**Project 2:** Pictionary



**Author:** Siva Chinthalapudi

**Course:** BME 5425

**Description:** Introduction to Robotic Systems I

**Partner:** Rocky Saxena

### Table of Contents

Contents

[Abstract 3](#_bookmark0)

[Introduction/Problem Description 3](#_bookmark1)

[Numerical Solutions 4](#_bookmark2)

[TPOLY 4](#_bookmark3)

[Methods 5](#_bookmark4)

[Hardware 5](#_bookmark5)

[Software 9](#_bookmark6)

[Results 13](#_bookmark7)

[Discussion/Conclusion 14](#_bookmark8)

[Appendix 15](#_bookmark9)

[GUI Code 15](#_bookmark10)

[Robot\_class code 25](#_bookmark11)

[Reference 26](#_bookmark12)

# Abstract

In today’s industry there is a push for interfacing hardware and software together, many employers are looking for talented individuals who can work with both hardware and software. This project builds on the principles of project 1 by using an Arduino to connect to MATLAB and move a robotic system. The robot was tasked to draw certain shapes and 2 players were asked to guess what the robot was drawing (Pictionary). The problem that arose was that the robot was not drawing the shapes smoothly, it was drawing in a jerking discontinuous way which made it difficult for the player. This problem was solved by using TPOLY function in MATLAB that used a quintic 5th order polynomial to allow for the robot to move smoothly. There was an increase in smoothness, however, issues arose with friction between the clip board and the pencil resulting in the pencil falling. A solution to fix this was using overhead projector laminated paper and marker to decrease the friction. The result of interfacing both software and hardware allowed the robot to draw nice smooth lines, while utilizing Arduino push buttons to create a game interface that worked synergistically. The techniques utilized in this project included object- oriented programming, using hardware (Arduinos, breadboards, circuits), and using TPOLY as an artificial intelligence tool. These techniques are excellent because knowledge of these topics can be used in a variety of stem fields including surgical robotics, machine learning, and software engineering. A YouTube link is provided for the project <https://www.youtube.com/watch?v=RyVyO8pSkhU>

# Introduction/Problem Description

Project 2 is a continuation of or rather a buildup of project 1 where denavit hartenberg (dh – parameters) and optimization algorithms were used to build a robotic system while a graphical user interface (GUI) models the behavior of it. This project aims to interface hardware and software to allow a robotic system to move physically based on the modeled GUI, additionally a game called Pictionary was developed to provide some intelligence to the robotic system to give a practical application of artificial intelligence. The game will allow two users to press a physical push button on an Arduino to create a guess of what the robot is drawing, given only 3 attempts the algorithm will check if each

player’s guess is correct or not, if so then the game is over and the player who guessed it wins, if neither

guessed correctly and 3 attempts are over the game is over with no winner.

Project 2 allows for practical experience using Arduino, wiring, breadboard, different servo motors, a push button, and resistors as the hardware component. The ability to work hands on with these hardware components gave the tools needed to build up a movable robotic system. Additionally, MATLAB programming will be used as the programming language environment due to its user-friendly graphical user interface and a programming technique called object-oriented programming (OOP) to keep code organized.

The challenges that this project aims to address is the ability for the robotic system to draw smoother shapes. Initially, without mathematics the robot will draw very jerky and discontinuous lines that make it hard to understand what shape is being created, which leads to players having a difficult time guessing what the robot is truly drawing.

# Numerical Solutions

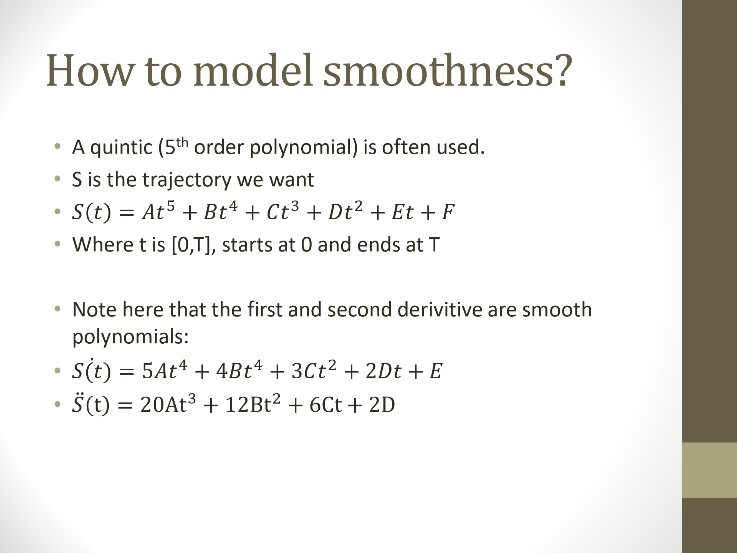
## TPOLY

Project 1 gave the mathematical concepts of transformation matrices, rotational matrix, forward kinematics, inverse kinematics, translations, and dh-parameters which will once again be applied to this project. The new mathematical concept that will be introduced is the idea of smoothness. In mathematics the idea of smoothness is when the first derivative (velocity), second derivative (acceleration), and third derivative (jerk) are continuous. Therefore, generally a polynomial function is used for smoothness. The field of robotics uses a 5th order polynomial referred to as quintic [1]. This project aims to use a MATLAB pre-defined function called TPOLY to allow the robot to draw smoother curves, therefore, a mathematical example will be used to understand what it is that TPOLY function is doing.

Suppose our mathematical model is given as

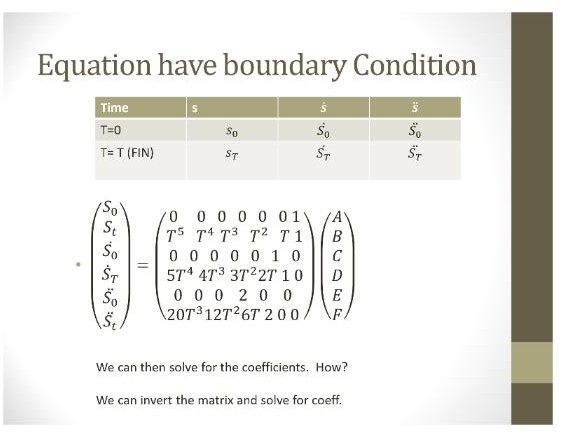
S(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + f

where the letters are all coefficients, and t starts at 0 and ends at T, these will be the boundary conditions of the function. Next, if the first and second derivative are taken, then smoothness occurs. Refer to **figure 1** for the derivation [1].

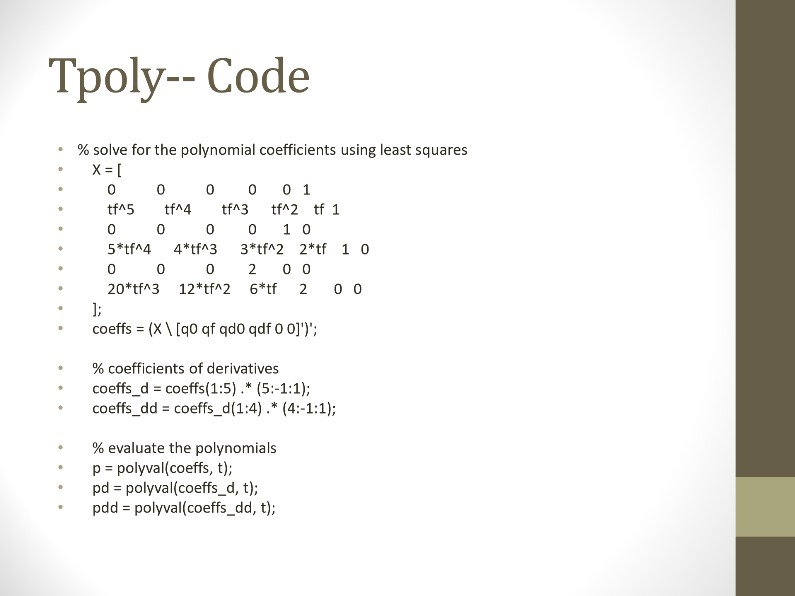


**Figure 1:** Shows a quintic polynomial that by taking its derivatives gives smoothness.

Since our model will have a start and an end boundary, we can set 3 conditions, start will be given a position, velocity, and acceleration, and end will have another 3 boundary conditions. The purpose of this is to individually set the robots starting velocity, position, acceleration, and at its ending as well. **Figure 2** shows how matrix multiplication can be utilized to solve for the coefficients of the system. One thing to note is that performing matrix multiplication via the rows of the 6X7 matrix and the coefficient matrix will yield our initial position, velocity, and acceleration. This mathematical process is what occurs in the MATLAB function TPOLY. TPOLY in our robotic system allows for the robot to draw smoother lines that may otherwise be discontinuous due to the hand measured joint angles. Refer to **figure 3** for the MATLAB function definition of the TPOLY [1]. This is how the problem of discontinuous unsmooth lines drawn by the robot will be solved.



**Figure 2:** Matrix multiplications to solve for coefficients taking the inverse.



**Figure 3:** function definition of TPOLY that utilizes the inverse matrix to solve for the coefficients of a quintic polynomial, thus allowing the robot to draw smoother lines. Note that position, velocity, and acceleration are returned.

# Methods

This section will explain both the hardware and software that were utilized in the robotic system. The same hardware that was utilized in project one will be carried over to this project with additional materials, as well as similar elements of the graphical user interface will be carried over.

## Hardware

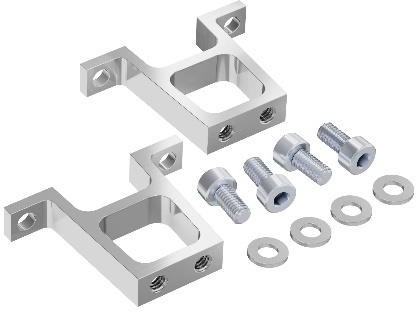
The following hardware materials were used

2 x HS-311 Standard Servo Motor 2 x Aluminum Beams 4.62” 1 x HS-81 Micro Servo Motor



2 10k ohm resistor Blue sticky tack 1 x clipboard

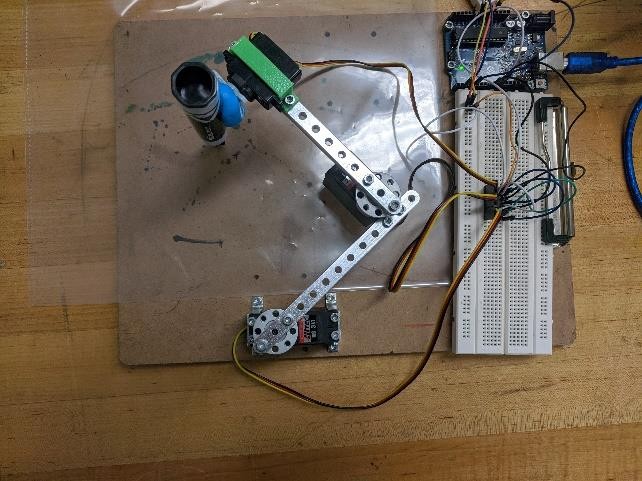


2 vertical mounts with 4 screws and 4 washers 1 bread board 2 Arduino push buttons



1 Arduino Uno R3.

Additionally, one fan was used as the end-effector. **Figure 4** provides a visualization of our robotic system.



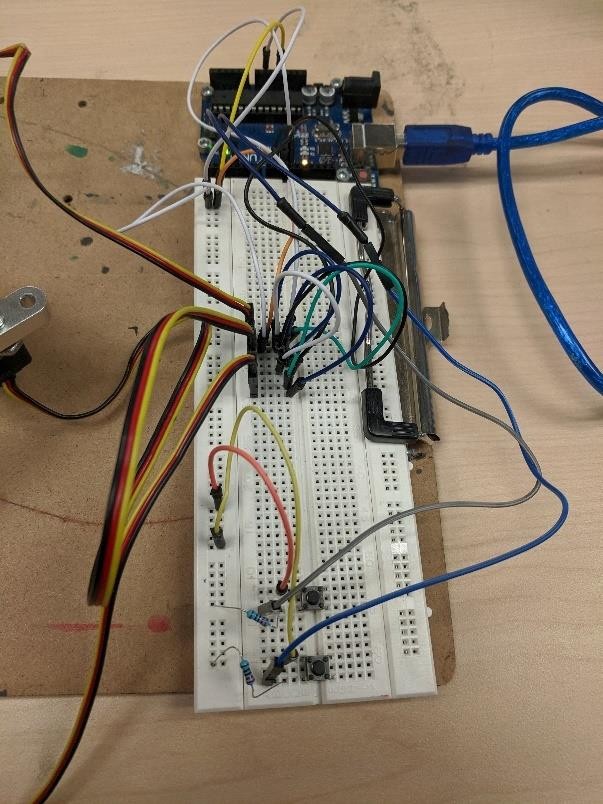
**Figure 4:** Visual representation of the robotic system developed

Before explaining how the hardware system was connected, knowledge of the physical Arduino must be understood to understand why each step was taken into consideration. The Arduino contains a place called the analog pins that will be utilized to connect ground and power, a reset button used to restart the board in case a glitch occurs, a USB interface used to connect the Arduino to the computer so MATLAB programming language can be utilized to control the robot, and digital input/output pins for the pulse width modulation to read the digital signals from the motors. These are the basics of an Arduino board and understanding these leads into the methods of connecting the hardware.

### Connecting hardware

1. Connect a wire to the analog 5V pin to the vertical portion of the breadboard and a wire from the ground analog pin to the other vertical portion of the breadboard. This provides 5V through all the vertical positions of the breadboard and the ground through the other vertical position of the breadboard.
2. Connect a wire from the vertical portion of breadboard that contains 5V running through, to the horizontal portion of the breadboard. There will be a horizontal power of 5V on the bread board now.
3. Connect a motors red, yellow, and black wire to the horizontal portion of the breadboard where the power (5V) is running horizontally but align it vertically. The red wire of the motor must be aligned with the horizontal 5V (perpendicularly). This is because the red wire in the motor corresponds to power, black is ground, and the yellow is the signal.
4. The other 3 sets of wires for the 2 motors are connected vertically one after another. This time another wire was added from the horizontal 5V to provide horizontal 5V to the red wire of the motor 2, while a 3rd wire was added to the second horizontal 5V to the 3rd motors red wire providing once again 5V moving horizontally.
5. Steps 2 through 4 will be repeated but this time using the vertical ground in step 2 and connecting it to the black wires of each motor which is the ground.
6. Attach two push buttons on the breadboard and running a wire from the vertical 5V and vertical ground to each corner of the push button
7. Attach a 10K ohm resistor to the each of the push button, this is so that when the button is not pushed the signal will be 0, pushing the button will deactivate the ground and output a 1 signal.
8. The final step is to connect a wire for each of the motors yellow wire (signal) to the Arduinos digital input, choose the ports that have ~ for inputting it as the pulse width modulation. Same process is applied to the two push buttons.

This concludes what hardware was used and the setup process, it will be a great service if one can understand the basic components of an Arduino and the different color schemes a servo motor uses (red for power, black for ground, yellow for signal). **Figure 5** provides a close-up visualization of the hardware connections. Additionally, understanding how to connect a push button and the purpose of using a resistor provides great knowledge of using the readdigitalpin() function.



## Software

**Figure 5:** close view of the hardware connection layout.

This section will explain the software that was utilized to control the robot’s movement, it will introduce classes known as object-oriented programming (OOP) as well as touch on the Arduino commands needed to move the end effector. Refer to the **appendix** for the code. Note that some coding from project 1 is used in this project, which include slider controls, reset robot, and key press function which will not be described in detail here (refer to project 1 ).

Object oriented programming is a way to keep code organized and provide something known as information hiding. For example, if there is a variable called social\_security and it grabs a user’s social security number, this information must be private that way it won’t be shared to other employees who use this program as well. Additionally, the use of OOP allows for all variables and functions to be defined and placed all together allowing one to debug and change their code efficiently. This project defined a class called Robot\_class that has properties known as variables (refer to **appendix** to view all variables). These properties are what will be manipulated throughout the class and utilized in the methods (functions) of the class. The methods of the class are where there will be some actions that will be carried out. For example, the constructor function is

function obj = Robot\_class()

one thing to note is that this is considered a default constructor meaning no parameters are being passed through and the purpose of this function is to initialize all the properties of the class.

Constructors are used because if a property is used without being initialized there will be some garbage data in it so initializing it to some value will not cause undesired results. Another important use of constructors is to check if there were any changes to a property, if there were changes to it then the default values would change. The last two methods in the Robot\_class are the functions

function movePosition(obj, joint,temp) writePosition(joint,temp)

and

function temp = getPosition(obj, joint) temp = readPosition(joint);

These are pre-defined functions that utilize user defined library functions from the servo motor. The movePosition essentially moves the robot physically by calling the pre-defined writePostion function which will move a specific joint with a specified value (temp), while the getPosition() will read the angle of the movePosition function. There is one thing to note and it is the use of the obj parameter in the function. Obj is considered an object of the class Robot\_class, which means that it has all the properties and methods of the class and the way to access them would be Obj.getPosition (obj, joint). However, in the case of using a class in the graphic user interface things become a little tricky. The graphic user interface uses something called a structure (struct for short) which is the same thing as a class but a struct by default allows access to all of its properties and methods publicly by default( public means any inherited class has access to them) while by default a class will have them private meaning only its class can directly access them. The struct in the GUI is called handles and what is happening is the Robot\_class is a class within the structure handles, so different syntax is needed to access the properties and methods of Robot\_class within the handles struct which will be explained below.

Inside the opening function of the GUI there is the following code

handles.rc = Robot\_class();

what this handles.rc does is create a variable of the handles structure that holds an object of the Robot\_class, this means that all the Robot\_class properties and methods are held inside the handles.rc and can be manipulated. The syntax to access the Robot\_class methods follows

handles.rc.theta(1) = get(hObject,'Value');

this is similar to the above dot notation, but an extra layer was added the struct, then the Robot\_class, so dot indexing must be used twice since it is a class within a struct. Now that the idea of OOP is understood, explaining what the GUI functions do will be much clearer.

The 3 slider functions will behave similarly to how they did in project 1 but the addition of the movePosition function. The slider controls will allow the physical robot to move by taking in the s variables (explained in the create robot function) and utilizing the values of the slider. Note that writePosition can only take values between 0 (smallest angle) and 1 (largest angle). Therefore, a method from the Robot\_class was created called map\_fn, which normalizes values between the range of 0 and

1. This is an important concept to understand if one decides to learn machine learning because it will be used extensively. Now moving the slider in the GUI will move the robot incrementally.

The Key\_press\_Callback function was modified slightly to allow for writing a text file that will hold the joint angles of the shapes. What was added was the

fprintf(handles.fileID,'%3.2f %3.2f %3.2f

%3.2f\n',handles.rc.getPosition(handles.rc.s1),handles.rc.getPosition(handles.rc.s2), handles.rc.theta(1), handles.rc.theta(2)); % will print it to the text file

This function will print out to a struct variable called handles.fileID which is the location of a text file, what will be printed are the three parameters. Note that when an error key is pushed the movePosition is called which will move the robot and the readPosition function will read the angle moved which will then be stored in the handles.fileID text file. A prerequisite for this function to work properly is the File\_Open\_Callback function which will be called in the push button of the GUI (Open file) to open the file. An error will occur if the keypress function is called before opening the file. Similarly, the close file call back will close the file when done writing out it.

The Create\_Robot\_Callback will be utilized the same way as project 1 with some modifications as well. Here the new code will be explained and reference to the older code can be found in project 1.

handles.rc.a = arduino('COM3', 'Uno', 'Libraries', 'Servo');

This function utilizes the property of Robot\_Class called a, what is happening is an object a is created from the Arduino class that uses the Servo libraries. Now a holds the properties of the Arduino and can be passed to other functions retaining these properties, this leads to the usage of the function

handles.rc.s1 = servo(handles.rc.a, 'D9', 'MinPulseDuration', 575\*10^-6, 'MaxPulseDuration', 2460\*10^- 6);

which will use the Arduinos properties a, to create a connection (signal from the motor) between the motor 1 (called s1) to the Arduino. Note the min pulse duration and max pulse duration was obtained through HS-311 website tech specification, it is important that this information be updated to prevent breaking the motor. Now that the s1 is initialized for each motor (s1, s2, s3) it can be used in the movePosition function to cause the physical robot to move.

The final function that was created from the ground up that was not used in project 1 is the Play\_Game\_Callback function. The purpose of this function is to allow for the robot to draw some shape from a text file (the joint angle files stored in the text file in key press function) and allow user to physically push a button on the Arduino to make a guess (which will stop the robot from drawing), if the player is correct the robot stops drawing and the player wins, if the player is wrong then they lose a life (each player only has 3 lives). The coding of this is explained below.

The first step is to load a text file of a particular shape, this is randomly loaded by using randi function which will return a value 1 or 2 (2 is the maximum number of shapes), followed by an if statement which will load a file based on what number is chosen. Next step is to use the class properties handles.rc.x, handles.rc.y, etc to hold each column of the text files, note each column of the text file holds a motors joint angle. The first 2 columns of the text file will hold the normalized values (0 – 1) and the 3rd and 4th hold the unmapped values. Once the file is loaded and the variables are initialized to the joint angles a nested for loop is created. The outer for loop will iterate through each of the joint angles (note the use of the length function which returns the number of values in the vector) and use the TPOLY function which was explained in the mathematical section. A second for loop is needed because TPOLY returns a vector of position m steps between the handles.rc.x(i) (current joint angle) and the next joint angles to come (handles.rc.x(i + 1)). Since m values are returned the robot must move m steps so a

nested for loop was needed to iterate to these values, by doing this it will allow smoother lines between the current joint angle and the one to come.

The if statement in the inner for loop is created to allow users to guess what shape the robot is drawing. Therefore, originally the push button returns a signal value of 0, but if a user presses the button the signal returns a 1, which is exactly what was coded.

if readDigitalPin(handles.rc.a,'D2') == 1 || readDigitalPin(handles.rc.a,'D4') == 1

Now if this button was pressed (value becomes 1) that means that player 1 or player 2 wants to make a guess of what the robot is drawing. Once the player makes a guess the string of it is loaded to the following variable

Player1\_guess = inputdlg(prompt,dlgtitle);

Note that the data type stored is of a cell type so it must be converted to a string type with the following code

Player1\_guess\_str = char(Player1\_guess);

Once the guess is done it must be checked if the player guessed what shape the robot is drawing with the following if statement

if strcmp(Player1\_guess\_str,answer\_file)

note that in MATLAB The only way to compare two strings is to call the strcmp() function. Now if the player is correct then that player wins, however, if they are not correct the else statement is carried out and the player will loose a life.

player1 = player1 - 1;

The same coding is utilized for player 2.

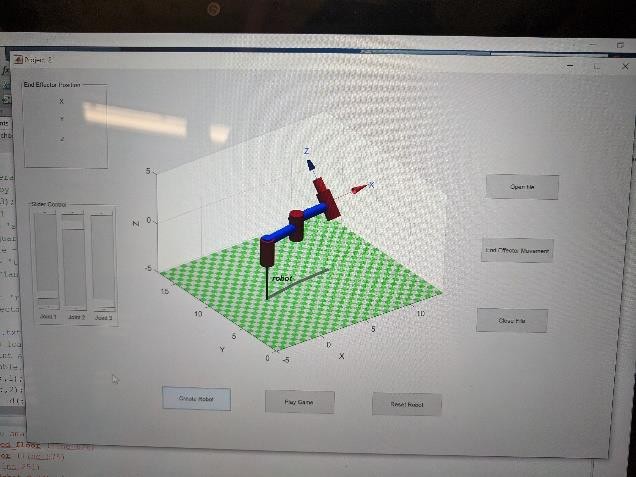
Towards the end of the outer for loop the handles.rc.robot.plot is called to plot the GUI robot exactly as the hardware robot is moved, but forward kinematics is used to draw the shape on the GUI.

t = handles.rc.robot.fkine ([-handles.rc.x\_pos(i), -handles.rc.y\_pos(i), z(i)]);

hold on; plot3(t.t(1), t.t(2), t.t(3), '--r.');

As mentioned in project 1, forward kinematics will take the joint angles and give the transformation matrix, in which the x, y, z positions can be extracted and utilized to plot end effector movement, adding ‘—r’ will plot a red line and the hold on command will allow the previous red line mark to stay with each iteration. **Figure 6** shows a visual look of the GUI.

This concludes the software utilized in project 2, it showed an introduction to object-oriented programming, controlling an Arduino linked to a servo motor, and a practical approach of using TPOLY.

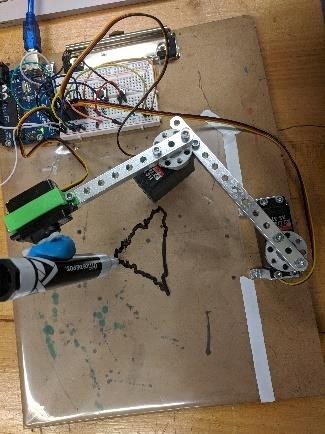
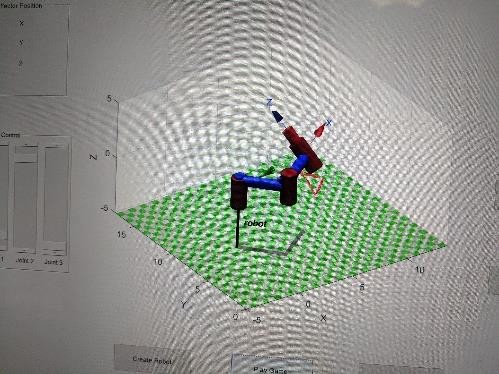


**Figure 6:** Shows a visual look at the graphic user interface (GUI).

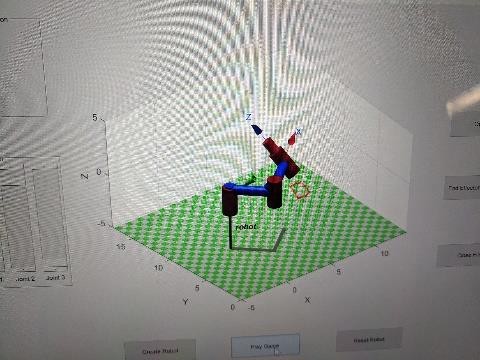
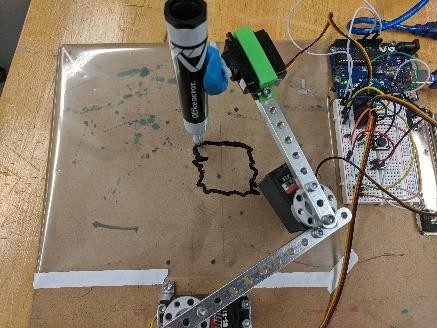
# Results

Project 2 was a great exercise interfacing both hardware and software, it is a great hands on approach in working in any technical field, whether one works strictly in software or hardware, learning about the other will provide great insight on the intricates of what is going on in the background. For example, working in robotics field both coding and hardware are needed, working in machine learning which is strictly software can be understood more if one understands the hardware components of a computer (compliers functioning based on the different gates). This project used object-oriented programming to keep code nice and structured that allowed users to debug efficiently. Furthermore, an Arduino was used to create a connection between each servo motor and the MATLAB programming environment to allow for movement of the physical robot, this impart was done by the servo library to utilize commands such as writePosition and readPosition. Once coding and hardware was implemented correctly the robot was able to draw based on joint angles stored in a text file, however, the issue arose in the shapes being created were jerky and not smooth, this bridged into using artificial intelligence. The use of TPOLY to smooth out the robots drawing gave a little preview of how intelligence can be provided in robotics. The accuracy of the robot drawing a shape with TPOLY improved but not by much, this may have been due to the pencil not being attached securely to the end effector. The pencil and the paper on the clip board provided a great deal of friction and not enough force to move the robot resulting in the pencil failing to draw. A great solution of this was to use laminated paper (used in overhead projectors) and a marker to create a less friction surface to draw on. Additionally, the use of Arduino push buttons allowed for players to guess what the robot is drawing to create the Pictionary game.

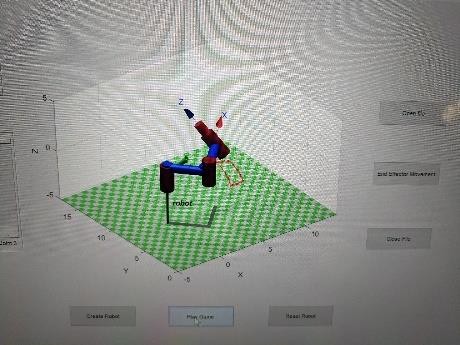
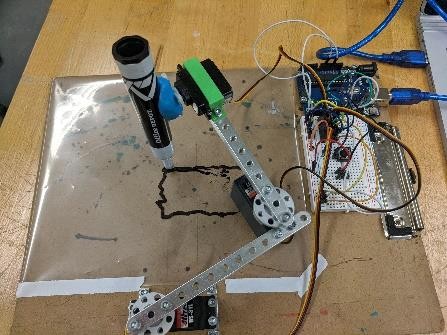
Issues did arise with the Arduino push buttons with the signal being read erratically, the solution to this was using a new breadboard due to the old one being fried. The following figures below show the drawn shapes on the hardware robot as well as the GUI.

**Figure 7:** Hardware Triangle drawn **Figure 8:** GUI Triangle



**Figure 9:** Hardware square drawn **Figure 10:** GUI Square



**Figure 11:** Hardware Rectangle draw **Figure 12:** GUI Rectangle drawing

# Discussion/Conclusion

Looking forward, the techniques of this project can be utilized in many domains of science and engineering. For example, the information learned in project 2 can be applied to robotic surgery, which will allow doctors to have the tools needed to provided precision based surgery, one issue that many robotic systems may have is moving in jerky and non-smooth motion, this can be disastrous in a surgical environment. The technique utilized in this project known as TPOLY uses a quintic polynomial to allow the robot to move more smoothly. Additionally, project 2 used object-oriented programming which is used in all phases of computer science especially software engineering. OOP can be used in all domains of STEM because it provides a way to hide information and organize code, the ability to organize code to allow for better debugging does a great service to any programmer in terms of saving time to focus on the research project in hand. Therefore, the techniques and concepts utilized in this project will be lifelong skills in whatever domain an individual chooses to work in.

# Appendix

## GUI Code

function varargout = Project\_2(varargin)

% PROJECT\_2 MATLAB code for Project\_2.fig

% PROJECT\_2, by itself, creates a new PROJECT\_2 or raises the existing

% singleton\*.

%

% H = PROJECT\_2 returns the handle to a new PROJECT\_2 or the handle to

% the existing singleton\*.

%

% PROJECT\_2('CALLBACK',hObject,eventData,handles,...) calls the local

% function named CALLBACK in PROJECT\_2.M with the given input arguments.

%

% PROJECT\_2('Property','Value',...) creates a new PROJECT\_2 or raises the

% existing singleton\*. Starting from the left, property value pairs are

% applied to the GUI before Project\_2\_OpeningFcn gets called. An

% unrecognized property name or invalid value makes property application

% stop. All inputs are passed to Project\_2\_OpeningFcn via varargin.

%

% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one

% instance to run (singleton)".

%

% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Project\_2

% Last Modified by GUIDE v2.5 12-Nov-2019 14:48:51

% Begin initialization code - DO NOT EDIT gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ... 'gui\_OpeningFcn', @Project\_2\_OpeningFcn, ... 'gui\_OutputFcn', @Project\_2\_OutputFcn, ... 'gui\_LayoutFcn', [] , ...

'gui\_Callback', []); if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:}); else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code - DO NOT EDIT

% --- Executes just before Project\_2 is made visible.

function Project\_2\_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to Project\_2 (see VARARGIN)

% Choose default command line output for Project\_2 handles.output = hObject;

% creates an object of the robot class and stores it in handles.rc

%struct property. handles.rc = Robot\_class();

clc

% This code is needed to make sure our variables retain there values

% in other functions guidata(hObject, handles);

% --- Outputs from this function are returned to the command line. function varargout = Project\_2\_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure varargout{1} = handles.output;

% --- Executes on slider movement.

function Joint\_1\_Callback(hObject, eventdata, handles)

% This code will grab the value of our slider position and store it to our

% structs variable theta(1) handles.rc.theta(1) = get(hObject,'Value');

y = handles.rc.map\_fn(handles.rc.theta(1),-0.261799388,3.18522588); handles.rc.movePosition(handles.rc.s1,y)

% with the new theta value we will plot our robot with the new angle

handles.rc.robot.plot(-handles.rc.theta);

%joint1\_position = readPosition(handles.s1)

% The code below will be used to update end effector x position in

% real time as we move the slider

T = handles.rc.robot.fkine(-handles.rc.theta); X = sprintf('X = %f', T.t(1)); set(handles.x\_func,'String',X);

drawnow();

% Update handles structure guidata(hObject, handles);

% --- Executes during object creation, after setting all properties. function Joint\_1\_CreateFcn(hObject, eventdata, handles)

% hObject handle to Joint\_1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background. if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Executes on slider movement.

function Joint\_2\_Callback(hObject, eventdata, handles)

% This code will grab the value of our slider position and store it to our

% structs variable theta(2) handles.rc.theta(2) = get(hObject,'Value');

y = handles.rc.map\_fn(handles.rc.theta(2),-3.18522588,0.261799388); handles.rc.movePosition(handles.rc.s2,y)

% with the new theta value we will plot our robot with the new angle handles.rc.robot.plot(-handles.rc.theta);

% The code below will be used to update end effector x position in

% real time as we move the slider

T = handles.rc.robot.fkine(handles.rc.theta); Y = sprintf('Y = %f', T.t(2)); set(handles.y\_func,'String',Y);

drawnow();

% Update handles structure guidata(hObject, handles);

% --- Executes during object creation, after setting all properties. function Joint\_2\_CreateFcn(hObject, eventdata, handles)

% hObject handle to Joint\_2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background. if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Executes on slider movement.

function End\_Effector\_Callback(hObject, eventdata, handles)

% This code will grab the value of our slider position and store it to our

% struct variable theta(3) handles.rc.theta(3) = get(hObject,'Value');

handles.rc.movePosition(handles.rc.s3,handles.rc.theta(3))

% with the new theta value we will plot our robot with the new angle handles.rc.robot.plot(handles.rc.theta);

% The code below will be used to update end effector x position in

% real time as we move the slider

T = handles.rc.robot.fkine(handles.rc.theta); Z = sprintf('Z = %f', T.t(3)); set(handles.z\_func,'String',Z);

drawnow();

% Update handles structure guidata(hObject, handles);

% --- Executes during object creation, after setting all properties. function End\_Effector\_CreateFcn(hObject, eventdata, handles)

% hObject handle to End\_Effector (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background. if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Executes on button press in Create\_Robot.

function Create\_Robot\_Callback(hObject, eventdata, handles)

% Specify the axes to use axes(handles.axes1);

% value of our work space to change the floor titles W = [-5 13.5 0 17.5 -5 5];

% deg will be used to convert degrees to radians deg = pi/180;

% By inputting our dh parameters as an argument we will generate each link

L(1) = Link('a', 3.8490,'alpha', 0.0030, 'd', 1.0303, 'offset', 0.1864, 'qlim', [15 -182.5]\*deg);

L(2) = Link('a', 3.7936, 'alpha', -0.6883, 'd', 0.7163, 'offset', - 0.1017, 'qlim', [182.5 -15]\*deg);

L(3) = Link('a', 0.0016 , 'alpha', -0.0008, 'd', 0.2595, 'offset', 0.0087, 'qlim', [0, pi]);

% serial link will be used to help create the robot handles.rc.robot = SerialLink(L, 'name', 'robot');

% robotic base that is 8 inches in y direction from universial coordinate

% to robotic base, similarly for the tool tip handles.rc.robot.base = [1 0 0 0; 0 1 0 8; 0 0 1 0; 0 0 0 1];

handles.rc.robot.tool = [1 0 0 0; 0 1 0 0;0 0 1 -2.5; 0 0 0 1];

% below code will plot our robot with the theta values and update

% the floor of our robot with the 'workspace' command handles.rc.robot.plot (handles.rc.theta, 'workspace', W);

% create an arduino object and store it to robot class property a handles.rc.a = arduino('COM3', 'Uno', 'Libraries', 'Servo');

% this code is used to make the pin an input opposed to an output handles.rc.a.configurePin('D2','DigitalInput'); handles.rc.a.configurePin('D4','DigitalInput');

% create a servo object for each of the motors and inputting its pulse

% duration range from tech specs from website

handles.rc.s1 = servo(handles.rc.a, 'D9', 'MinPulseDuration', 575\*10^- 6, 'MaxPulseDuration', 2460\*10^-6);

handles.rc.s2 = servo(handles.rc.a, 'D10', 'MinPulseDuration', 575\*10^-6, 'MaxPulseDuration', 2460\*10^-6);

handles.rc.s3 = servo(handles.rc.a, 'D11', 'MinPulseDuration', 640\*10^-6, 'MaxPulseDuration', 2250\*10^-6);

% Update handles structure guidata(hObject, handles);

% --- Executes on button press in Reset\_fn.

function Reset\_fn\_Callback(hObject, eventdata, handles)

% reset the robot back to starting postion by making theta values 0 handles.rc.robot.plot ([0 0 0]);

% choose the physical starting postion of our hardware robot handles.rc.movePosition(handles.rc.s1,0.1) handles.rc.movePosition(handles.rc.s2,0.9)

% Update handles structure

guidata(hObject, handles);

% --- Executes on button press in Key\_press.

function Key\_press\_Callback(hObject, eventdata, handles)

% ignore this function

% --- Executes on key press with focus on Key\_press and none of its controls.

function Key\_press\_KeyPressFcn(hObject, eventdata, handles)

% hObject handle to Key\_press (see GCBO)

% eventdata structure with the following fields (see MATLAB.UI.CONTROL.UICONTROL)

% Key: name of the key that was pressed, in lower case

% Character: character interpretation of the key(s) that was pressed

% Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed

% handles structure with handles and user data (see GUIDATA)

% this code will get the current graphic figure f= gcf;

% m matrix specfies the axis (x,y,and z) and if there are any yaw, pitch, and row

m = [1 1 0 0 0 0];

% val will get the current arrow key from the keyboard, note computer can

% only recongnize numbers so the ascii table assigns a number to each

% keyboard key and this number is what is stored in the val variable val=double(get(f,'CurrentCharacter'));

%do a forward kinematics

T = handles.rc.robot.fkine([handles.rc.theta(1), handles.rc.theta(2), handles.rc.theta(3)]);

% depending on the val value it will run the switch case based on it. switch (val)

case 28 % right

T.t(1) = T.t(1) + .1;

case 29 %left

T.t(1) = T.t(1) - .1;

case 30 %up

end

T.t(2) = T.t(2) + .1;

case 31 %down

T.t(2) = T.t(2) - .1;

otherwise

disp ('unrecognized key');

% inverse kinematics is used here to get the joint angles which are theta 1

% theta 2 and theta 3.

q = handles.rc.robot.ikine(T,'q0', handles.rc.theta,'mask', m);

% if q is not empty meaning there is some value in it then we will plot the

% robot and store the new joint angles in our theta values. This is good to

% prevent self assignment and decrease the time complexity of our program.

if ~isempty(q)

handles.rc.robot.plot([-q(1) -q(2) q(3)]); handles.rc.theta(1) = q(1); handles.rc.theta(2) = q(2);

handles.rc.theta(3) = q(3); else

disp ('no solution');

end

% these will cal the Robot\_class mapping function which will make each of

% our theta values a range between 0 - 1 because the movePosition function

% can only take values between 0 and 1.

y1 = handles.rc.map\_fn(handles.rc.theta(1),-0.261799388,3.18522588); y2 = handles.rc.map\_fn(handles.rc.theta(2),-3.18522588,0.261799388);

% move position is a Robot\_class function that calls a servo pre- defined

% function writePosition to move the specfied motor based on s used and

% given a theta value that is mapped between 0 and 1. handles.rc.movePosition(handles.rc.s1,y1) handles.rc.movePosition(handles.rc.s2,y2)

% This code will print out to a text file the pre-condition is that the

% open file function must be called to open the actual file. fprintf(handles.fileID,'%3.2f %3.2f %3.2f

%3.2f\n',handles.rc.getPosition(handles.rc.s1),handles.rc.getPosition( handles.rc.s2), handles.rc.theta(1), handles.rc.theta(2)); % will print it to the text file

% Update handles structure guidata(hObject, handles);

% --- Executes on button press in File\_Open.

function File\_Open\_Callback(hObject, eventdata, handles)

% opens a file

handles.fileID = fopen('joint\_angles.txt','a');

% if the file is not empty then display message if ~isempty(handles.fileID)

msgbox('File is Open!') else

msgbox('File is not open') % file is empty display message

end

% Update handles structure guidata(hObject, handles);

% --- Executes on button press in File\_Close.

function File\_Close\_Callback(hObject, eventdata, handles)

% closes a file fclose(handles.fileID); msgbox('File is closed!')

% Update handles structure guidata(hObject, handles);

% --- Executes on button press in Play\_Game.

function Play\_Game\_Callback(hObject, eventdata, handles)

% code is a buffer to prevent error readings of our digtail pin signal for i = 0:10

readDigitalPin(handles.rc.a,'D2') readDigitalPin(handles.rc.a,'D4')

end

% graph work space

W = [-5 13.5 0 17.5 -5 5];

% each player is set a max life of 3. player1= 3;

player2 = 3;

% this code will generate a random number up to 2 and call a file based on

% the vale returned by the if statement. random\_file = randi(3);

% if random\_file ==1

% answer\_file = 'square.txt';

% ld = load('square.txt');

% elseif random\_file ==2

% answer\_file = 'triangle.txt';

% ld = load('Triangle.txt');

% else

% answer\_file = 'rectangle.txt';

% ld = load('Rectangle.txt');

% end

ld = load('triangle.txt');

% once the file is loaded with the joint angles we set each of the columns

% which is the joint angles per motor and the unmapped values to each

% respective variable. handles.rc.x = ld(:,1);

handles.rc.y = ld(:,2); handles.rc.x\_pos = ld(:,3); handles.rc.y\_pos = ld(:,4);

% need to have z vector of values up to our joint columns. z = zeros(1,length(handles.rc.x)-1);

% outer for loop to iterate through each joint angle for i = 1:length(handles.rc.x) -1

%P1= spline(X,Y,XX)

%P2 = spline(X,Y,XX)

% this code is used so tpoly can draw out smoother lines between

the

% current angle and the angle to come by iterating through each

step.

[P1,SD1,SDD1] = tpoly(handles.rc.x(i), handles.rc.x(i+1),10); [P2,SD2,SDD2] = tpoly(handles.rc.y(i), handles.rc.y(i+1),10); for j = 1:length(P1)

% the code below the if statement is to allow for each player

to

% guess, based on which pin is pressed that player will given

a

% choice to guess

if ((readDigitalPin(handles.rc.a,'D2')) ||

(readDigitalPin(handles.rc.a,'D4')))

if readDigitalPin(handles.rc.a,'D2') == 1 prompt = 'Enter your guess(add .txt)'; dlgtitle = 'Player 1 Guess'; Player1\_guess = inputdlg(prompt,dlgtitle); Player1\_guess\_str = char(Player1\_guess);

% compare the players guess with the file loaded use

string display

% compare function to do this if they are right

% message and say they win.

if strcmp(Player1\_guess\_str,answer\_file) msgbox('Player 1 wins')

pause(2)

return % code to end everything else

% Player is wrong so take away a life player1 = player1 - 1;

disp('Player 2 lost a life')

end

end

% player 2 pressed button so below else if runs elseif readDigitalPin(handles.rc.a,'D4') == 1

prompt = 'Enter your guess(add .txt)'; dlgtitle = 'Player 2 Guess'; Player2\_guess = inputdlg(prompt,dlgtitle); Player2\_guess\_str = char(Player2\_guess);

% if player 2 guess rights they win.

if strcmp(Player2\_guess\_str,answer\_file) msgbox('Player 2 wins')

pause(2)

return % code to end everything

end

else

end

% player 2 guessed wrong so they loose a life. player2 = player2 - 1;

disp('Player 2 lost a life')

% This code is to check if either player's lives are over then

no

% one wins the game.

if player1 == 0 || player2 == 0 msgbox('Game over no one wins') pause(2)

return % code to stop the robot

step

end

end

% This code will actually move the hardware robot with each m

% in tpoly. handles.rc.movePosition(handles.rc.s1,P1(j)) handles.rc.movePosition(handles.rc.s2,P2(j))

% This code will plot the GUI robot based of each theta values unmapped.

handles.rc.robot.plot([-handles.rc.x\_pos(i),- handles.rc.y\_pos(i),z(i)], 'workspace', W);

t = handles.rc.robot.fkine ([-handles.rc.x\_pos(i), - handles.rc.y\_pos(i), z(i)]);

hold on;

plot3(t.t(1), t.t(2), t.t(3), '--r.');

end

guidata(hObject, handles);

## Robot\_class code

classdef Robot\_class

properties

% These are all the variables of the robot class, this is

what

end

% will be used in the handles structure of the GUI to

% manipulate data around with the functions of this class. robot;

theta; x;

y;

s1;

s2;

s3;

a;

b; x\_pos; y\_pos;

is are

values to

methods

function obj = Robot\_class()

% This function is called the default constructor it

% used to intialize all of the properties of the

% Robot\_class, the purpose of this is that if values

% not intialized then there will be some arbitrary

% stored in each variable which can cause the program

% misbehave. obj.x = [];

obj.y = []; obj.s1 = 1;

obj.s2 = 1;

obj.s3 = 1; obj.a = [];

obj.b = []; obj.theta = [0 0 0]; obj.robot = [];

obj.x\_pos = [];

obj.y\_pos = [];

end

with as an

function y = map\_fn(obj, theta, min, max)

% This function is to standardize the theta values

% values between 0 and 1 so movePosition can use them

end

% argument in there parameters. y = (theta - min)./(max - min);

as a

hardware

function movePosition(obj, joint,temp)

% The function will move the robot based on which joint

% selected and which value (temp) used. the obj is treated

% pointer to the object of Robot\_class writePosition(joint,temp)

end

function temp = getPosition(obj, joint)

%This function will get the actual values of the

% movement of the robot. temp = readPosition(joint);

end

end

end

# Reference

* 1. Dr. Abhilash Pandya, “Time and Motion”, Introduction to Robotic Systems I. 12 November 2019.