pdf)
- [7] "RPi Low-level peripherals." *Elinux*. [Online]. Available: [http://elinux.org/Rpi\\_Low-level\\_peripherals#General\\_Purpose\\_Input.2FOutput\\_.28GPIO.29](http://elinux.org/Rpi_Low-level_peripherals#General_Purpose_Input.2FOutput_.28GPIO.29)
- [8] "Really Bare Bones Board (Arduino) Revision B & B2 Instructions." *Modern Device*. [Online]. Available: [http://cdn.shopify.com/s/files/1/0038/9582/files/RBBB\\_Instructions\\_06.pdf?1260749296](http://cdn.shopify.com/s/files/1/0038/9582/files/RBBB_Instructions_06.pdf?1260749296)
- [9] Electronic Industries Association. Engineering Dept. *Interface between data terminal equipment and data communication equipment employing serial binary data interchange EIA standard*. Washington: Electronic Industries Association. Engineering Dept, 1969.

## Appendix

These equations are used to determine how to move the carriage in an approximately straight line.

### 1) Calculating the distance d between two points P<sub>1</sub> and P<sub>2</sub>.

This equation is simple, but very useful in our calculations. It is

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

### 2) Calculating the coordinates of a point P=(x,y) that is a distance d<sub>1</sub> from a point P<sub>1</sub>=(x<sub>1</sub>,y<sub>1</sub>) and d<sub>2</sub> from point P<sub>2</sub>=(x<sub>2</sub>,y<sub>2</sub>).

The general equations for this are

$$d_1 = \sqrt{(x_1 - x)^2 + (y_1 - y)^2}$$

$$d_2 = \sqrt{(x_2 - x)^2 + (y_2 - y)^2}.$$

However, these equations can be difficult to solve. It greatly simplifies the problem to let our point P<sub>1</sub>, which is the location of the left spindle, to be defined as the origin, (0,0). Because the point P<sub>2</sub> is the right spindle, its location is simply (L,0), where L is the distance between the spindles. This reduces the above equations to

$$d_1 = \sqrt{x^2 + y^2}$$

$$d_2 = \sqrt{(L - x)^2 + y^2}.$$

These can then be easily solved for x and y:

$$x = \frac{d_1^2 - d_2^2 + L^2}{2 * L}$$

$$y = \sqrt{d_1^2 - x^2}$$

**3) Calculating the distance  $d$  from a point  $P_0=(x_0,y_0)$  to a line defined by points  $P_1=(x_1,y_1)$  and  $P_2=(x_2,y_2)$ .**

This equation is key for picking the optimal movement choice in our control scheme. As shown in [5], the equation is

$$d = \frac{|(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$