Painting Robot

Chen, Chien-Ning - 804514266

Khandelwal, Shubham - 904516496

Lee, SungWook - 804412480

Nguyen, Anthony - 703877763

Table of Contents

1. Abstract

2. Introduction

3. Motivation

4. Background/Existing Work

5. System Setup

6. Robotic Arm Setup

7. Robotic Arm Analysis

8. System Design

9. Motion Generation

10. Evaluation

11. Observations and Limitations

12. Results

13. Future Directions

14. Conclusion

15. Demos

16. References

1. Abstract

Existing painting robots tend to resemble plotters in a way that their prime concern is about the color pixel value at a particular location. Due to this principle, paintings created by these robots tend to be more dry and inorganic as they lose subtle variations that can be added to the painting because of various features with which a brush tip can be moved relative to the canvas. In this project we work to overcome this limitation by considering these features in our system and accounting for them mathematically in order to generate robotic motions which accounts for these factors. Based on this concept, we create a set of commonly used painting strokes that considers with these rich features. We have then tried to modularize approach to painting by defining a painting as a set of these strokes and thus be able to create it by the robot as a combination of such strokes. Our results show that it is possible to obtain a wide range of dynamic and organic paintings by robots with accuracy depending on the quality and resolution of motors used and the structure of the robotic arm. Results also show that it is possible to modularize paintings as a set of variety of strokes and preserver the organic nature of the paintings.

2. Introduction

Art is an expression bestowed by God on humans to create masterpieces which intrigue and also motivate and inspire generations with new ideas. The talent of an artist is in his skill to be able to emote efficiently and effectively the idea in his mind to large audience. For a painter this is emoted through his paintings, which through its dynamisms expresses a variety of feelings. The thoughts of a painter are transferred from his mind to his hand which is then translated to the paint brush and results in creation of art which spell bounds the audience.

Clearly, not everyone is so talented or gifted to be able to execute all his ideas into actions using this form of art. But everyone does have thoughts to express. Only if it was possible to bridge this gap between thoughts and talent to execute, can the true potential of humanity be explored and exploited. Geniuses in all fields must be able to express their ideas in ways other than words. For example a scientist or a mathematician may find it hard to connote his feelings through such artistic forms, unless they are Da Vinci. It is with this idea that we aim to try to create a robot which can help execute this gap in flow from idea to art. If we are able to characterize some key elements of a painting into some well defined modules or blocks, a combination of various such modules can be used to recreate a painting that originates from some idea.

Thus our aim is to come up with a set of features, using which some commonly used painting strokes or techniques can be recreated by a robot, a combination of which can be used by an individual who is less proficient in successful execution of his feeling to a painting. Based on the feelings and thoughts, an individual can use these modules and create a piece of art which is beautiful and also reduces his effort.

A common observation among available robotic painters is that they tend to be more technical in terms of their task, so they are more concerned about the color value at a given location. This takes away the ‘art’ factor of the painting. A digital representation of painting may lose the original intended idea behind the creation of the painting, or the feeling that painting was intended to

express. Thus we plan to use a set of features that the various strokes or modules of the painting must have in order to preserve this 'organic' factor of the painting.

In this report we describe the motivation for this project. We then do a overview of available existing work in literature related to this system and give a brief background of our project. This is followed by describing the overall layout of the system. The robotic arm structure that we have used for this project has been described next. We then perform the analysis of this robotic arm for being able to come up with the efficient robotic model. The forward and inverse kinematics are described in detail in the next section along with feedback analysis of the arm. Next section describes design of our system, software and hardware components involved and its integration based on MATLAB and arduino packages. We then explain our methodology for generating various motions considering the different feature sets of the strokes. We have listed the method of our system evaluation. This is followed by our observations and results. We then give an idea about the applications of this and some future directions followed by the conclusion.

3. Motivation

Painting is more than just a combination of colors at a position. A digital representation is a description of (r,g,b) value of every pixel. Recreating image from this or creating image based on this concept causes an artificial painting or 'inorganic' painting. Capturing of the expression of the artist and the feeling of the painting is the single most important key while recreating a piece of art. The color dynamics and 'liveliness' of a painting is because of the variety of expressions that can be generated because of the way a brush is held and also on the way a stroke is created. Thus paint brushes can generate a larger variety of dynamic outputs by varying features like the angle, pressure on the tip and speed of the brush relative to the canvas etc.

It will be beneficial to create a modular system by which a painting can be broken into small modules or strokes. Thus a combination of such different available strokes can increase the organic factor of a painting. Thus our aim is to try and create a robot which can learn execute such different strokes. If a series of these organic strokes to be executed is given to the robot, it must be able to create a painting with rich variety. In this project we thus try to design a robot which can execute some key strokes in painting without losing its 'art' factor.

Thus we plan to create a robotic structure which can be used for creating more than a geometric/artificial painting. We can achieve this by coming up with a set of parameters or features, some mathematical expressions using which different strokes in painting can be defined. We can then create a program which can execute these different strokes created as a function of such features described above. This will be helpful in modularizing approach to painting by defining a painting as a set of such patterns and thus be able to create it by the robot as a combination of such patterns. In this project, for simplicity we have assumed the base frame to be fixed. This is because scope of this project is exclusively on the ways of usage of the brush. Thus we have explored various techniques of using brushes.

4. Background/Existing Work

Existing work for our system would consist two kinds of work - existing robotic painting or drawing machines, or existing work characterizing paintbrush behavior. In our search we found limited examples of either kind of existing work

4.1 Robotic Painting

N-Degree of freedom robotic arms are used extensively in many manufacturing industries such as automobile manufacturing. However, the end effector of these arms is never a brush, but instead spray paint. The arms are moved into a certain position in space, and when the arm reaches the desired locations they will then begin to spray paint.

On the Instructables website two projects that were most relevant to our work were two robot arms that would take as an input a picture, and produce as an output a drawn version of that picture. The first project was an arm that had 2 DOF and drew using a pen. It was controlled by MATLAB and Arduino [1]. Because this project only had 2 DOF, the work was relevant to show what was possible to accomplish, but none of the work was actually applicable to our robot which has more than 2 DOF. The second project was a 4 DOF arm that used Intel Galileo and a camera to take a picture before painting it [2]. This second project is much closer to what the desired goals of our project are, but it also differs in the DOF and it wants to create from a picture, but the goal of our project is to make a painting dynamically based off of strokes.

4.2 Paint Characterization

The literature survey did not reveal any existing papers that were relevant to our project, which required characterizing strokes. What the survey did find was that there was a large body of historical work about different styles of painting and influential painters. Thus, the sources used consisted typically of online articles and direct assistance from staff at the Blick Art Supply store.

4.2.1 Canvas Size

The painting canvas comes in many different shapes and sizes. Rectangular and Square are the most common shapes, and there are several standard sizes of canvas that can broadly be categorized into mini, medium, and large [3]. In our project, 9x12 canvas size was used.

Mini	Small	Medium	Large
2x3	4x6	8x10	18x24
3x3	5x7	9x12	20x24
		11x14	12x12

4.2.2 Types of Paint

The main types of paint are acrylic and oil.

Acrylic	Oil
water based/soluble; easy to clean and work	resists water

dries quickly	require oils to thin or thicken the paint body
sticks to most surfaces	requires more time to set up
possible to paint layers atop each other immediately	more expensive
less runny than watercolors and can be used for serious painting	may take days to dry

Table 1: A comparison between acrylic and oil colors.

In the end, we chose acrylic because it made more sense to focus our project on the robot motion, and not emphasize other aspects of art such as mixing the paints

4.2.3 Paintbrushes

Making the same motions while using different types of paintbrushes will give different strokes on the painting canvas. Therefore, it is important for artists to know all the types of brushes and make the choice appropriate for them. There are 8 main types of paintbrushes, all with different functions [5].

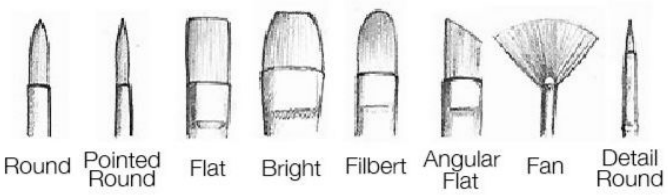

Figure 1. Eight Types of Paint Brushes

Because the most basic type of paintbrush is the round brush, this is the paintbrush that was used in this project. The key feature of this brush is that produce the same stroke regardless of the angle the brush is held. The second type of paintbrush that was used in actual paintings was the rectangular shaped flat brush. This brush can create two strokes that are fundamentally different. One stroke will result if the brush motion is in the plane of the length of the brush, and the other from the width of the brush. Because the length is longer than the width, the lengthwise stroke will make more contact with the canvas, resulting in a larger stroke where the paint is thinner and more spread out across the stroke. The widthwise stroke will make less contact with the canvas, but the contact it does make will have the same amount of paint, so the stroke will be more thick.

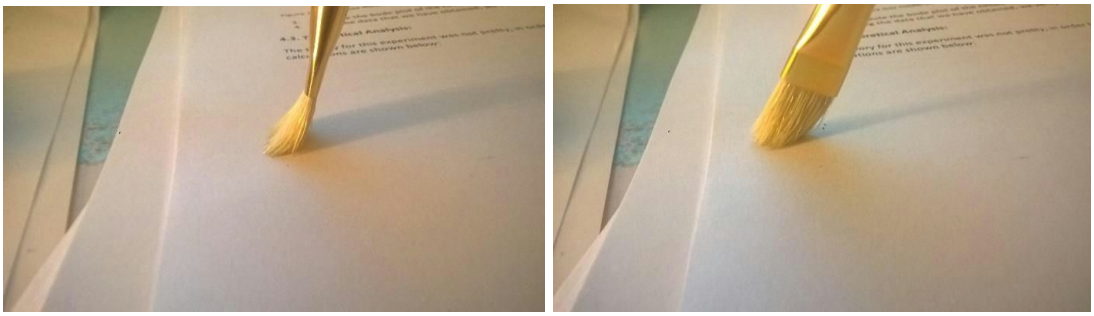

Figure 2. Lengthwise and Widthwise stokes

Finally, by rotating the brush by angles that are not a multiple of 90 degrees, other strokes can be obtained.

5. System Setup

This system consists of four key components. The first component is the electrical subsystem for power along with Arduino Uno for efficient controlling of the servo motors, and thus arm. Control of the Arduino is handled by MATLAB. The second and a key component is the robotic arm structure. The third component is the paintbrush with paint, attached as an end effector to the robotic arm. And last but not the least, we have the canvas over which we create the paintings. All these components are shown in Figure 3.

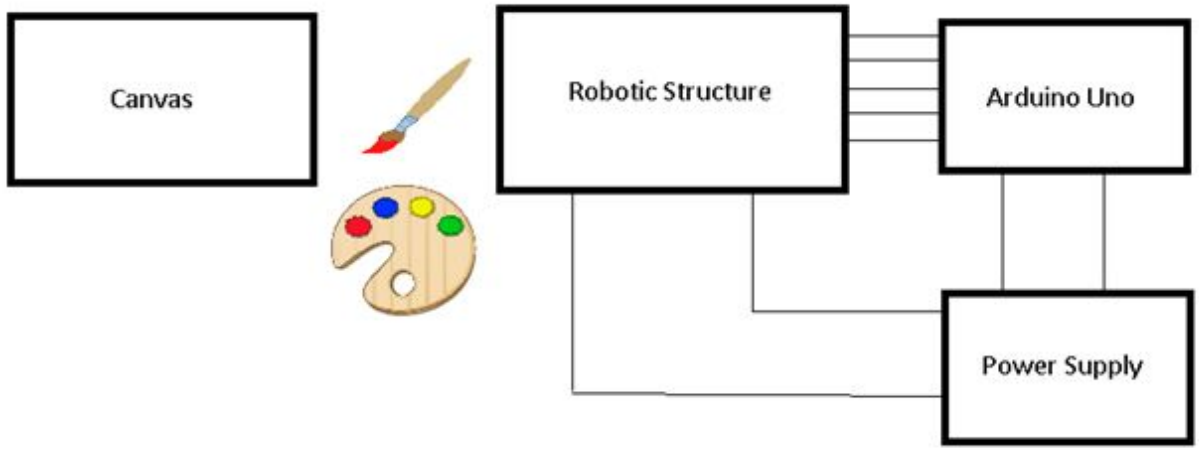

Figure 3. System Block Diagram

6. Robotic Arm Setup

We have used SainSmart 6-Axis control Robot Arm. This arm is made up of PVC materials. It is CNC processed. The arm ships with 6 servo motors and thus has 6 degrees of freedom. However for our project we use only 5 servo motors as the last motor which is used for clamping is not required for our project. The arm is constructed in a manner such that there is a spherical wrist at the end. We replace the gripper that ships with the arm by a paint brush. The entire robotic structure is mounted on a rotatable platform which helps it easier to position it. The arm comes with no installation or construction manual. Thus the user is free to design it as per the needs of the system. We have tried to construct in a manner such that we can get a closed loop. Figure 4 shows the original robot.

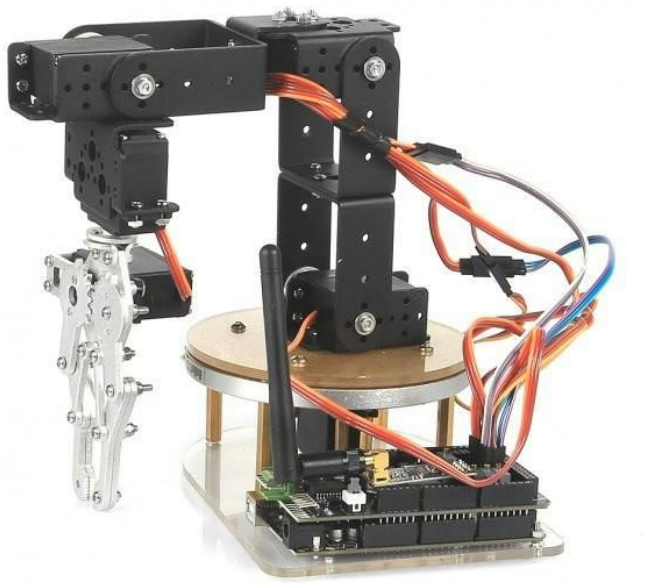

Figure 4.Image showing the default construction of robotic arm structure

Based on our system requirements, we construct the arm such that it is more suitable for the painting task and we get a spherical wrist at the end effector. We then constructed a model of this arm on a simulator with accurate measurements. This model is used for generating all the simulations for verification of calculations. The model is shown in figure 5.

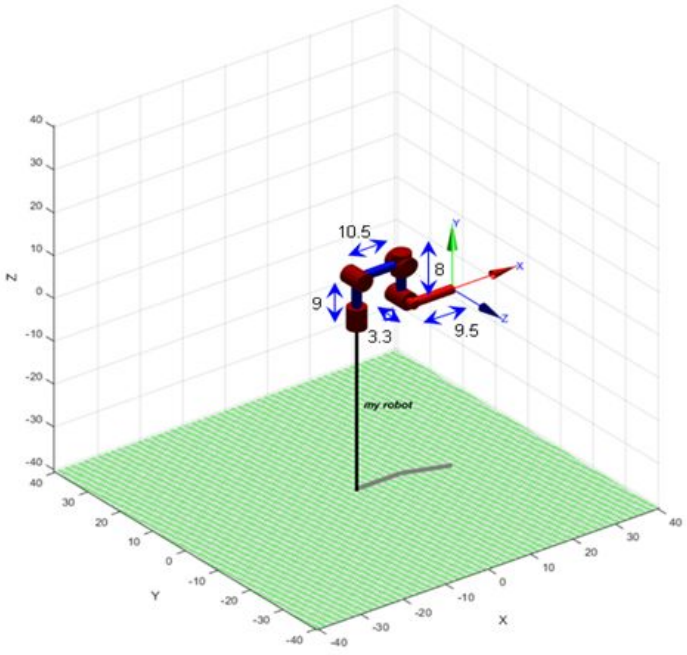

Figure 5.MATLAB based model for the robotic arm showing all the dimensions in cm.

After construction of the arm, we were able to calculate the standard DH parameters for the above robotic model. This is a key step as we need this for efficient forward and inverse kinematics. The DH parameters for this robotic model are shown in the table below:

α	90°	0°	90°	-90°	0
D	9	0	0	8	3.3
A	0	10.5	0	0	9.5
θ	θ1	θ2	θ3	θ4	θ5

Table 2:.Table showing the standard DH parameters for the robotic arm

6.1 Calibration

Calibration is a necessary step in order for the robot to scale and match with (x, y ,z) coordinate of its operational space in terms of base frame. In MATLAB-Arduino package, we needed to specify minimum PWM and maximum PWM, which is different between each servo, even amongst the same type of servos. This has been manually done and empirically tuned. Minimum values and maximum values that the servos accept as the angles are separately kept track. As it is reiterated in the next section, 1° is the best resolution that servo can understand and therefore execute its revolution motion from its current position to achieve the desired position. Therefore, not all points that are within the range of the arm are reachable statically, though it may be dynamically reachable while moving from one position to another.

We defined angle vector that is between 0 and 1, where there are total of 180 points, which each point represent 1°. The angle 0° is be represented as t(1), rather than 0 in this system. All the servos are physically tuned so that it has the same position as the simulation figures shown below when the index of the angle vector t('index') is provided as the angle value in degrees. Note that angle variation with input is not precisely linear, although it can be approximated as linear.

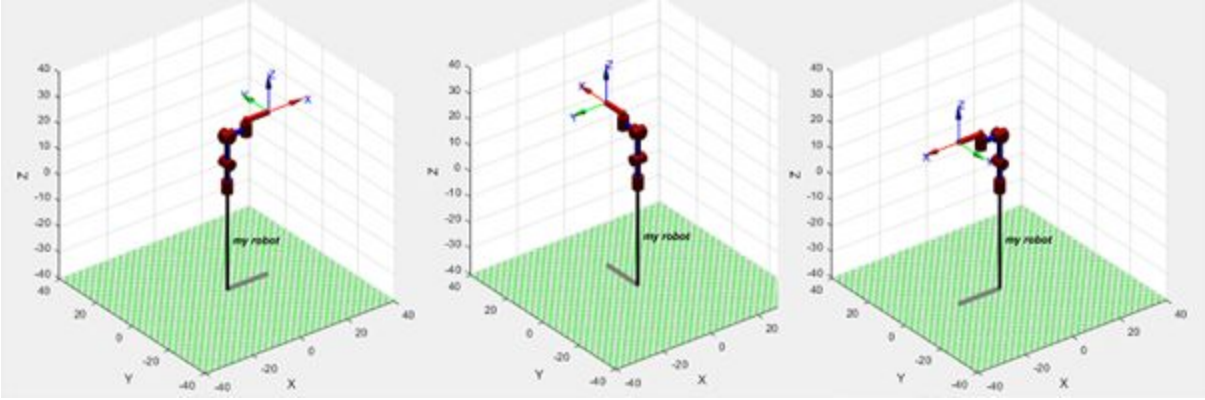

Figure 6. First servo angle (the bottom most) has limitation from 0° to 180° and motion from 0°, 90° and 180°

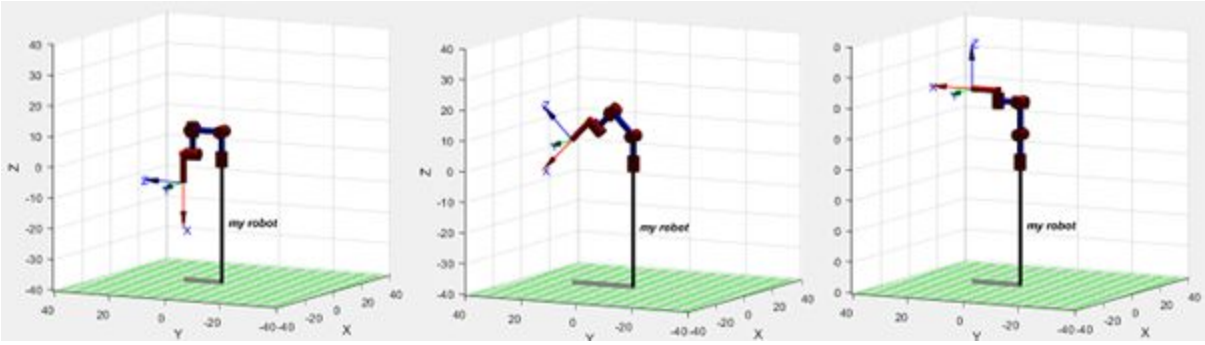

Figure 7. The second servo angle has limitation from 0° to 90° and motion from 0°, 45° and 90°

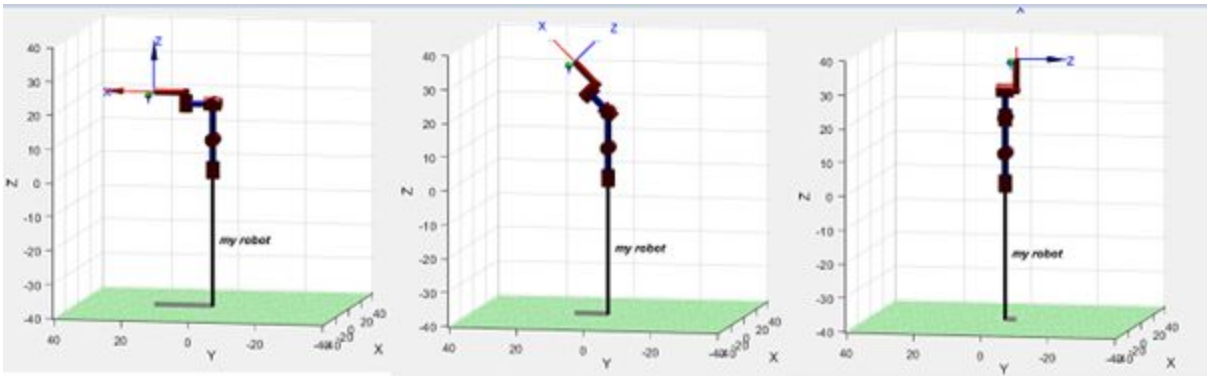

Figure 8. The third servo angle has limitation from 0° to 90° and motion from 0°, 45° and 90° are shown.

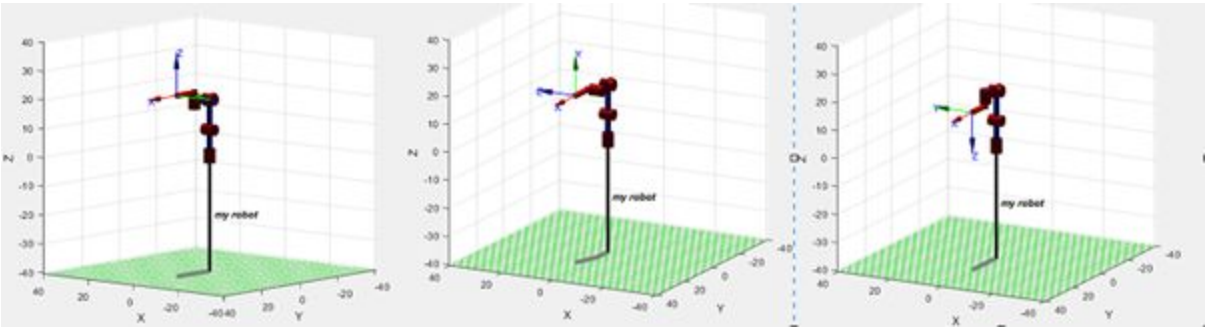

Figure 9. The fourth servo angle has limitation from -90° to 90° and motion from -90°, 0° and 90° are shown.

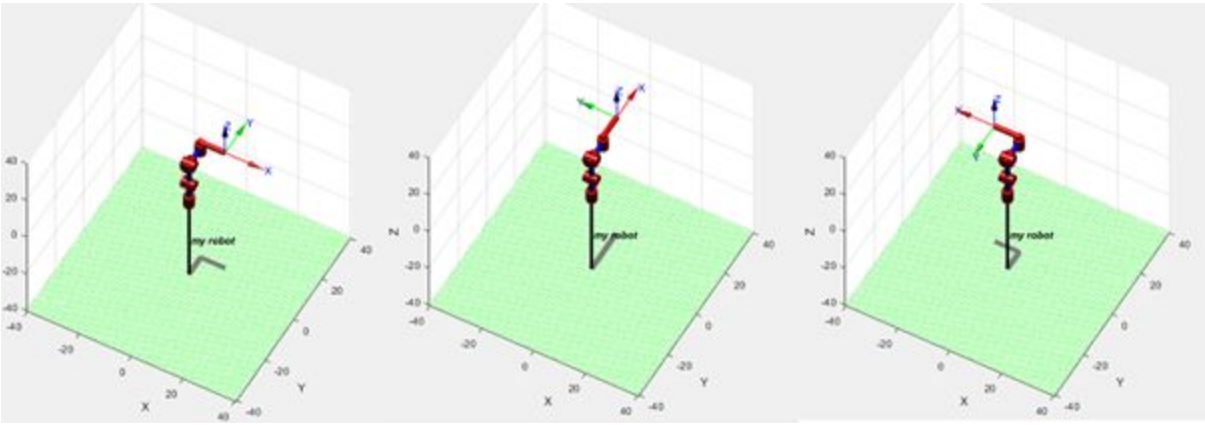

Figure 10. The fifth servo angle has limitation from -180° to 0° and motion from -180°, -90° and 0° are shown below

7. Robotic Arm Analysis

7.1 Forward Kinematics

Forward kinematics can easily be calculated if DH-parameter is found. With the classic DH parameters, transform equation from (2.53, Bruno, Chap 2) can be used:

- $$A_i^{i-1} = \begin{pmatrix} \cos(v_i) & -\sin(v_i)\cos(\alpha_i) & \sin(v_i)\sin(\alpha_i) & a_i\cos(v_i) \\ \sin(v_i) & \cos(v_i)\cos(\alpha_i) & -\cos(v_i)\sin(\alpha_i) & a_i\cos(v_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Equation 1. Transform equation [2]

By postmultiplying transform matrices from the base frame to the end-effector frame, final T matrix that describes the end-effector's end points (x,y,z) and orientation (rotational matrix) in term of the base frame can be expressed as below:

- $$T_5^0 = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4 = \begin{pmatrix} Rotational\ (3X3) & Position(3X1) \\ 0(1X3) & 1 \end{pmatrix}$$

Equation 2 Transform equation that describe end-effector position and orientation in terms of base frame [2]

The resulting matrix, expressed in shorthand notation, for our system is as follows:

$$T_5^0 = \begin{pmatrix} c_5(s_1s_4 + c_{23}c_1c_4) - s_{23}c_1s_5 & -s_5(s_1s_4 + c_{23}c_1c_4) - s_{23}c_1c_5 & c_4s_1 - c_{23}c_1s_4 & \frac{33c_4s_4}{10} + \frac{19}{2}c_5(s_1s_4 + c_{23}c_1c_4) + \frac{c_1(16s_4 + 21c_2)}{2} - \frac{33c_{23}c_1s_4}{10} - \frac{19s_{23}c_1s_5}{2} \\ -c_5(c_1s_4 + c_{23}s_1c_4) - s_{23}s_1s_5 & -s_5(s_1s_4 + c_{23}c_1c_4) - s_{23}c_1c_5 & c_1c_4 - c_{23}s_1s_4 & \frac{s_1(16s_{23} + 21c_2)}{2} + \frac{19}{2}c_5(c_1s_4 + c_{23}s_1c_4) + \frac{33c_1c_4}{10} - \frac{33c_{23}s_1s_4}{10} - \frac{19s_{23}s_1s_5}{2} \\ c_{23}s_5 + s_{23}c_4c_5 & c_{23}c_5 - s_{23}c_4s_5 & -s_{23}s_4 & \frac{21s_2}{2} - 8c_{23} - \frac{33s_{23}s_4}{10} + \frac{19s_{23}c_4c_5}{2} + 9 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

End-effector orientation (rotational matrix) — End effector position (x,y,z)

The result shown above is calculated through simple series of matrix multiplication, which done through MATLAB. As it is introduced in the next section, forward kinematics has reasonable usage and it is necessary to implement the inverse kinematics.

7.2 Operational Space Analysis and Off-Limit Spaces

Both operational space and off-limit space can be found through the use of only forward kinematics. Operational space can be found by using uniform ranges of points for all angles and draw a scatter plot, which is shown in below picture in 3 dimensions.

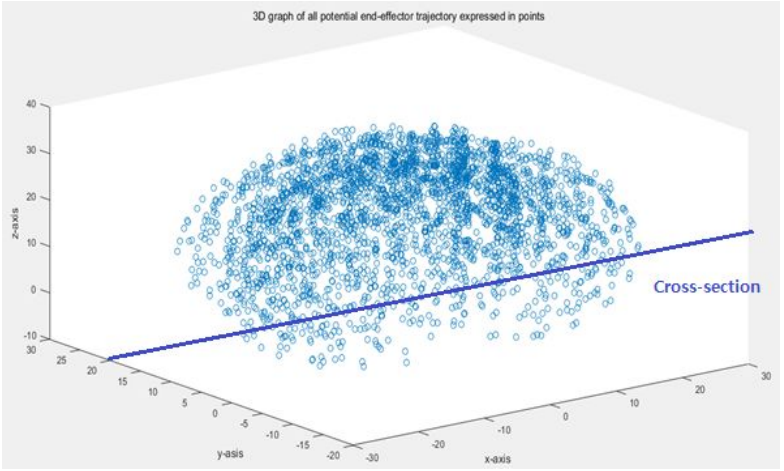

Figure 11. Scatter 3D plot of end-effector potential trajectories.

The figure above is potential trajectories that are represented in a 3D space. As it is, it may not provide information regarding what this may entail. The next figure shows the cross-section at y = 19.5cm, as to show the plane of the canvas showing cross-section reachable space. Determining whether a point in space is within the reachable space can be found by finding the nearest neighbor of the point of interest and see how find the distance. If the distance is more than 2 time longer than typical distance between the points within the reachable space, then the point of interest is determined to be not within the reachable space. This function is implemented in our code, so that inverse kinematic is not performed if the desired position is not within the reachable space.

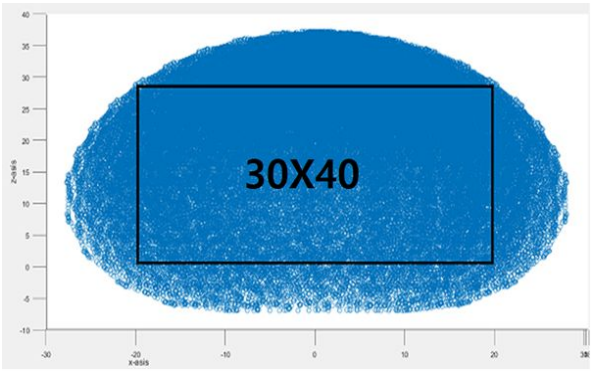

Figure 12. Reachable space at y = 19.5cm plane, not considering the angle resolution of servos.

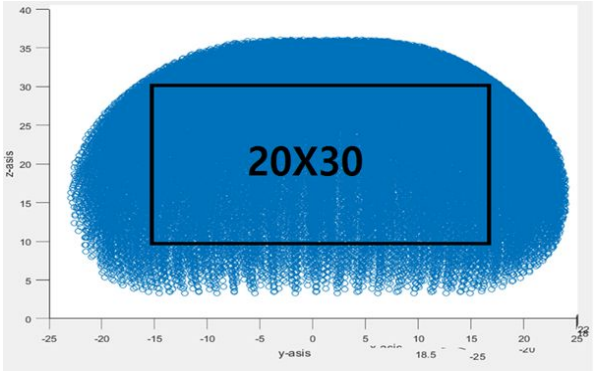

Figure 13. Flexible space at y = 19.5cm plane, not considering the angle resolution of servos

The flexible space is smaller in area than the reachable space, which is defined for this system as the space where all the angle have at least more than 15° before it reaches its joint limits.

The off-limit space refers to the space that the arm should not try to extend itself to. For instance, although the arm has ability to reach areas below the y = 0 plane, which is the floor, there is physical obstacle that can possibly damage the arm by strongly acting against it. Off-limit space can easily be recognized by first seeing whether the angle combination results in negative Z position or far beyond 19.5cm away from the base frame origin, determining from the forward kinematics. However, off-limit space is rarely ever reached, because when we use inverse kinematics to draw trajectories, we always pick points where it is not in the off-limit area of our base frame.

7.3 Inverse Kinematics

Calculating inverse kinematics is absolutely necessary for robotic arm control, because it is much more closely relevant and intuitive than forward kinematics, considering the tasks that this robot arm must perform. Either the end point position or both the position and orientation were specified in order to obtain the solution to inverse kinematics to achieve our drawing.

There are two main ways of calculating inverse kinematics, namely numerical calculation and symbolically solving for a closed form solution. However, closed form solution exists only for some specific structure of arms especially if DOF is greater than 3. Special structures such as 6DOF with a spherical wrist at the end or three consecutive parallel joint can have closed form solution by separately considering the wrist and arm. However, having even a slight offset may complicate the equations, making them unsolvable by hands or symbolic solver programs.

Only solution we had in our 5 DOF arm was to use numerical iterative solution to obtain the inverse kinematics for given information about the end-effector. We have approached in two different ways to calculate the inverse kinematics. The first way was to use analytical Jacobian transpose and iteratively reduce the end-effector profile. Using Jacobian inverse also works, but it has problem when the Jacobian matrix becomes singular, which is not an uncommon incident. Jacobian transpose, on the other hand, does not have this problem and provides solution that approaches the destination the fastest rate, among multiple solutions, in the case where one or more of the orientations are left as "don't cares," or if only the positions of the end-effector are of interest. The advantage of using Jacobian matrix is that it will provide a very accurate result, with a downdie of being very slow to simulate compared to the second method introduced below. Typically it takes about 10 minutes or more to obtain one point with accuracy for iteration stop norm of error 0.00001 with steps 0.0001. The function that performs this action is "get_q.m" in our code. The algorithm is as follows:

1. Enter initial point, initial angle

2. Find target point, initial error

3. If the point is within the reachable space, start the loop.

4. Find analytical Jacobian at current point

5. Let $\Delta Q = K^* (J\_numeric)T * error$

6. Let $Q = Q + \Delta Q$

7. Update new position with new Q

8. Find new error: new position – previous position

9. Repeat 4~8 until desired norm(error) is achieved.

The second way is to use a fmincom function in MATLAB, which the code intends to reduce the objective function: minimize (inv(T)*robot.forwardkinematics(q) - eye(4)) * omega). "robot.fkine(q)" is simply a T (transformation) matrix that is generated by plugging in the q(angle values). Overall the code is very simple and short that consist of only few lines. During development It was discovered that robotic library provided by Peter Corke makes use of the same code with faster convergence. This method, however, suffers from high errors on certain circumstances. To make it worse, it is rather arbitrary when this can occur, so that knowing when to expect high error is unclear. In general the

accuracy is much less than the Jacobian numerical iteration approach and the error cannot normally be, improved beyond some limit, but the computation speed is much faster, providing nearly instant solution. The error obtained is oftentimes large enough to affect painting results, even in simulation. The function that performs this inverse kinematics is “ikcon.m.”

Use of fmincom function is most useful for the purpose of quick painting of single line or simple stroke that does not need to be precise, while the Jacobian iteration approach is more suitable for precise drawing, as the error can be set such that it is very low. The Jacobian method is only appropriate when it is set to move through the fixed number of points in the operational space which its inverse kinematics are calculated and saved beforehand. Of course to achieve a very high quality set of points (dense points) it may take more than months to simply gather inverse kinematic solutions to form tables, especially if it is intended to reach different Y-planes. We have used both approaches for drawing strokes and random figures.

7.4 Feedback Scheme

During the stationary state, the error is assumed to be very small. Noise of the PWM signal or distortions does not cause our system to move in any directions unless it is directed to. In our robotic arm design, feedback is used not because of the noise or distortion, but to move from current location to desired location, which is equivalent to the reference location. Scheme 1 takes advantage of this condition, while scheme 2 is implemented and analyzed in our robot as though when designing a control scheme for a general plant system.

7.4.1 Scheme 1: Simple addition/subtraction controller (Default)

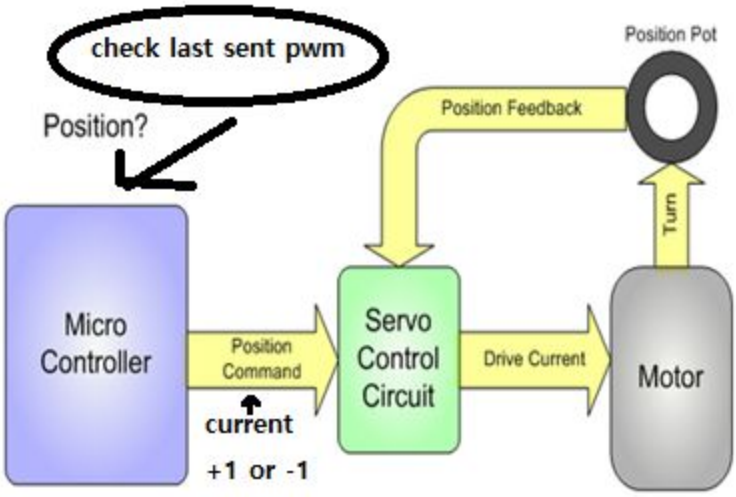

Figure 14. Controller scheme 1 block diagram

Scheme 1 analysis

Although there is no direct connection between the microcontroller and position feedback, MATLAB-Arduino package keep track of what PWM signal it is currently providing to the servos to maintain the current position. Although not very common, electric interruption or physical interruption may result in slightly less accurate reading, which may contribute in further increasing our errors. Moreover, the monitoring of angle difference between desired angle and actual angle is not done in real time, but rather only done during the motion of angle movement initiated by the

command of the user. This can be crucial issue if the system is not robust against disturbance or noise, which it does not apply to our robotic are.

Scheme 1: Algorithm of using the feedback control (1 instance of MATLAB required)

"servo_drive.m" is used:

1. Read the current position through command readPosition() for all servos

2. For each of the servos, find whether the desired position or current position is larger. If former set x = 1 and if latter set x = -1;

3. Find the absolute value of the difference of the desired position and current position for all servos.

4. If any of the servos have error greater than 1°, then add x to its current position parameter.

5. Send in the new PWM command: writePosition(servonumber, PWM_parameter)

6. Repeat 1~6 until the error of all servos are less or equal to 1°.

7.4.2 *Scheme 2: PI controller (Optional)*

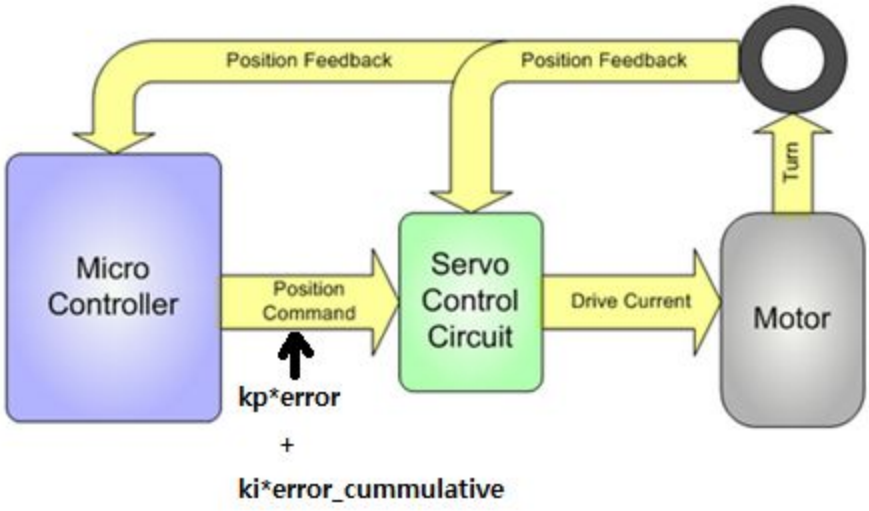

Figure 15. Controller scheme 2 block diagram

Scheme 2 analysis:

Scheme 2 analysis has another additional feedback that is directed to the microcontroller. The value that is read is the analog value from the potentiometer within the servos, which directly addresses the actual positions. It is more accurate than simple readPosition() function on MATLAB-Arduino package that is used for control scheme 1.

Analog value that represents the position can be obtained by simple steps introduced in below figures:

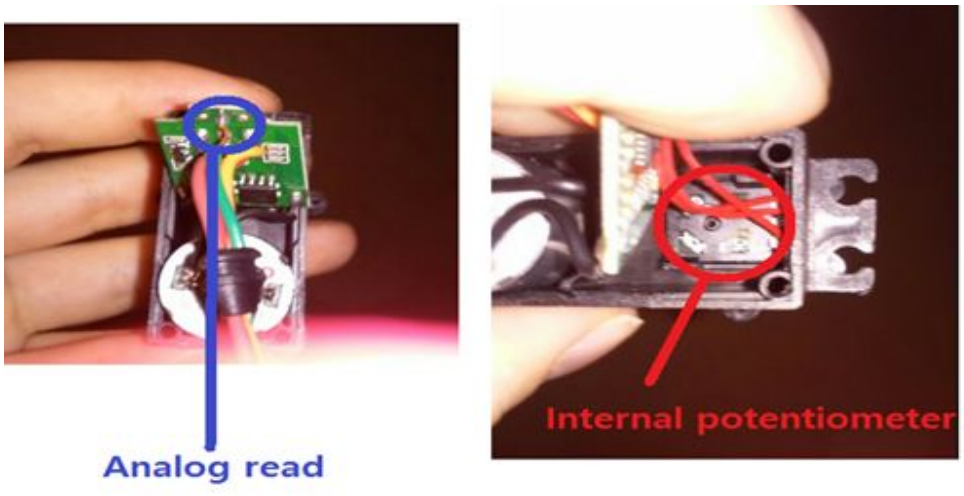

Figure 16. Soldered wired on the analog of the middle potentiometer pin (left) and internal potentiometer (right)

Basically, first open up the servo bottom, then find the potentiometer reading slot and solder an analog read wire to it.

Another feature that is distinct from control scheme 1 is using PI control method, which is the most common type of control schemes in industry. Real time monitoring of current angle and execute movement as soon as it detects that the desired angle is not equal to the current angle is highly responsive and very reliable feature to have in our system. Therefore if the motor has been, for some reason, moved by something other than user command, the robot will detect that change and transform its angle back to where it is supposed to be. This optional control mode is implemented for completeness of our project rather than its appropriateness or usefulness for our purpose, as it is further discussed in later section.

Scheme 2: Tuning PID feedback controller parameters for servos

As mentioned before the servo has a good accuracy on its own, because servo motor itself contains its own feedback system. There is no straightforward or guidelines to calculate K parameters. Rather, manual tuning while observing the output can be helpful if the requirement of the system is not very tight. This means if response time or overshoot is not strictly limited to be fast and very low, respectively, there are ranges of K parameters that are suitable for our system. Unfortunately, we did not have a mathematical function to describe our system, so we have used input/output data to analyze our system and tune parameters, which has been done with the help of MATLAB app called PID tuner. All five servos that consists our arm have been manually tuned to observe their responses. Steps of tuning one of the servos are introduced as follows:

What is presented below is the situation where a servo is commanded to move from 0° to 90°, which is equivalent to 0 to 0.5, out of maximum 1.0, for the arduino PWM input parameter that represents angles.

Step 1: Create the vector of desired data and measured data

Vector of actual value is created simply by executing PWM writePosition() function to go 90° from 0° and read the actual position for 100 times instantly after. This can be done by simply running the

script called “Gather_data.m.” Because the analog reading scale is not the same as the scale that MATLAB arduino uses (0~1), it need to be converted accordingly.

Step 2: Use MATLAB PID tuner app and try to fit the measured data as accurately as possible to desired data. One may use “Auto-estimate.”

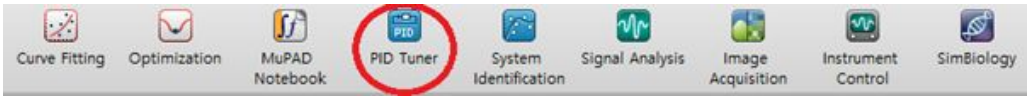

Figure 17. MATLAB toolbar

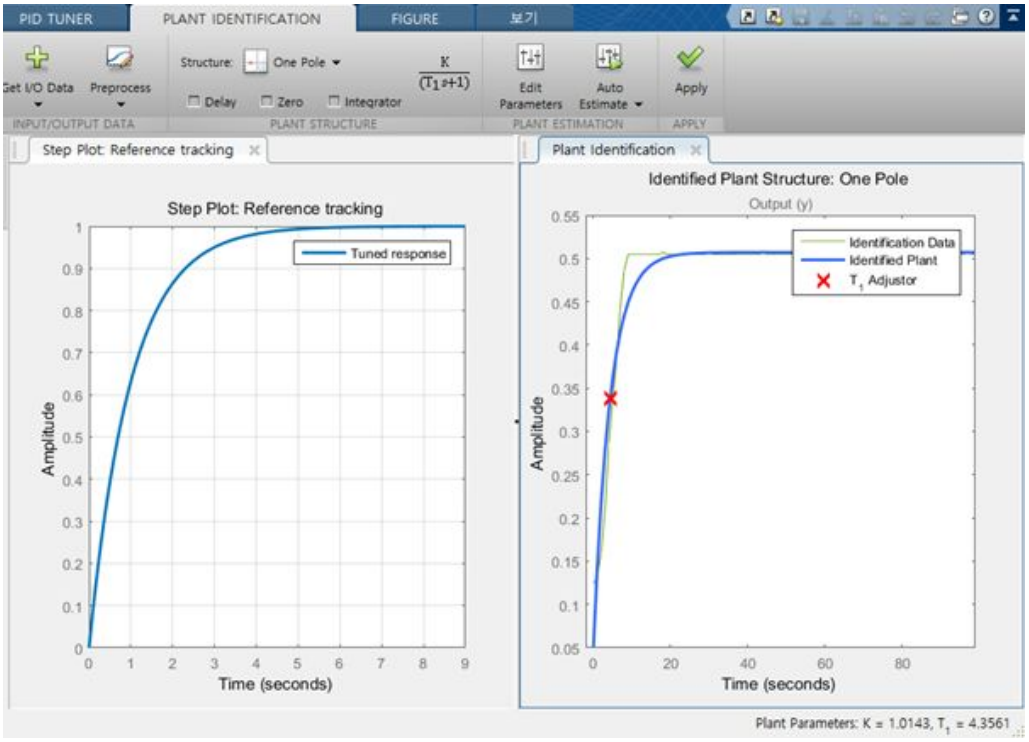

Figure 18. Default setting (left) and curve fitting of the data points to step response (right)

Step 3: Then execute “apply“ to see step response.

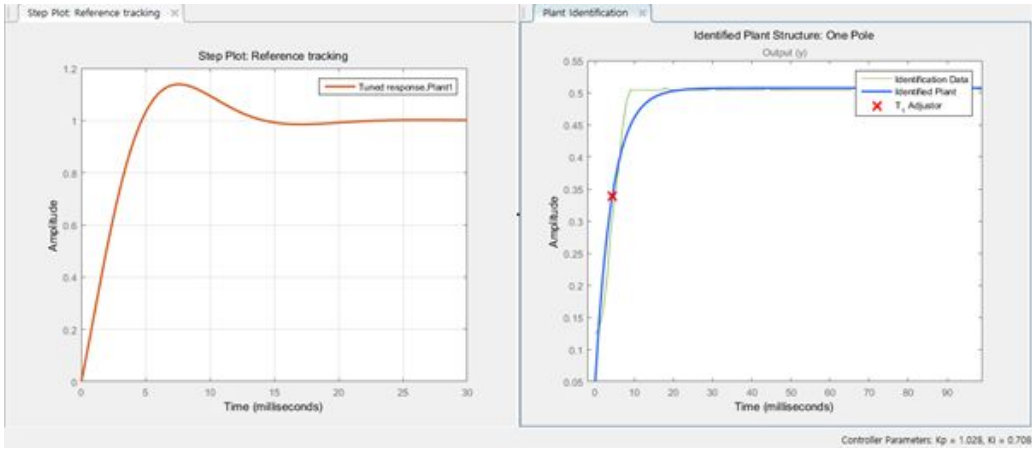

Figure 19. Balanced (robustness VS fast response) controlled output (left)

As you can see, that Plant parameters such as K and T1 are provided on the bottom right corner.

Step 4: Manually tune the system response that fits the specification of the system.

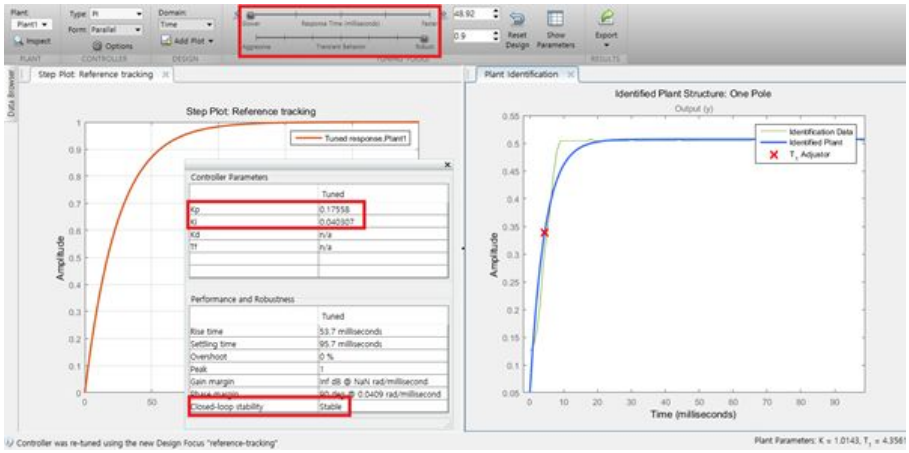

Figure 20. Top box is the tuner, the middle one is the K parameters for PI controller, the bottom one is stability.

Our system prefers slow, but robust system rather than fast or aggressive system. As it can be seen, we tuned in accordance to our preference.

Step 5: Obtain the K parameters from MATLAB.

In our particular case of moving from 0° to 90°, Kp = 0.17558, Ki = 0.040307 and Kd = 0. One may think that these values are too low, but as it is provided by the performance table, it is robust and stable since it converges to a goal value without having any steady-state errors. Lastly, although we tried to PID controller, the coefficient Kd = 0, which means that simple PI feedback controller can achieve our goal.

Tuning has been done for all five servos, because their response is slightly different from each other, even if the servo is of the same type. However, our requirement is not very strict in the sense that we can tolerate some overshoot and slow responses. Therefore same K parameter has been applied to all five servos.

The more significant difference in K parameter tuning comes from how much the current actual angle deviates from the desired angle. Therefore, we used a variable K parameter depending on current error values, as it is shown in the next subsection. Also, to further minimize the overshoot, the accumulation of error is reset, when once the desired position is achieved.

Scheme 2: Resulting K parameters and their justification.

Angle errors	Kp	Ki
0° ~ 30°	0.023834	0.001958
30° ~ 70°	0.056494	0.022334
70° ~ 120°	0.177558	0.040307
120° ~ 150°	0.237431	0.042331
150° ~ 180°	0.27534	0.048704

Table 3. Variable K parameters used in our controller algorithm.

The values are obtained by extracting angle values and changes on different angle error ranges and tuning PID parameters. Notice that Kp and Ki are set to a very small values, because we intentionally aimed to achieve a slow gradual change to achieve the desired angle with small overshoot. Slow response is desirable because sudden movement wobbles the whole arm and severely distorts drawing as our hardware joints are not tightly fixed in space. Fast and accurate drawing is the best possible output, but in our case, it is best for us to have as accurate drawing as possible even if the process is slowed down significantly. Feedback system is used as to let the actual servo follow our trajectories of desired angles.

Depending on how big the angle error is, the Kp and Ki parameter is slightly different. In fact, for these parameters it to be optimal for our systems, the values of K need to be generated for all possible ranges of angle errors rather than lumped angle ranges. However, the difference is not crucial considering that our system can tolerate slow response or some overshoot, as long as our system is stable. In the MATLAB code, above K parameters are used for all five servos, because between each servo, K values did not have big variation.

Even so with the very small value of Ki parameter, integral term can make our system still move even if the error approached zero, causing oscillation or overshoot. In our code, we have done our measure to set the accumulated error to zero once the desired position is achieved. This is only possible because MATLAB is a high level program that can easily do this task.

One must also note that the parameter obtained by tuning through PID tuner app need to be tested rather than blindly trusting them, because the result is approximation based on input and output vectors than a more precise mathematical representation of the system.

Scheme 2: Using PI feedback control as means to control the position in MATLAB

Included with the report and other codes are a script named “PI_CONTROL_ALL_FIVE_Servos.m” and function named “PI_CONTROL_ALL_FIVE_Servos_mode2_sender.m.” For the execution of the robot arm using PI feedback control, two instances of MATLAB is required; one for executing “PI_CONTROL_ALL_FIVE_Servos.m” script, which contains an infinite while loop that looks out for any changes on desired angle of all five servos and one for using the function named “PI_CONTROL_ALL_FIVE_Servos_mode2_sender.m” has the ability to change the angle values.

Scheme 2: Algorithm of using the feedback control (2 instances of MATLAB required) **–“PI_CONTROL_ALL_FIVE_Servos.m” and “PI_CONTROL_ALL_FIVE_Servos_mode2_sender.m” are used**

1. Load ‘data.mat’

2. Wait until “work” == 1 (work = 0 initially)

3. Read the actual servo values by reading the analog feedback voltage (all 5 servos)

4. Convert it to equal scale that MATLAB servo function recognizes.

5. Find errors: desired – actual position

6. Accumulate all the errors for all servos as going through the loop. If error is zero set the accumulated = 0.

*7. Set PWM = angle pwm from 4 + Kp*current error + Ki*accumulated error, which effectively changes current angle*

8. Repeat 2~3 until the errors is very small. Then stop the simulation.

9. Set the 'done' variable to zero

10. Other instances of MATLAB script will set work ==0, so it stops looping.

Comparison of control scheme 1 (simple addition/subtraction controller) and control scheme 2 (PI controller)

Strictly speaking, the control scheme 2 (PI controller) resulted in a similar result as the scheme 1 in terms of robot output, because the performance degradation is contributed mainly by angle resolution and inverse kinematic errors rather than servo angle feedback data accuracy.

Main difference between scheme 1 and 2 is the accuracy of the current position feedback reading. Both are relatively very accurate, but because scheme 2 reads analog value directly from the potentiometer inside the servo, it is more accurate than readPosition() command which merely keep track of what PWM is being output from Arduino. Although, not very often and not by much, the position reading from current PWM setting can be affected by electrical interference, physical interference or weight of joint.

Considering the fact that controlling scheme 1 increments/decrement one degree at a time until the desired error is achieved, it can be considered a variable P controller where Kp*error is always equal to $\pm$ 1 degree, whose reference angle is occasionally not accurate. This approach is expected to be slow and have non-zero steady-state error less or equal to 1, but because of angle resolution limitation of servo, this effect is not easily detected in comparing between scheme 1 and 2.

Simplicity of execution of commands led us to use scheme 1 for the demonstration, but scheme 2 codes is provided along with the rest of the code for completion of our project and it can be used instead. Further advantage of scheme 1 is that angle movement is always 1° at a time rather than variable, so diagonal lines are better achieved with scheme 1.

8. System Design

This section will describe the system design in detail, from both a hardware and software perspective.

8.1 Hardware Design

The hardware components mainly include the following three parts: the robotic arm with servos and rotatable platform, the Arduino Uno and the external power supply. Below is a simple circuit diagram to illustrate the hardware setup.

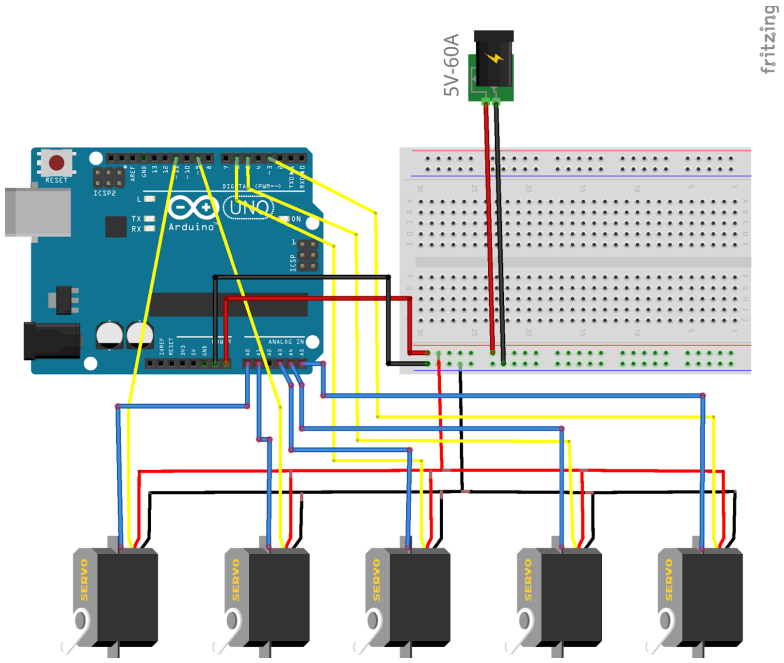

Figure 21. Circuit Diagram of electrical subsystem

First of all, in order to allow the end-effector to be able to operate the brush tip freely in a 3-dimensional space, we must provide the robotic arm with sufficient degree-of-freedoms by having servos attached to it. Originally, the robotic arm we ordered from the website came with 6 servos and each servo would contribute 1 DOF to the system. However, as we were not rotating the brush tip about its approaching direction, the last servo was removed to simplify our hardware design and the corresponding analysis. For the finalized robot arm design, we ended up with a robotic arm with 5 DOFs. In addition, as mentioned in the previous section, the servo feedback signal is an essential mechanism to achieve the precise control over the rotation angle. Hence, we would need to hack the original servo to obtain the reading of its internal potentiometer. Lastly, we also purchased a rotatable platform to hold and anchor the entire arm structure on the table, reducing the impact of arm shaking and making each brush movement more smooth.

Secondly, the microcontroller, Arduino Uno, was chosen for the arm manipulation in our project. To begin with, as the mathematical computation was shifted from the controller board to the host laptop, a relatively complex platform like Intel Edison was a bit over-kill in terms of our usage. Another vital factor to consider was that the current design of Intel Edison only tolerated at maximum 4 concurrent PWM outputs, which was apparently not enough for our robotic arm. Hence, we turned to Arduino Uno instead, which was able to provide us with up to 6 concurrent PWM signals. Moreover, another huge benefit of using Arduino was its full compatibility and support with MATLAB. We will elaborate more on the integration between Arduino and MATLAB in the following software design section.

Finally, an external power supply was necessary to supply sufficient amount of current to the servo. We have done a brief experiment on the current consumption of a single servo that we were using and the result discovered that even with only one servo running, the measured peak current drawing from the power supply could reach as high as ~1 A. This definitely exceeded the maximum current that Arduino Uno could output from its internal voltage regulator. When connected to the servos, the Arduino board would sometimes reset on its own. As a result, a separate (5V-60A) power supply was used to power the servos in this system. Thereafter, it was possible to drive all the servo

simultaneously without observing any abnormal behaviors. The actual hardware setup is shown in Figure 22.

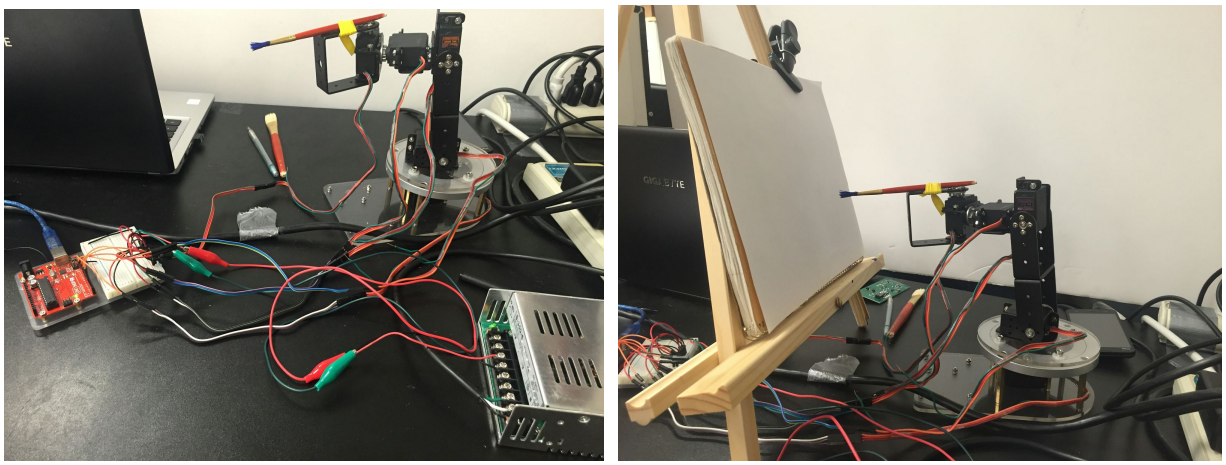

Figure 22. Hardware setup of the system

8.2 Software Design

As can be seen from the discussion in previous section, this system requires a broad range of mathematical computations, such as matrix analysis and numerical iteration, which cannot be processed real-time on an Arduino's microprocessor. Due to this hardware limitation, it made sense to off-load all computation overhead from the controller board to the host laptop. Fortunately, MATLAB offered a convenient library called "Arduino-MATLAB package", which provides the user with an interactive communication with Arduino over the USB without compiling the program. The way how the package worked was, the MATLAB would flash a pre-compiled "agent" program into the Arduino upon library initialization. When successful, the MATLAB could directly send commands to the "agent" running on the Arduino and the agent would perform required actions according to the received commands. With this new workflow, there is no need to re-compile and re-burn the program over and over. Instead, it is possible to leverage MATLAB's console-like GUI to access all peripherals on the Arduino and further utilize MATLAB's powerful mathematical functions for data processing. In particular, the beauty of the MATLAB integration was to enable us to derive the solution to both forward kinematics and inverse kinematics with only a few lines of codes within only a few seconds, sparing us more time and space for the whole system design.

The aforementioned Arduino-MATLAB integration scheme is abstracted in the figure below.

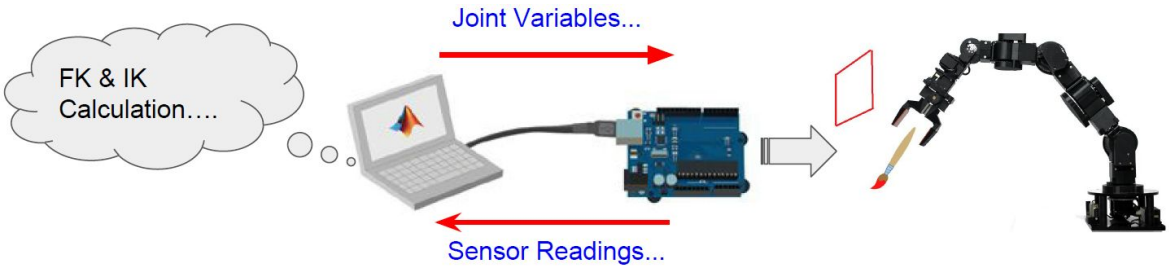

Figure 23. Arduino-Matlab integration.

9. Motion Generation

In this section, we discuss two methods of programmatically generate a set of end-effector coordinates which can then be translated into the actual brush motions.

9.1 MATLAB Motion Generation

The ultimate goal of the motion generation using MATLAB is to mathematically approach real-world strokes by exploring the following tuning features for a stroke: the stroke trajectory, the stroke speed, the stroke granularity and the stroke angle.

To mimic the paint brush trajectory in a painting, we attempted to use a couple of math functions to generate a series of discrete points that represents different types of strokes. The most straightforward example would be a simple horizontal straight line as shown in the figure below. In MATLAB, we could invoke “linspace()” function to generate linearly spaced vector for X, Y and Z axes. Afterwards, we would want to visualize all generated points as a trajectory in 3D space with MATLAB’s handy built-in plotting tool. One thing to note was that there was always a seemingly outlier in the figure, but that far-away point was actually the initial position of the brush tip whose Cartesian coordinates were set to constants (x,y,z) = (0, 3.3, 37).

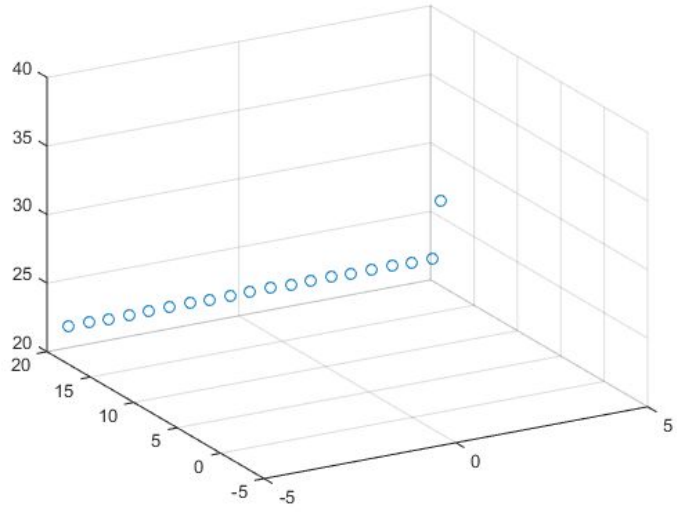

Figure 24. Horizontal straight line. (Linear spacing, 20 points).

Following the same step, we would be able to generate more trajectories that use complex equations to emphasize more stroke details. We would now summarize the math functions that we have used for stroke generation in the table below. One thing to note here was that the number 17.5 in Y axis defined the distance to canvas while the number 22.8 in Z axis compensated the height offset of the robotic arm. Those two constants were explicitly assigned and should not be changed; otherwise, this would ruin the relative position between the brush tip and the canvas.

Some non-conventional trajectories were also tested on our system, such as the batman shape and the heart shape. We will not list them out here as they rely on many sophisticated math transformations.

Resembled Stroke	Formula for X	Formula for Y	Formula for Z
Horizontal Straight Stroke	X = linspace()	Y = 17.5	Z = 22.8
Vertical Straight Stroke	X = 0	Y = 17.5	Z = 22.8 + linspace()
Diagonal Stroke	X = linspace()	Y = 17.5	Z = 22.8 + linspace()
S Stroke	X = linspace()	Y = 17.5	Z = 22.8 + 5 * sin()
Batman	Complex compound functions. Please refer to the corresponding *.m file to for the formulas.		
Heart			
Spiral Square			

After we were capable of generating various trajectories, we soon realized that the spacing between each consecutive points would cause a significant impact on the stroke as well. As listed in above table, the MATLAB function “linspace()” would always lead to a series of points falling within a specified range with equal spacing. However, if we correlate each point we have generated above with a “pause” action in the real-life drawing, it is almost for sure that human painters could possibly pause anywhere anytime during a stroke, equivalently causing such non-uniformly distributed points of a stroke. As a result, to account for this feature, we adopted “logspace()” function in MATLAB to artificially generate logarithmically spaced points, trying to mimic the speed variant in a stroke.

The example of plotting a horizontal straight line is again used in the below figure. Though the final trajectory remains the same, we can now tell the spacing variation within the line. If we further plug this feature into the motion generation and make the robot arm to paint, we will be able to see notable color and texture gradient within the line.

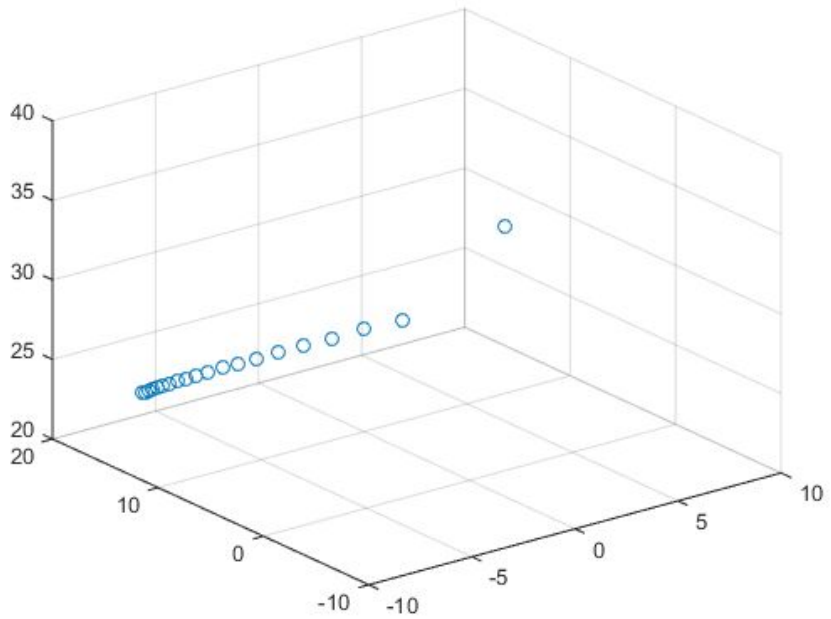

Figure 25. Horizontal straight line. (Logarithmic spacing, 20 points).

Another obvious control factor was the number of points within a stroke. The more points that we could possibly have, the more fined-grained details that we could reproduce. For the two straight lines shown above, both of them encompassed 20 points excluding the initial position. We could reduce the number of points in a stroke by a simple down-sampling technique. Again, in terms of the stroke trajectory, we should be able to obtain the exactly same stroke judging from the programming perspective. However, the eye-catching nature of the painting lies in the variation of stroke speed, angle and force. The nature of aforementioned brush dynamics were really difficult to model with math equations previously. Here, we introduced a systematic way to approximate these features in human painting, allowing us to generate different styles of strokes even with similar trajectories.

Below gives an example of a degenerated horizontal straight line with just 2 points.

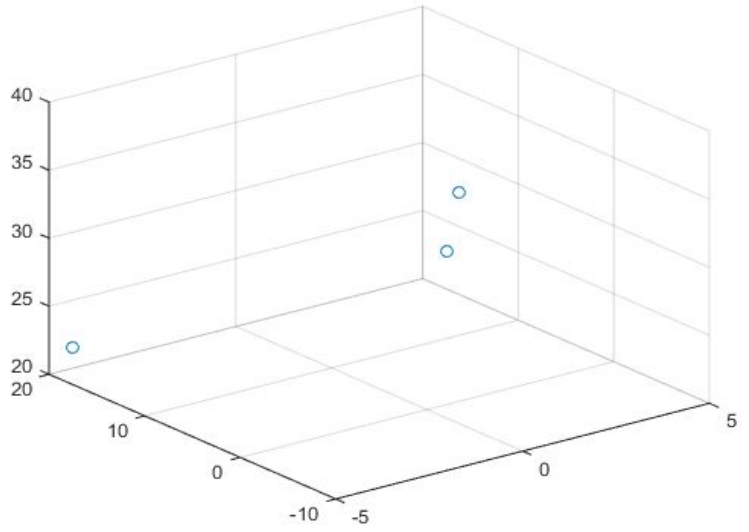

Figure 26. Horizontal straight line. (Linear spacing, 2 points).

The remaining feature that we did not consider carefully was the stroke angle. The stroke angle was determined by the rotation matrix in the forward kinematics. However, in order for our inverse kinematic solver to be able to come up at least a close solution, we would need to manually fix the values in the rotation matrix, implying that we lost the freedom to adjust the brush angle. We will leave this part as a future work if we can refine our inverse kinematic formula to include the rotation term.

9.2 Tablet-Pen Motion Generation

To allow users who are not familiar with MATLAB to interact with our system directly, we explored an alternative method of generating motion patterns for the robotic arm. The result of this exploration is a Windows 10-compatible app that can translate user input in the form of tablet pen strokes into 3d coordinates that are compatible with the MATLAB inverse kinematics.

9.2.1 Windows Inking API

Windows 10, like Windows 8 before it, is designed to support apps, much like how mobile devices do, moving away from the more traditional desktop application. The data collection is made possible using the “Simple Inking” sample application that Microsoft provides to try to encourage developers to develop apps for their OS.

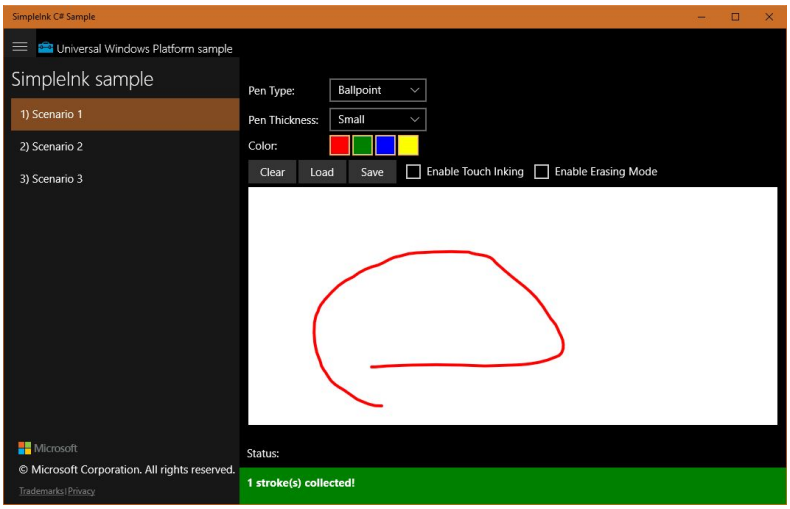

Figure 27: Simple Ink Sample App

By default, the application can only save the user strokes as GIF files with ISF information embedded within them. ISF stands for Ink Serialized Format, a propriety Microsoft file format that is stored in binary format, which makes it not useful for MATLAB, which does not have a means to import these files. Thus, the built in save functionality of the sample application was not used, and a custom save functionality was created instead. The custom save extracts the XY coordinates of the stroke, as well as the pressure values. Strokes are thus saved as CSV files is created, which can be read into MATLAB directly.

9.2.2 Coordinate Processing

As discussed earlier in section 7.2, there is a certain operational space that the robot arm is able to move within, and thus the XYZ coordinates resulting from the user created strokes should also be within this operational space. However, the CSV file gives data based on an InkPoint class that defines axis using “device-independent pixel (DIP) relative to the upper left-hand corner of the inking area.” Attempting to plot the data in the CSV file, it becomes clear that because of this convention, the size of the sketch will be too large, and the orientation of the sketch will be mirrored.

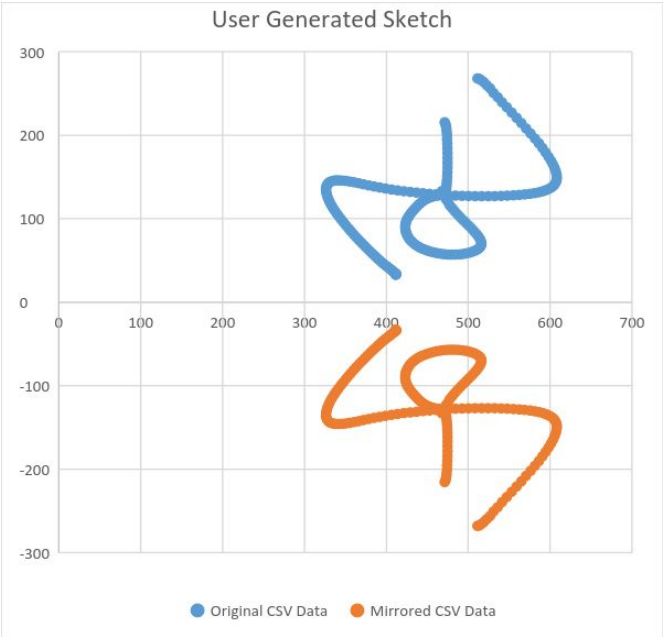

Figure 28: Drawn images

In MATLAB, centering and scaling functions are applied in order to obtain the same shape, but in a region that is reachable for the robotic arm.

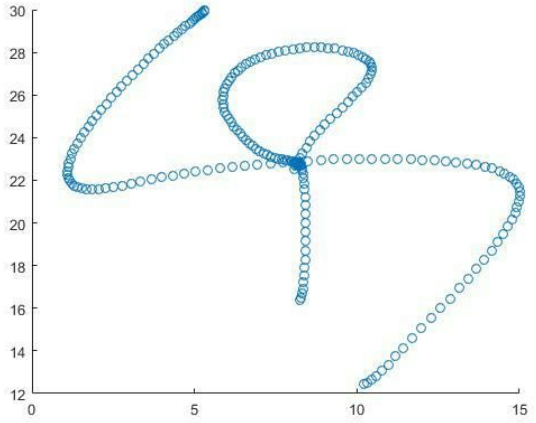

Figure 29: MATLAB centered and scaled data

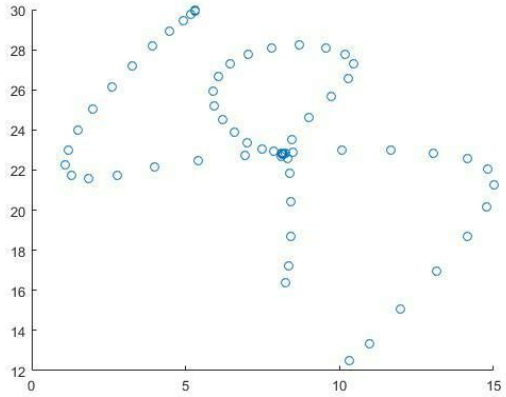

Figure 30: MATLAB Sampled data

Lastly, this image features all of the 250 original points that were collected by the Windows 10 application. However, if the robot arm is instructed to move to each of these 250 points, errors will accumulate quickly and cause the accuracy to deteriorate. Therefore, we will sample the 250 points to a reduced number of points, 63 points in this example. These sets of points are now ready for use.

9.3 Simulation

Lastly, before actual deploying the motion to our robot arm, we would use a “Robotic Toolbox” which is already available in MATLAB for motion simulation. Particularly, we animated the brush tip motion to see whether it confirmed with our planned trajectory. Only two parameters were needed for initiating a simulation: one is the matrix containing the DH parameters of the robot arm while the other was a set of generated coordinates. Afterwards, the toolbox would automatically take care of the remaining computation for animation. This turned out to be an extremely useful tool which provided us a handy way to verify the end-effector motion. Also, with the aid of the animation, we could also better visualize how brush tip is going to move in a 3D space.

10. Evaluation

Evaluation is a critical component of any system design. Our system has multiple levels of evaluation. We first generate the required motion trajectory. A plot of this trajectory is observed and analyzed. This helps us eliminate any mathematical errors in the system. We then use this trajectory with both the inverse kinematics available at our disposal to generate a series of required values of thetas for all the motors in the system. Once we have these values we then simulate the trajectory of the brush in operational space and observe the trajectory for inaccuracies. This gives us insight about the deviation of the actual results from expected results. The generated trajectories are then used for simulating the motion using the MATLAB robotics toolbox based on our robotic model. This will further help in understanding deviations from expected behavior. These set of values are then passed on to the robotic arm and executed which then generates the painting on the canvas. The output is then analyzed for deviations from expectations.

We have tried various possible strokes, designs and patterns. We have tried with a variety of colors and also with various types of paint brushes. This will help us to see the true dynamic variations available to an artist and will add more organic content to the painting. Based on various strokes that we have described, we create paintings which consist of random combinations of these strokes. The location of these strokes on canvas is randomly generated but it is made sure that they lie within the reachable space on canvas. Various well known patterns like heart, batman symbol, spiral shapes etc were also made using mathematical equations. Finally user input was incorporated by having a user to draw a pattern on the tablet and record its trajectory. This was then recreated by the robot using the user input.

11. Observations and Limitations

The physical arm suffers from degraded performance due to many different factors. The two factors that mainly impact our drawing output are angle resolution limitation and inverse kinematic calculation errors of fmincom function.

11.1 Angle resolution

Angle resolution limitation of our servo movement, is by far, the greatest contributor to degrade the performance of our robot. It is caused both by the PWM resolution of Arduino and sensitiveness of the servos to change of PWM.

Unfortunately, the degradation of this angle cannot be predicted with the simulation of the robot arm motion, since the simulator joints have very high resolution angle movement. This severely limits statically reachable space in the canvas plane. Specifically, 1° is the smallest resolution that a servo can recognize to actuate, so any decimal representation of angles may be ignored. This seemingly small angle can affect quite visibly because of the nature of our task, which is to draw on a canvas.

Take for example, we are trying to reach (-1, 18.3, 15.2) in the coordinate axis. To only show the effect of angle resolution we used the more accurate inverse kinematic algorithm (Jacobian) to get the angle that reaches this position (use “get_q.m” function). The resulting angles (in degrees) with the norm error tolerance of 0.00001 (500+ iterations) are:

Angles = [123.7302°, 59.5704°, 0°, -87.1240°, -164.98342°]

With these angles, the position achieved is:

Position = [-1.0000, 18.3000, 15.2000], which is very accurate

However, using Angle values

Angle = [123°, 59°,0°,-87°,-164°]

The achieved position is:

Position = [-0.8999, 18.4468, 14.9464]

The distance between accurate position and position achieved with 1° resolution can be calculated as:

$$\text{sqrt}((1-0.8999)^2 + (18.4468-18.3)^2 + (14.9464-15.2)^2) = 0.31\ cm$$

In painting, 0.31 cm of error is very crucial as it is visible. Note that Y position, which is the position of canvas from the robot arm, is off by 0.1 which is not severe if the brush is used, but if pens or markers were used, the line may be drawn with too much pressure against the canvas that are supposedly held at a fixed Y = 18.3cm away from the base frame.

Even though the position is not statically reachable, almost any space in canvas can be reached dynamically. This means that statically unreachable points can be reached as it moves from one point to another, while it is unable to be reached when directed to that specific point. As a result, a straight line that is drawn with only two points on the end points is much straighter than straight line that is drawn by more than 2 points, which is illustrated below figure.

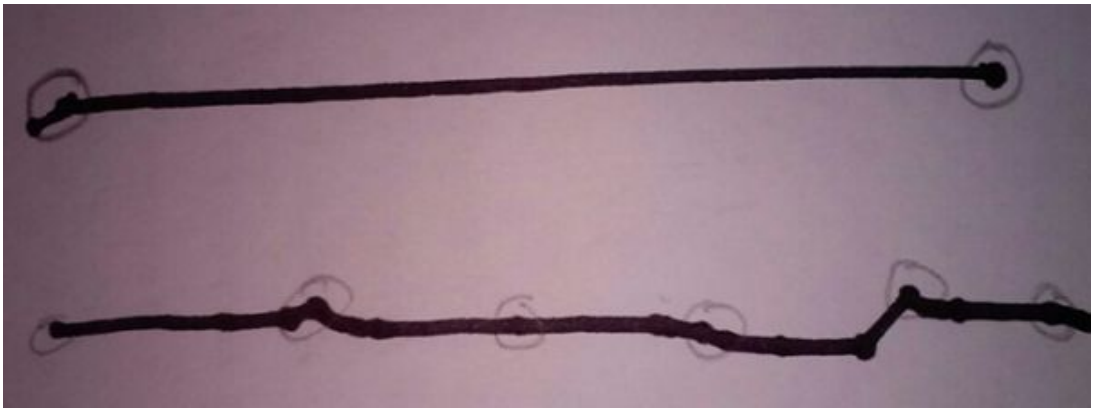

Figure 31. Horizontal line comparison between two point line drawing (top) and six point line drawing (bottom)

Dynamically reaching to point has superior performance, but it may not be easily applicable for shapes that need multiple points to achieve its distinctions. In addition, it defeats the purpose of drawing a line that shows “organicness” of drawing which we are trying to achieve with our robot arm.

In general drawing is done by series of trajectories especially when it contains curves or distinctive end points. Therefore, we are always faced with the trade-offs: 1) simple, highly accurate, but not an organic looking drawing versus 2) complex, inaccurate drawing that contains organic properties, which all bring about the decision of how many points to be used for the drawing of some given shapes. For the demo purposes, we have chosen to maintain “organicness” of drawing lines, as opposed to simple and accurate drawing of lines, as it can easily be completed by plotters, regular printers or Xerox machines.

11.2 Inverse Kinematic Errors

Inverse kinematic error can be considered negligible with the use of accurate Jacobian transpose inverse kinematic algorithm. However, usual time it took to calculate just one point led us to use inverse kinematic algorithm that use fmincom function, despite its inaccuracy. To maintain “organicness” we usually require 50+ points for some particular shapes and the Jacobian inverse kinematic method may take hours for us to just practice some lines and shapes. One must note,

however, output of our robot is still dominated by the error caused by angle resolution, so merely improving inverse kinematic error does not visibly improve the output.

Below figure demonstrates the effect of inverse kinematics error:

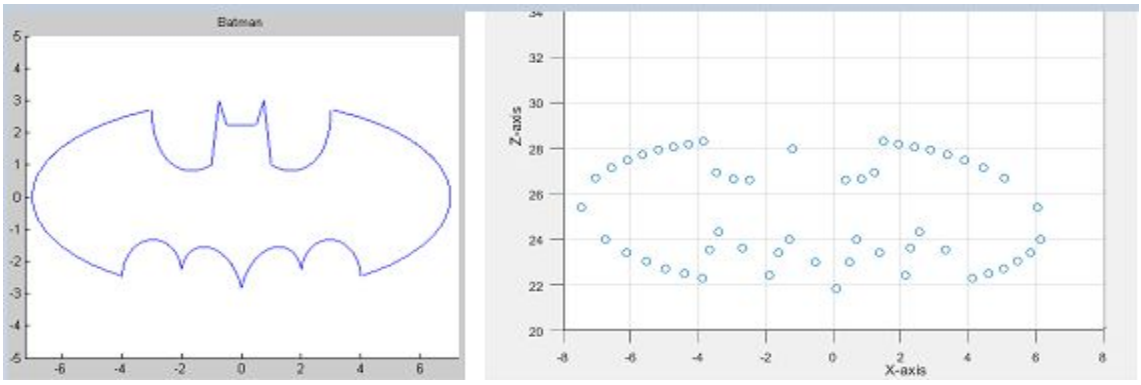

Figure 32. Batman figure intended to draw (left) and scattered trajectory points expressed in MATLAB (right)

Note that scattered representation of trajectory is represented in a MATLAB plot rather than in an actual drawing, to emphasize only the effect of inaccurate inverse kinematic calculation. As it can be seen from above figure we have generated trajectory points to draw a batman symbol. This seemingly looks quite accurate in a x-z plane. However, if one were to examine y-z plane:

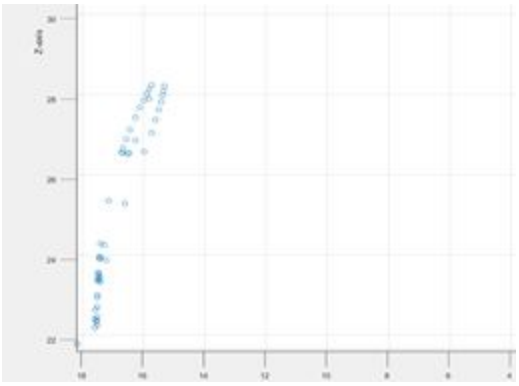

Figure 33. x-y plane of the batman trajectory scatter points.

As the figure shows, the trajectory points does not have an accurate y value, which is the plane on the canvas that we are drawing on. This affects our drawing in that some part is not drawn properly and some parts are drawn with too much pressure causing zigzags for having high friction against the canvas by the brush. It can be recognized that it somewhat employs the characteristic of an arm that only has revolute joint, as it is not very straightforward to draw a flat straight line against the plane of the canvas. One should note that this effect is purely by inverse kinematics inaccuracy as it is shown in a MATLAB figure where is does not consider angle resolution limitation.

11.3 Robotic Kit Properties

The robotic kit that we have purchased is not necessarily made for drawing. The joint fixture is not robust enough to sustain its firm position causing the wobbly behavior of the end-effector irrelevant to previous two errors discussed. As it has been mentioned in the previous subsection about feedback, we have mitigated the problem by slowing down its motion by either adding delay function or making the system respond very slowly. The faster or the more impulsive the motion is, the more

errors in the output. In fact, without our mitigation this could have been the most dominant factors that contributed to the degradation of our output. However, the wobbly behavior is not completely eliminated and it is a legitimate contributor to output performance degradation.

11.4 Lack of Prismatic Joints

Having only the revolute joint severely limits the range of the arm motion in addition to being disadvantageous against drawing directly on the canvas that is flat against the surface of the wall. In fact, drawing a horizontal straight line can be easily done if there was a prismatic joint such as a set of wheels under its platform (wheels that can only move from left to right without any directional changes), not only because it provides alternative options to arrive at the desired position but also has repeatability on any distance along the Y-plane. Note that some of the strokes are not repeatable because depending on the location of the canvas, the orientation of the brush is not necessarily free if it is located near the edge of the reachable space.

11.5 Nonlinearity

First of all, the contribution of nonlinear actual angle movement response with the input value is not the major factor that contributes in the degradation of our output. As it has been briefly mentioned in the previous section on calibration, the angle increase is not necessarily linear, but it can be approximated as linear. For instance, 0° to 90° does not necessarily have the same increment of input parameter (for PWM signal) as 90° to 180°. For smaller scale movement, it may be approximated as linear and in most of our paintings, angle does not move much beyond 90° at one time.

12. Results

In this section, we will be discussing on the static results we got from our robot arm. For demo videos, please refer to the demo section. Link to source code can be found in references section.

12.1 Horizontal Straight Line (Linear Spacing)

As can be seen from the figure 34, this is the horizontal straight line with 20 equally-spaced points. We can see that the brush motion is pretty smooth and harmonic. Furthermore, if we take a closer look at the color gradient in the pigment, the paint texture flowing from the right to the left clearly indicates that the brush tip was indeed scratching the paper from the right to the left.

In addition, we could mix two different paint colors to the brush to generate a gradient colored stroke.

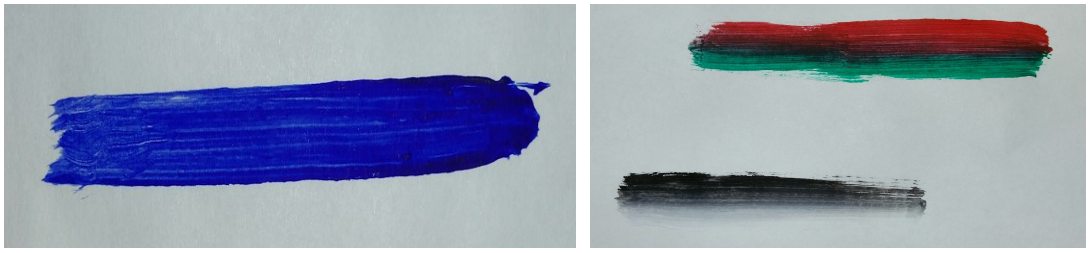

Figure 34. Linear-spaced Horizontal Straight Line

12.2 Horizontal Straight Line (Logarithmic Spacing)

As mentioned in the motion generation section, we used logarithmic function to generate a straight line with 20 unequally-spaced points for modeling the speed variant of a stroke. Different from a normal straight line, we can actually tell that the brush tip started drawing from the right-most side with a slower initial speed. This makes perfect sense as the points containing in a logarithmic-spaced line are denser among the beginning area of the line. Then, as the spacing between consecutive points increases, the brush tip gradually speeded up its motion until the stroke was finished. Equivalently, we created a way to mimic painter's various brush tip speed during painting.

Another observation is that compared with the stroke 12.1, there are some tiny spikes introduced into the strokes. This is due to the error arising from the inverse kinematics and the changing dynamics of the brush tip. In spite of its organic looking, this actually makes the stroke look more natural to audiences.

Figure 35. Logarithmic-spaced Horizontal Straight Line. (Single color v.s. Dual Color)

12.3 S Stroke

As shown in figure 36, a sinusoidal function was used to resemble a S stroke. Needless to say, there is still some imperfection in the stroke we draw, which can be again ascribed to the inaccuracy of inverse kinematics and servo controls. However, this serves as a justification that our system can indeed re-create complex geometric patterns on the canvas given its mathematical function. Again, we could combine two dissimilar colors together to produce a single s-stroke.

Figure 36. S Stroke (Single color v.s. Dual color)

12.4 Random Drawings

After we were able to draw some basic strokes, we attempted to combine all of them into a single painting. Basically, we used rand() function in Matlab to generate a random coordinate within the reachable space for placing the stroke. Of course, we could also randomize the parameter for each individual stroke, such as the direction of the straight line and the amplitude of the s stroke, to create a more diversified painting. As can be seen from the below two sets of random strokes, each drawing

is now a unique masterpiece since there is merely possible that one set of randomly generated strokes duplicates another set of random strokes.

Figure 37. Randomly generated strokes-Set 1

Figure 38. Randomly generated strokes- Set 2

12.5 Other Shapes

We also made use of some existing math functions to reproduce a more meaningful drawing. For example, a heart shape and a batman shape. Since all shapes would correspond to a particular math function, following the same logic, theoretically we should be able to generate any types of strokes as long as it can be represented by a math function.

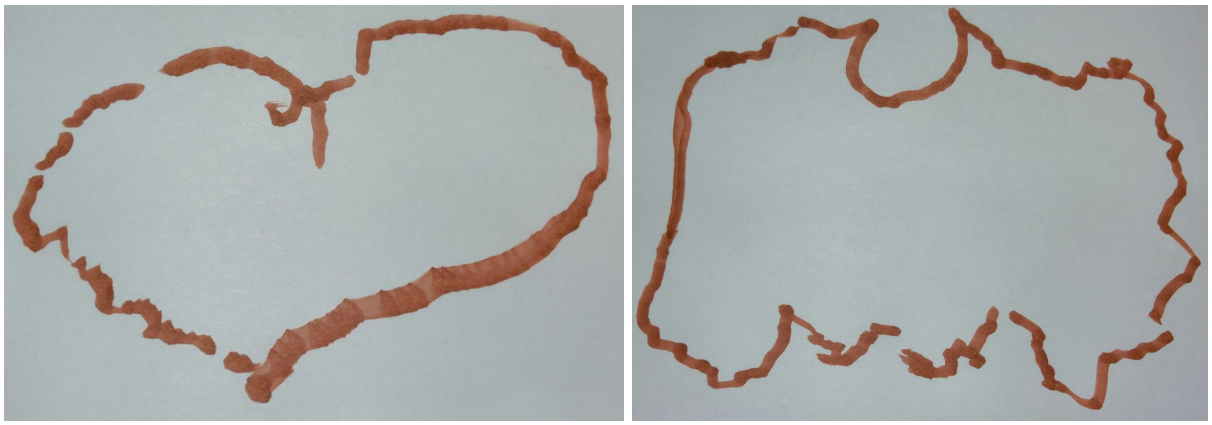

Figure 39. Heart and Batman.

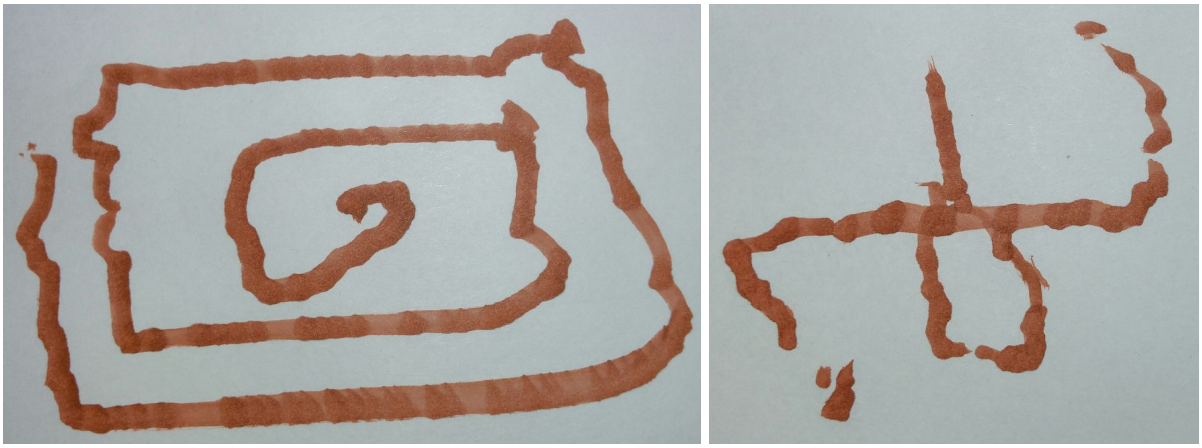

Figure 40. Spiral Square.

Figure 41. Professor's drawing

13. Future Directions

Based on the different types of limitations that we faced during our project, there are many potential directions to pursue further work.

13.1 Painting Related Work

13.1.1 Paintbrush Choice

As mentioned above in the section about painting design decisions, a choice was made to use the round and flat tipped paintbrushes in this project. Furthermore, when the flat tipped brush was used, the orientation was changed manually. Because of this decision, there is no need to have a servo that allows the paint brush itself to rotate about its center axis, reducing the complexity of the system. However, there are still six other types of paintbrushes that can be used to make strokes. This means that for any of the strokes motions mathematically generated using the round paintbrush, execution of the same stroke maneuvers with other paintbrushes would give 6x more possible strokes realized on the paper. Adding the rotation of the brush in would have another axis of rotation to consider mathematically, as well as requiring a different end effector to hold the brush. This could be very worthwhile if more advanced painting was requested of the robot.

If the system is meant to make use of different paint brushes, changing the paintbrushes out manually, as is currently done in the system, may not make much sense. In order to allow the robot to automatically change the paintbrush when given certain strokes to paint, a new, custom end effector that is capable of picking up and orienting the paintbrush could be designed. It would need to be more than just a simple gripper/pincher claw because the brush must be held in the same configuration every time.

13.1.2 Paint Mixing

The aesthetic of a painting does not depend solely on how the paintbrush moves in 3-space, but also on the paint on the brush. What is the total amount of paint transferred from the pallet to the brush? What is the wetness or dryness of the paint? What is the thickness/viscosity? How is the paint distributed on the brush, if not uniformly distributed? Based on these questions about paint, the same motions will produce different strokes on the canvas. Because this was not a focus of the current project these questions were left unanswered, but a follow-up project could do work in that direction. Controlling the arm so that it can take these factors into account when it replenishes the paint on the brush would be a good challenge.

Closely related to these questions is the issue of color. The current system does not have any notion of the color of paint that is on the brush. Key in painting is the mixing of different colors of paint to create more colors, or multiple distinct colors on one brush to create a gradient. An entire separate project could be to create a robot whose only purpose is to act as an assistant that will create a certain color of paint on demand for a human artist to paint with. Such a robot would need to have a very detailed awareness of how colors change as different colors of paint mix together; even a very experienced artist may not be able to mix the same two colors the same way each time.

13.1.3 Paint Choice

For this project, acrylic paint was used by our group in order to achieve realizable results in a short period of time. However, oil paints are very important in the world of painting. The Mona Lisa, perhaps the most famous painting in the world, was painted using oil paints. That being said, if the brush is the same, and the oil prepared to the same consistency as the acrylic, the differences in any individual created stroke may be characterizable, but we suspect it may not be drastic. It is difficult to say until actual work is done.

13.2 Robot Motion Work

13.2.1 Motion Planning

As mentioned in the final presentation, the stylus used to create custom strokes does not feature the tilt sensor that is present in the Apple Pencil. It would be a very good challenge to add the user's recorded tilt into the motion of the robot arm as it follows the xy position and the varying pressure (translated as z distance from canvas).

More ambitious would be to have the Hollywood-style full body motion capture system. The reason it may be useful to have more information than just the way the paintbrush is being held is because of one last piece of information we learned talking to the artists at Blick Paint Supply. Classically trained painters learn that when they paint, they should not paint using only their wrists. Instead, they should use the entire support structure of the arm and generate movements up from the shoulder. If the

future work intends to imitate painting as a human performs it, it would make sense to capture the full information about how the human is moving.

13.2.2 Canvas Size

A standard size of 8.5x11 sized was used for the creation of paintings for this project, however as discussed earlier there are a multitude of canvas sizes to choose from. Because our robot had a limited field of motion, the full 8.5x11 cannot be used. Using a different arm with a larger field of motion would allow painting with larger canvases.

13.2.3 Canvas Positioning

In this project, due to inaccuracies in the theoretical inverse kinematics, and due to the fact we only had revolute joints, we encountered the issue mentioned earlier where the arm would paint in 3D space even when it was instructed to paint points on a 2D plane. As a result of this, the position of the canvas would need to be adjusted to compensate for this inaccuracy. In future work, canvas adjustments could be accomplished through a separate actuation system that could move the canvas as well as the arm. Alternatively, adding prismatic joints as opposed to only having revolute joints could also be a way to improve this issue.

14. Conclusion

In this report we have described in detail, our motivation, problems/challenges, system design, implementation, hardware, robotic analysis and evaluation through observation and demonstration. We have explained our design choices and justified in terms of its trade-offs and appropriateness for our usage. Because of the task of the robot is painting, our requirement and preference is different from general purpose robots or any other types of actuation systems.

Our original intent was to achieve a drawing that has similar level as that of human's. Unfortunately, we were not able to achieve the level of drawing quality that we originally have planned mainly due to hardware limitation that we did not expect prior to purchasing our robot kit, which we believe is not necessarily made for drawing that requires extremely sharp and dense angle resolution of servos. The limitation of our arm is mainly caused by low angle resolution of each servo, lack of robustness in our joints and no clear-cut inverse kinematic solutions. We have however, explored what is feasible with our robot and analyzed its motion and limitations along with reasonable explanation in detail.

There were some design decisions, such as choosing the method of inverse kinematics, whose trade-offs were the speed and accuracy of the calculation. In case of feedback scheme, PI controller, although more accurate, efficient and robust for most of robotic or plant systems, did not necessarily generated better performance than simple add/subtract feedback scheme due to the existing servo angle resolution and inverse kinematic accuracy problem. Furthermore, its requirement to have two instances of MATLAB running continuously to detect changes in desired angle and its inflexibility to use delay statement freely in the code, made it cumbersome for us to use this scheme for demonstration and practice purposes.

We believe that improvement from our current robot can be achieved with the use of more robust robotic arm, including different types of joints, higher resolution of servo motors, fast and accurate inverse kinematic algorithms. In fact, having a wheel below the platform of our arm can highly improve our drawing as it add prismatic movement to our robot arm that can improve the freedom of our robot arm's end-effector location and orientation. Furthermore, our analysis on our robot can be

further analyzed by including forward and inverse dynamics of systems. Although our task at hand is not based on lifting different items with weight, so kinematic analysis is enough to implement drawing robot system, it can still provide us with insight about why the motion is generated and understand how the force exerted can be controlled by means of controlling motion of servos.

Overall, we have realized that seemingly simple task, such as drawing some picture, can involve much more mathematical, programmatic, and physical complication far beyond what one may understand from a mere glance. From the practice of this quarterly project, we have highly familiarized with robotic kinematics, controller schemes and trajectory planning aspect of robotics. We have understood now, that we need to be very careful in choosing appropriate hardware and why manufacturing one's own hardware is much better despite the time consumption of doing so.

Impact of our project can be understood as generating "organicness" of expression as to be distinctive from printers or simple sketching robots. People commonly believe that as the technology develops art and values of humanity may diminish. However, art performing robot such as ours, or music performing robot potentially dispute this conception. Painting robot might be the next generation of currently available sketching robot if it could achieve the accuracy that people can satisfy. In addition, it may help artists to assist their job as well as demonstrating painting skills as instructive or educational purposes since robots are usually excellent at repeating the same actions.

15. Demos

In this section, we will provide the youtube links to some selected demo videos we have.

Stroke Type	Simulation	Real Stroke
Horizontal Straight Line (Linear Spacing, 20 points)	Link	Link
Horizontal Straight Line (Log Spacing, 20 points)	Link	Link
S Stroke (20 points)	Link	Link
Batman Shape (55 points)	N/A	Link
Heart Shape (33 points)	N/A	Link
Spiral Square Shape (50 points)	N/A	Link
Random Drawings #1 (7 strokes)	N/A	Link
Random Drawings #2 (9 strokes)	N/A	Link

16. References

Source Code for this project:
https://drive.google.com/open?id=0B6pkVDXlu_QRWWl2bFdwQ1ZxWWs

[1] L. Garg, "Drawing Robot," 18 03 2016. [Online]. Available: http://www.instructables.com/id/Drawing-Robot/?ALLSTEPS. [Accessed 18 03 2016].

[2] Aleator777, "Pointillist Painting Robot Arm," 18 03 2016. [Online]. Available: http://www.instructables.com/id/Pointillist-Painting-Robot-Arm/. [Accessed 18 03 2016].

[3] Veronica, Interviewee, *Fundamentals of Painting.* [Interview]. 23 02 2016.

[4] J. O. Moore, "Painter's Reference: A Guide to Common Art Canvas Sizes," 24 5 2015. [Online]. Available: http://www.craftsy.com/blog/2015/05/art-canvas-sizes/. [Accessed 18 3 2016].

[5] T. McArdle, "artist paint brushes," 2016. [Online]. Available: http://www.art-is-fun.com/artist-paint-brushes/. [Accessed 18 03 2016].

[6] B. Earl, "Using Feedback," Adafruit, 04 05 2015. [Online]. Available: https://learn.adafruit.com/analog-feedback-servos/using-feedback. [Accessed 18 03 2016].

[7] B. Siciliano, L. Sciavicco, G. Oriolo and L. Villani, Robotics: Modelling, Planning and Control (Advanced Textbooks in Control and Signal Processing), Springer, 2009.

[8] MathWorks, "Arduino Support from MATLAB," MathWorks, [Online]. Available: http://www.mathworks.com/hardware-support/arduino-matlab.html?refresh=true. [Accessed 18 03 2016].

[9] Microsoft, "Simple inking sample," GitHub, 9 03 2016. [Online]. Available: https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/SimpleInk. [Accessed 18 03 2016].

[10] D. Hale, "Store and retrieve ink strokes," Microsoft, 26 02 2016. [Online]. Available: https://msdn.microsoft.com/en-us/windows/uwp/input-and-devices/save-and-load-ink. [Accessed 18 03 2016].

[11] Microsoft, "InkPoint class," Microsoft, 18 03 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.input.inking.inkpoint. [Accessed 18 03 2016].