Project Deliverable G:
Prototype II and Customer Feedback

Ani Preedom
Christy Lau
Paul Shedden
Claire Durand

GNG1103 Project Group 6
March 13, 2022

Abstract

On Halifax-class frigates, the Department of National Defence has a need for a robotic arm that uses inverse kinematics to paint surfaces. The robot must also scan and clean areas to identify and remove defects. To design the robot, a design process with several steps will be followed. Thus far, conceptual designs have been generated using design criteria based on raw data gathered from one of the users. At this stage in the design process, a second prototype has been created and further test plans must be made based on client feedback to ensure that the project can be completed in the allotted time. This report presents detailed prototyping progress, an updated prototype test plan and bill of materials (BOM).

Table of Contents

1.0 Introduction

The Department of National Defence expressed the need for a robotic arm with three degrees of freedom to paint areas on Halifax-class frigates. A second prototype testing plan was created based on the conceptual designs. Using this second test plan, prototypes were created and their progress was recorded. Client feedback was analysed and a third prototype test plan was built off of the original.

2.0 Detailed Design Drawing and Client Feedback

Prototype designs for each subsystem were created in deliverable F. These prototypes were shown to the client and no feedback was given. Physical and more detailed prototypes were then created based off of previous designs. Figure 1 shows a detailed design diagram of the three stepper motors for controlling the degrees of freedom of the robotic arm, the ultrasonic sensor to detect possible individuals or other objects in the area, and the kill switch to turn off the arm in case of an emergency.

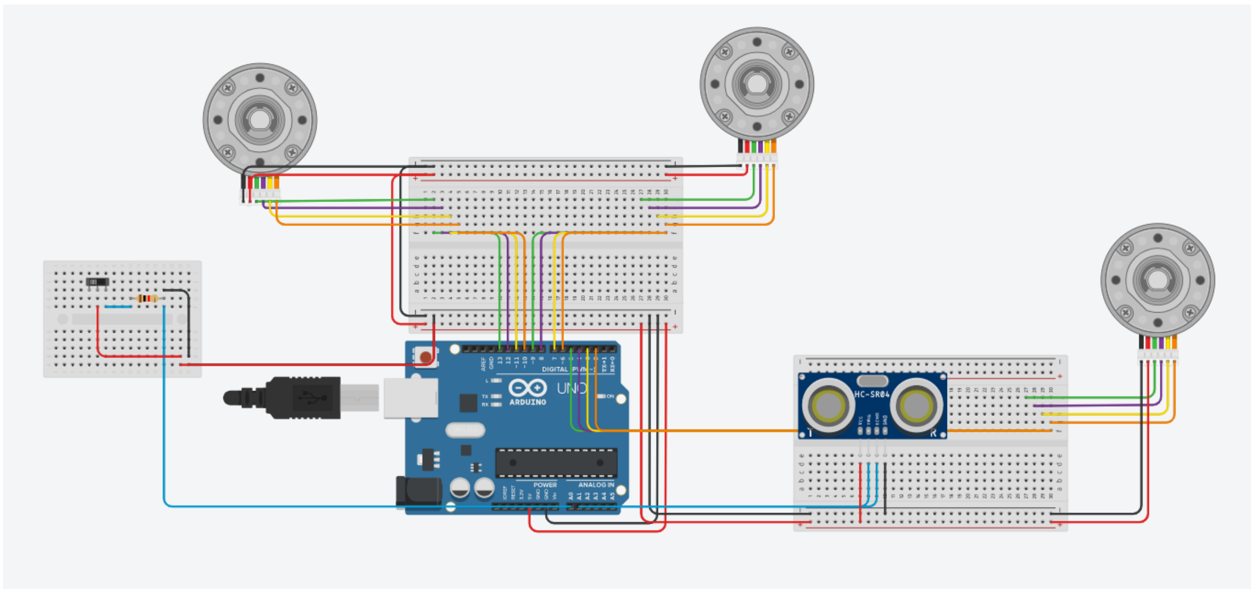


Figure 1. Detailed design diagram of Circuit for Motors and Sensors.

Figure 2 shows the full arm, with the attached 3D printed end effector prototype holding a pencil.

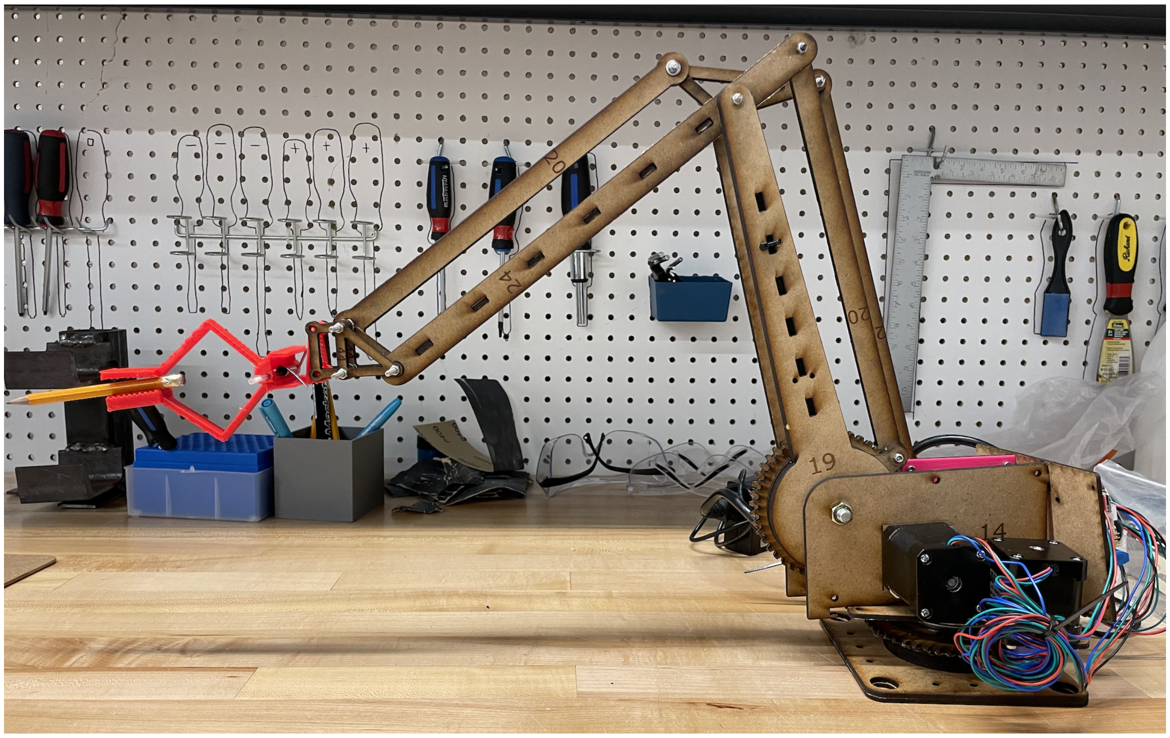


Figure 2. Robotic arm with a prototype end effector.

Figure 3 shows a close up of the full end effector prototype holding a pencil, including the spring mechanics.

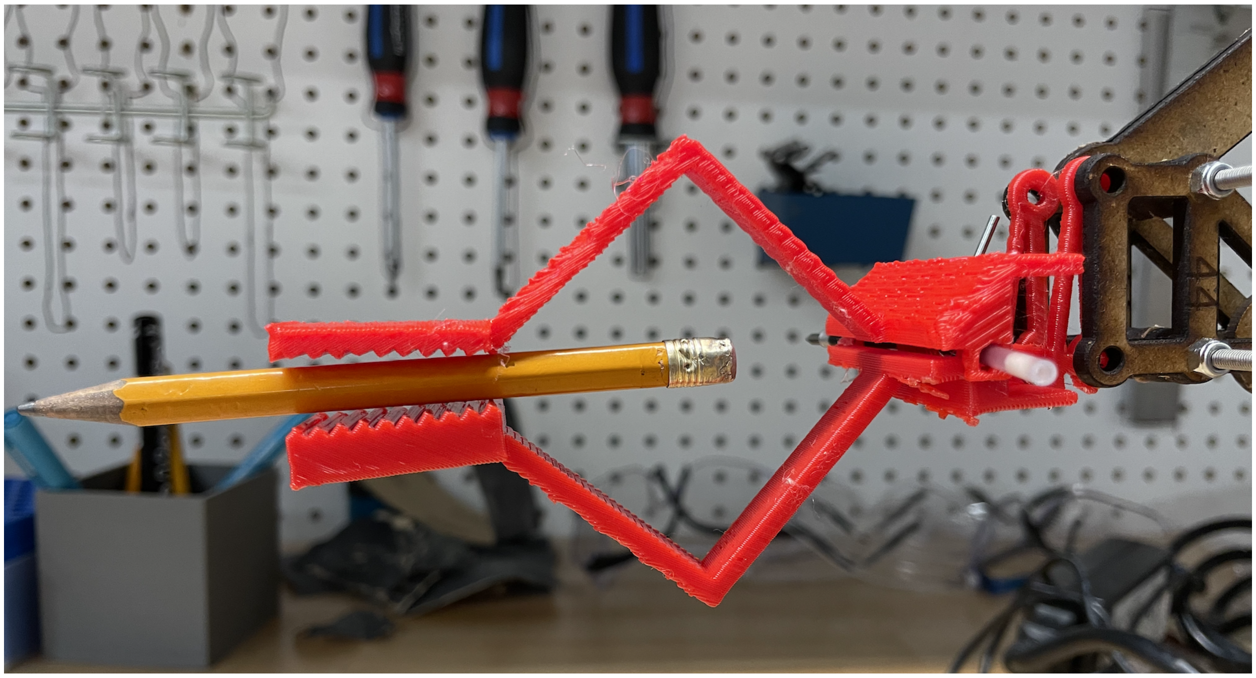
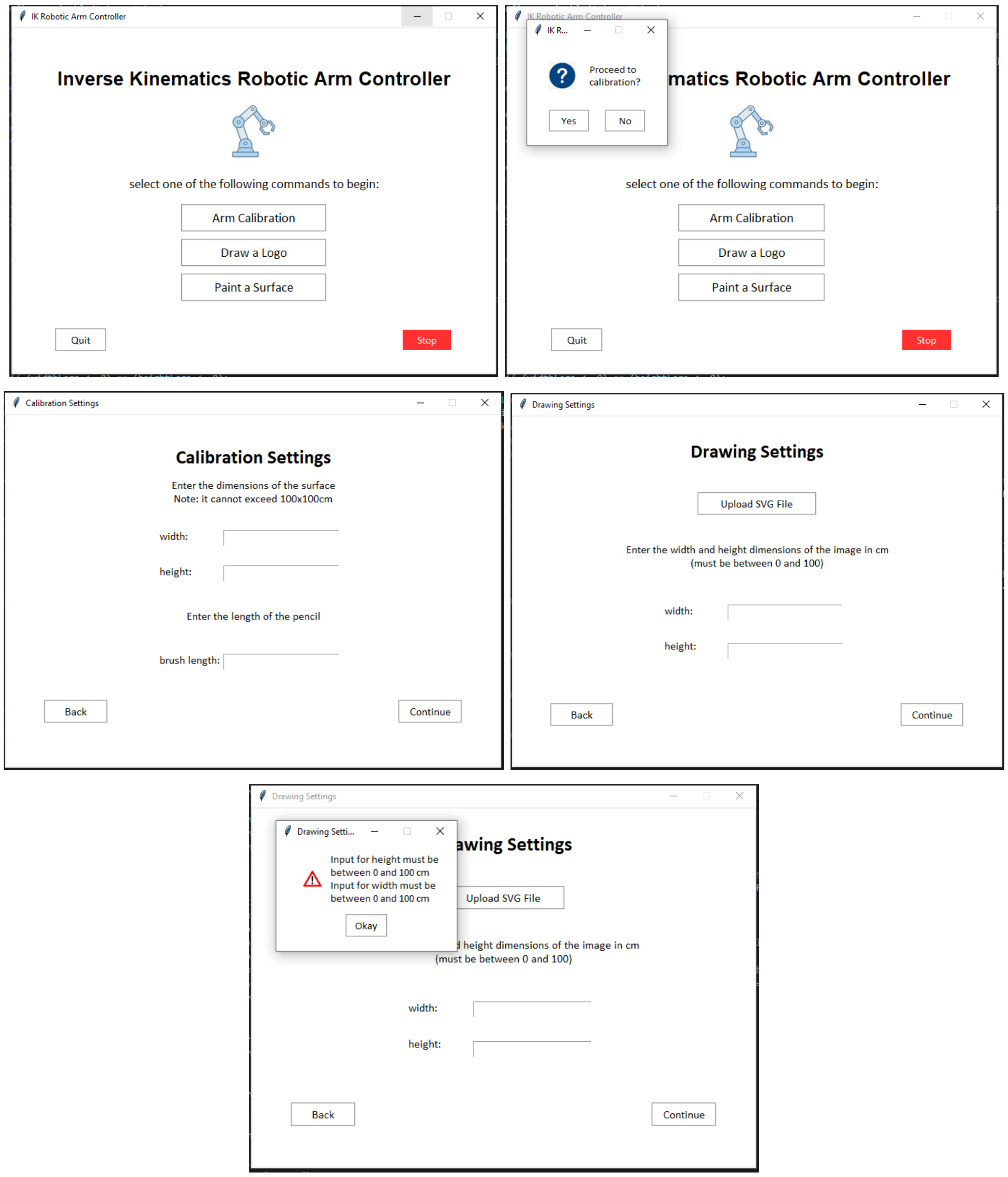


Figure 3. Close up of the end effector



Inverse Kinematics Robotic Arm Controller

select one of the following commands to begin:

Proceed to calibration?

Calibration Settings

Enter the dimensions of the surface
Note: it cannot exceed 100x100cm

Enter the length of the pencil

Drawing Settings

Enter the width and height dimensions of the image in cm
(must be between 0 and 100)

Input for height must be between 0 and 100 cm
Input for width must be between 0 and 100 cm

Figure 4. Detailed Images of User Interface

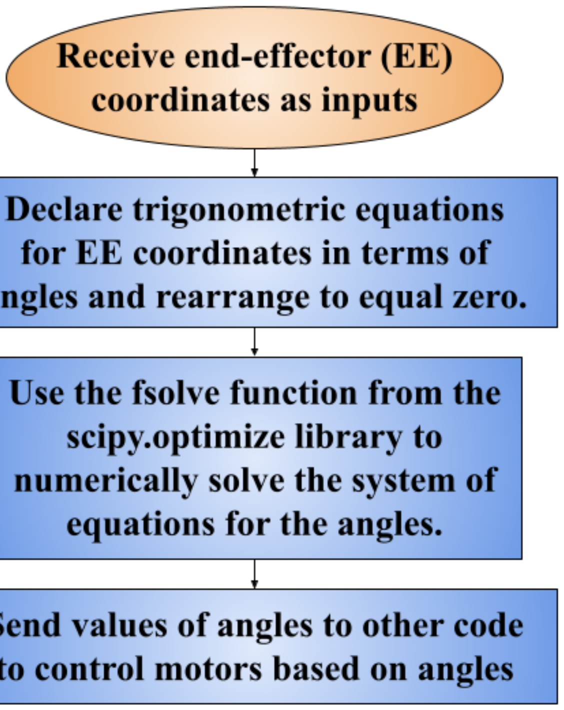

Receive end-effector (EE) coordinates as inputs
Declare trigonometric equations for EE coordinates in terms of angles and rearrange to equal zero.
Use the fsolve function from the scipy.optimize library to numerically solve the system of equations for the angles.
Send values of angles to other code to control motors based on angles

Figure 5. Flowchart of code for the inverse kinematics calculations in Python.

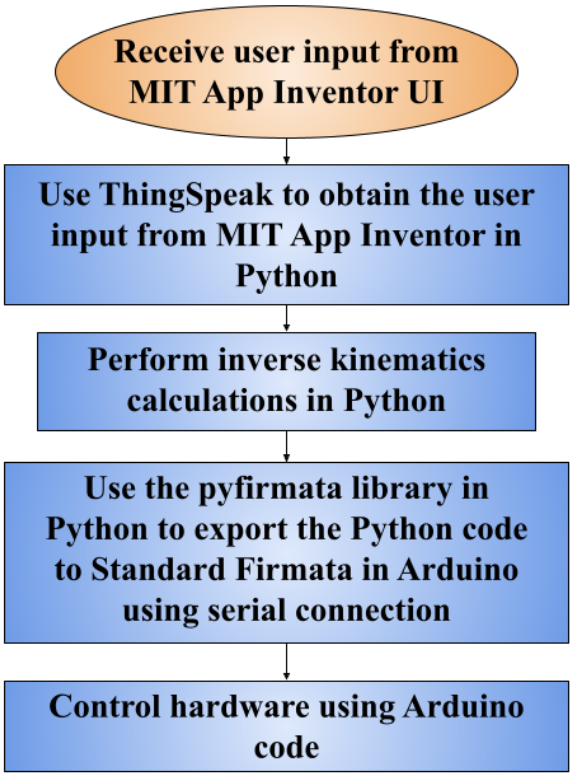

Receive user input from MIT App Inventor UI
Use ThingSpeak to obtain the user input from MIT App Inventor in Python
Perform inverse kinematics calculations in Python
Use the pyfirmata library in Python to export the Python code to Standard Firmata in Arduino using serial connection
Control hardware using Arduino code

Figure 6. Flowchart of overall software subsystem from the user interface to the Arduino.

3.0 First Prototypes

This section provides background information regarding all the prototype tests completed to date. The details of each test are presented in Table 1.

Table 1. Overview Prototype Test Descriptions

Test ID	Test Objective	Description of Prototype used and of Basic Test Method	Summary of Results	Date Conducted
S1	Compute correct values for angles	Python code numerically computed a solution of a 2D inverse kinematics problem of which the analytical solution was known. Angles corresponding to a single point were solved.	● Correct solution was obtained with fast convergence using Newton's method ● The method diverged when given a bad initial guess	
S2	Compute correct values for angles	Python code numerically computed a solution of a 3D inverse kinematics problem of which the analytical solution was known. Angles corresponding to a single point were solved.	● Correct solution was obtained with fast convergence using Newton's method ● The method diverged when given a bad initial guess	
E1	Check for interference in end effector design	Top and bottom parts of the end effector were mated together in onshape using the cylindrical mate. The angles were restricted by assuming what would be realistic on a physical model.	● The top part of the end effector could be successfully opened without hitting/overlapping the bottom end effector	2022-03-05

E2	Test the spring strength in a clipboard	A paintbrush was clamped in a clipboard and held at various angles and positions relative to the ground.	● The clipboard successfully held the paintbrush in place without any slippage or other movements	2022-03-06
E4	Ensure that the end-effector has enough compressive force structural integrity to lift objects such as a paint brush, nozzle, and camera.	A 3D printed end effector held various objects while being attached to the end of the arm.	● The end-effector was able to hold the mass of a pencil and paintbrush however slippage did occur	2022-03-11
E5	Test whether the end-effector spring has enough compressive force to hold a paint brush and a cell phone.	Someone's hand will open the end-effector and place a paint brush and a cell phone in the clamp and the spring force will hold the objects in place once the person lets go of the end-effector.	● The end-effector was able to hold the mass of a pencil and cell phone. Since some slippage did occur, there will be something like anti slip pads for furniture to increase friction and reduce slippage.	2022-03-11
E6	-Test whether the end-effector spring has enough compressive force to hold objects -Test if the robot arm can	Use a 3D printed prototype of the end-effector to connect to the robot and pick up an object.	● This test was successful, the arm did not bend or break under the weight of the end effector and attached object. ● The end effector can attach to the arm with no	2022-03-11

	support the end-effector and the object and that the end-effector can connect to the robot arm without any problems		issues.	
U4	Test if UI is able to receive and store user input	A number was inputted into the height, width and length sections of the calibration interface to ensure that the numbers could be recorded in variables and printed to verify	● The UI was able to successfully store and print the numbers inputted by the user to the terminal	2022-03-11
U5	Test UI for exception handling when values are outside of range	Multiple tests were conducted to verify the exception handling of the UI: 1. A number outside of the range was inputted into the UI 2. Letters were inputted into the UI 3. No input was also tested	● The error window successfully popped up for the three tests completed	2022-03-11
U6	Test file upload button on drawing interface	4 SVG files were stored in different locations and uploaded to the UI and the file location was printed to the terminal	● The files were accepted by the UI and python was able to print the file paths	2022-03-12
H3	Find the maximum radius of the robot's workspace.	To find the maximum radius, motors will be removed from the robot and the arm will be extended manually and measured using a measuring tape.	● The maximum radius of the arm with the end effector is around 76.2cm.	2022-03-11

3.1 Test Results

This section outlines some background information for each test and presents the results from each prototype test.

3.1.1 Open/Close End-Effector

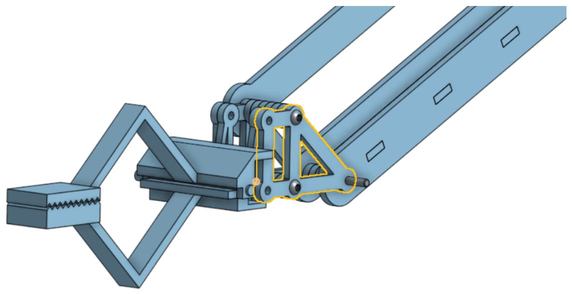


Figure 7. End effector in closed position

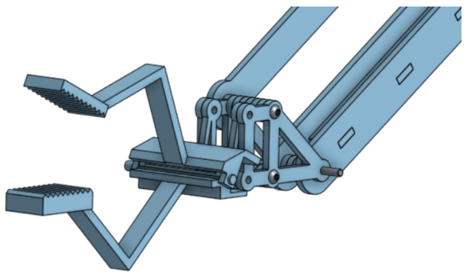


Figure 8. End effector in opened position

The purpose of this test is to determine using an analytical model if the end effector could successfully open and close without any disruptions or interferences being caused between the different parts.

3.1.2 Horizontal/Vertical Positions of a Clipboard Holding a Paintbrush

Figure 9. Clipboard and paint brush horizontal to the ground

Figure 10. Clipboard and paint brush vertical to the ground

The purpose of this test is to determine if the spring from the clipboard has the torque required to keep a paintbrush in place

3.1.3 Solution to Two-Dimensional Inverse Kinematics Problem in Python

To provide some background information about the prototype, the previous prototype will be summarised. The previous inverse kinematics (IK) solution solved a two-dimensional (2D) problem of which the analytical solution was known. To solve the IK, Newton's method for systems of equations took an initial guess of the solution (a set of angles) as input, then iterated to find the correction solution using a jacobian matrix. The previous prototype proved to perform well when the initial guess of the solution passed into Newton's method was relatively close to the actual solution. The method diverged when unreasonable initial guesses were passed into the function, due to the sinusoidal nature of the functions. A different method could have been chosen to solve this issue, but Newton's method is quadratically convergent (very efficient) and converges especially quickly when the functions are quadratic. Locally, the sinusoidal functions resemble quadratic functions and can be approximated fairly well using a second-order Taylor series as shown in Figure 11.

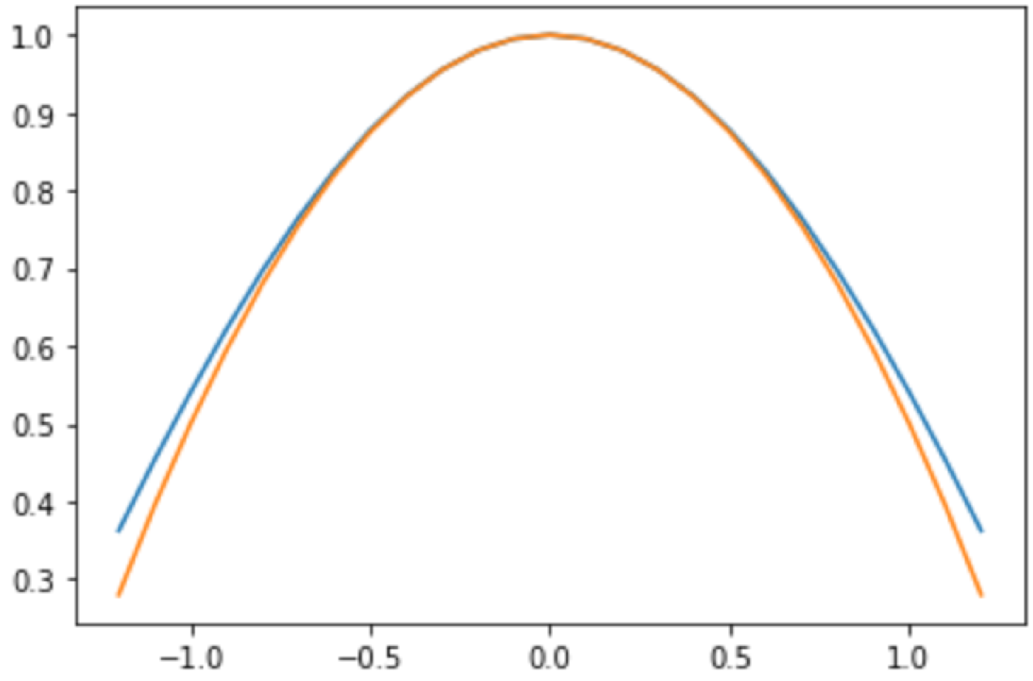


Figure 11. Second-order Taylor series approximation of sin(x), where sin(x) is blue and the quadratic is orange.

The chosen solution for this divergence problem was to analyse the spatial characteristics of the situation to determine a good initial guess for Newton's method. To generate a good initial guess, it was easier to test this concept using the scenario of the actual robot because the constraints of the workspace were defined, which are presented in Section 3.2.3. For this test, the code for the 3D IK solution written for the previous prototype was used and modified to fit the specifications of the robot and its workspace.

The most logical idea for a good initial guess was the solution for the angles when the robot is at the average position. This position was defined to be in the middle of the rectangular surface on which the arm paints.

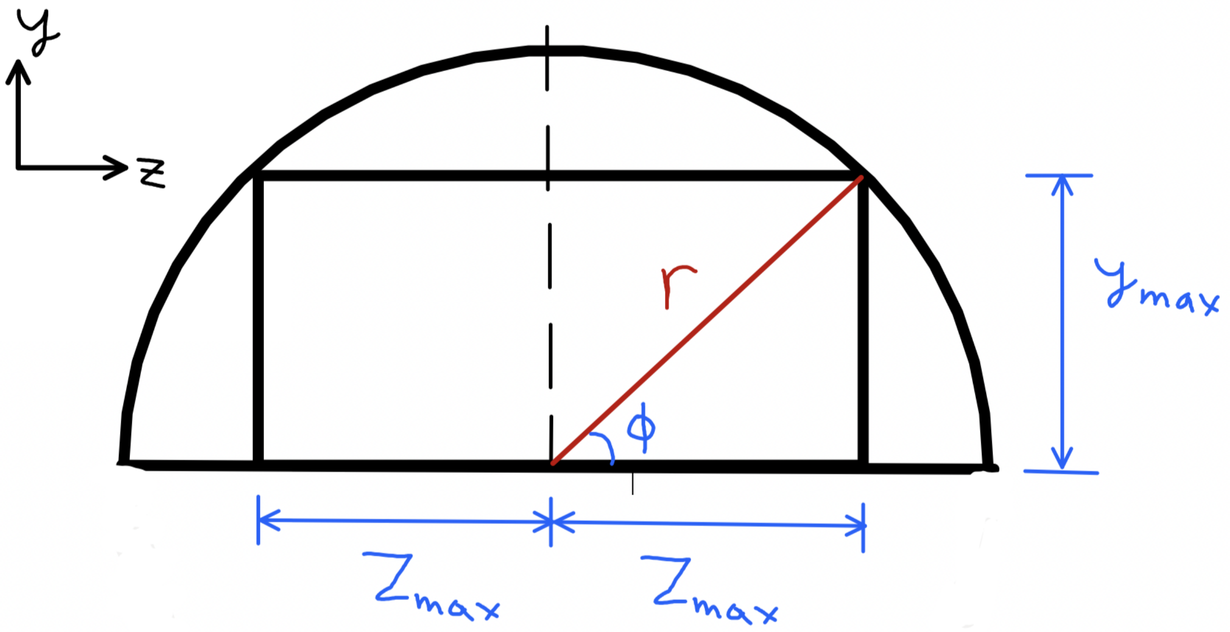


Figure 12. Schematic of the paintable surface.

Using single variable optimization, the y_{max} and z_{max} were calculated to maximise the surface area of the rectangle, which is explained in Section 3.2.3. An x_{max} was calculated based on assumptions discussed in Section 3.2.3. The average point lies at (x_{max}, $y_{max}/2$, 0). To use the angles of the joints at this point as an initial guess, they first need to be solved. The angles were solved using Newton's method and the initial guesses used were based on visually inspecting the Onshape CAD file. The angles were validated by plugging the angles into the equations and obtaining the correct coordinates.

To test the accuracy of Newton's method, ranges for each angle were generated. The ranges of these angles were divided evenly into 100 angles and using three for loops, coordinates for each point corresponding to each set of angles were calculated using forward kinematic equations. Then, the coordinates were passed as inputs into Newton's method and using the same initial guess (x_{max}, $y_{max}/2$, 0), the angles corresponding to each set of coordinates were calculated numerically. To test the accuracy of the solutions, the actual angles generated were compared to the numerically calculated angles using an error function, where the absolute values of the differences between each angle are calculated. If the absolute values of the differences between all three angles were less than 0.01 radians, then the solution was considered correct. A function called *testAccuracy* was created and if the numerical solution passed the error test, then a variable called *accuracy* was increased by one. To check the percentage of test cases that were correct, the final value of *accuracy* was divided by the total number of test cases. After running 1000000 test cases, covering the entire rectangular surface evenly, the *testAccuracy* function returned 1.0, indicating that 100% percent of the test cases were solved correctly.

3.1.4 Usability of User Interface

Test U4: User Interface information storage

Test U4 was conducted to determine if the coded file could properly receive and store information. The video linked above shows the test conducted and the results of the test. From the video, the user interface was able to store the numbers inputted by the user and successfully printed the width, height, and length of the brush into the terminal in VSCode

Test U5: User Interface Exception Handling

Test U5 was conducted to determine if the user interface was capable of properly handling incorrect outputs from the user, such as words instead of numbers, or no input into the system. The video linked above shows the tests conducted and the results of the tests. From the video, the UI was able to detect when the user inputted the wrong type of information, and when the user did not input any information into the system.

Test U6: Uploading and Retaining Paths for SVG Files

Test U6 was conducted to determine whether the user interface was able to store the link to an SVG file by placing four SVG files into separate locations and uploading them independently into the UI. The file paths were printed to the terminal to determine accuracy. From the video linked above, the UI was able to accept SVG files and print the path to the file.

3.1.5 Structural Strength of the End-Effector

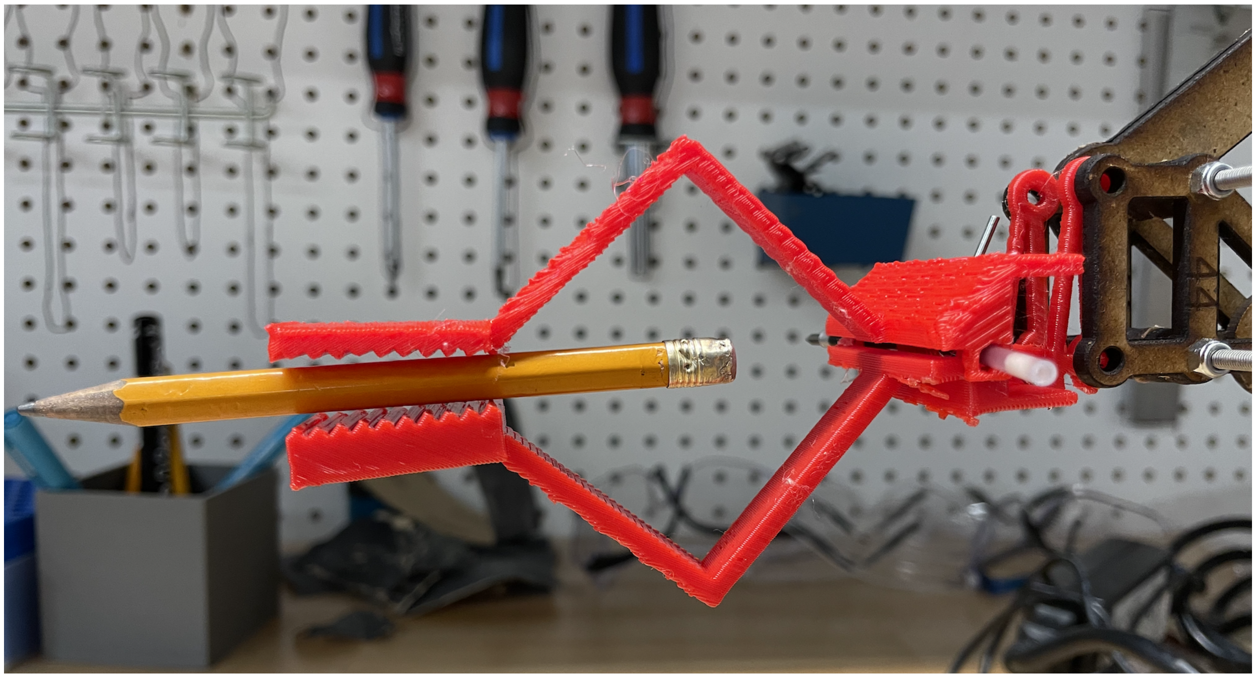


Figure 13. The end-effector holding a pencil

This test was to determine if the end-effector has the structural integrity to hold objects in place without having damage occur to the object or end-effector. A maximum safe mass was then going to be chosen based on the results of the test.

3.1.6 Compressive Strength of the End-Effector

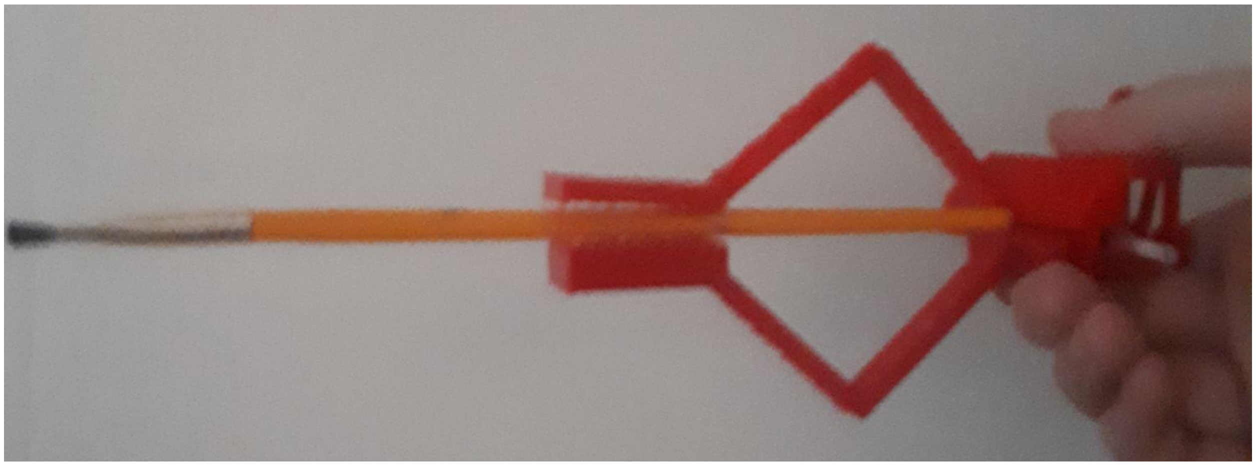


Figure 14. The end-effector holding a paintbrush

The purpose of this test was to determine whether the compressive strength of the spring was strong enough to hold an object in the end effector without slippage occurring.

3.1.7 Robotic Arm Supporting End-Effector and Object

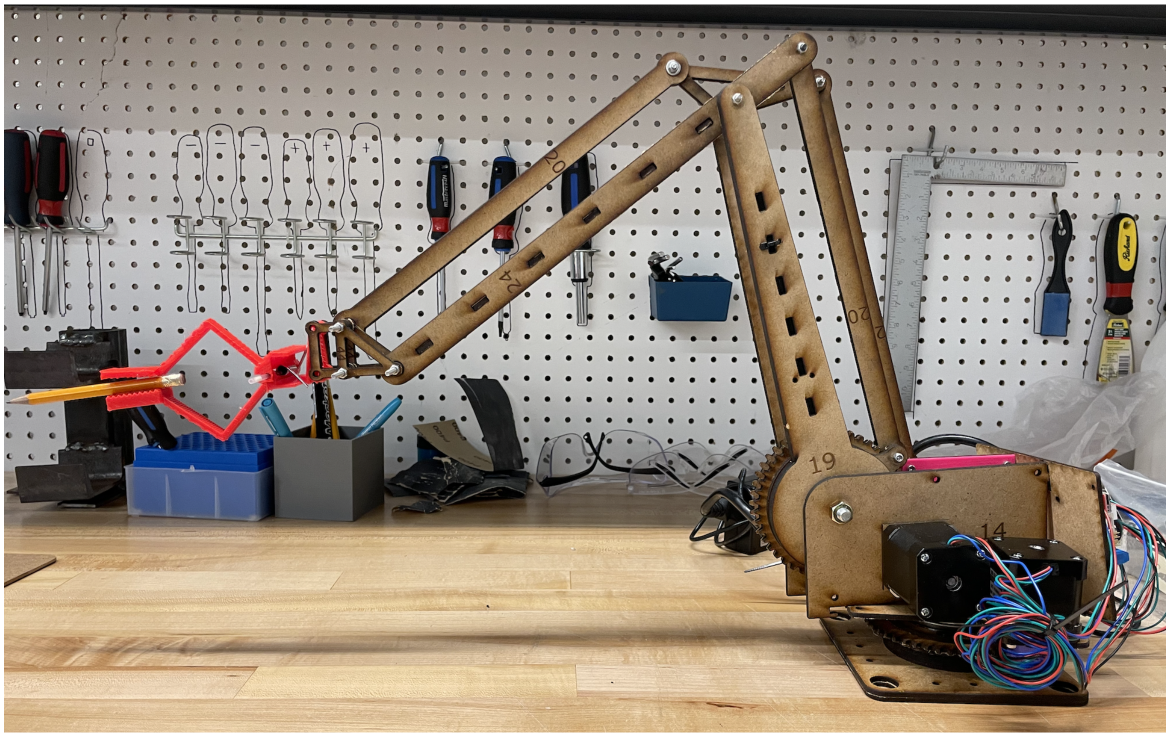


Figure 15. The end-effector holding a paintbrush while being attached to the arm

The purpose of this test was to determine if the arm had the structural strength to hold the combined masses of the end effector and another object.

3.2 Analysis of Results

3.2.1 Open/Close End-Effector

This test was to determine whether the dimensions of the CAD were sufficient in size to open the end effector while preventing the back end of the bottom and top end effectors from touching. When the Z angle of the cylindrical mate connected to the top and bottom pieces of the end effector was restricted between 0 and 30 degrees, the end effector could open a sufficient amount without the top end effector hitting the bottom end effector.

3.2.2 Horizontal/Vertical Positions of a Clipboard Holding a Paintbrush

This test was carried out to determine whether or not the spring of a clipboard had a sufficient amount of force to hold a paint brush in place under various angles and positions. The clipboard was held at extreme angles in the vertical and horizontal positions, as well as angles that would be seen by the robot. The clipboard held the paint brush in all of these positions without any complications.

3.2.3 Solution to Three-Dimensional Inverse Kinematics Problem in Python

When constructing the inverse kinematics solution, an assumption was made that is important to state: the maximum of the magnitude of the first angle at the base was chosen to be 45°, or π/4 radians. This assumption was not crucial, but if this assumption were not made, then the user would be allowed to place the robot anywhere they would like. With this flexibility, the user could place the robot too far from the wall, severely limiting the amount of the workspace that covers the paintable surface. Also, if the robot is placed too close to the wall, the user could have difficulty handling the end-effector when interchanging devices to be held. In future iterations of the design, it is possible to have flexibility in where the robot is placed. Due to the assumption of the angle and the measured maximum reach of the robot, the robot should be placed at a distance of 45 cm from the wall on which it is painting, as shown in Figure 16.

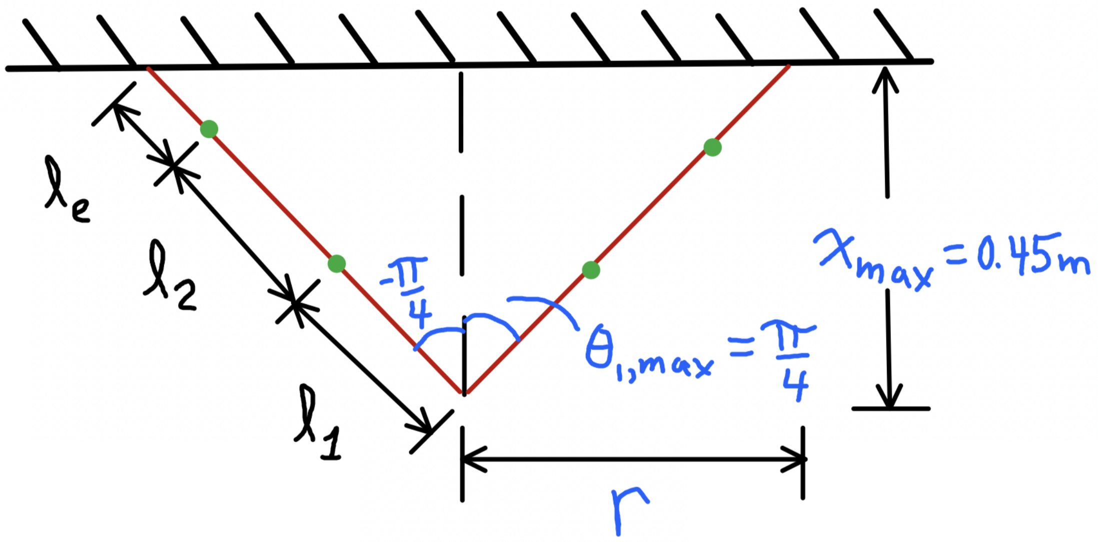


Figure 16. Top view showing the maximum range of angle one and the maximum distance from the robot base to the wall.

Based on the assumption that x_{max} is 45 cm and $|\Theta_{1,max}|$ is π/4, the other dimensional constraints were calculated using optimization, and the ranges for the angles were calculated using inverse kinematics.

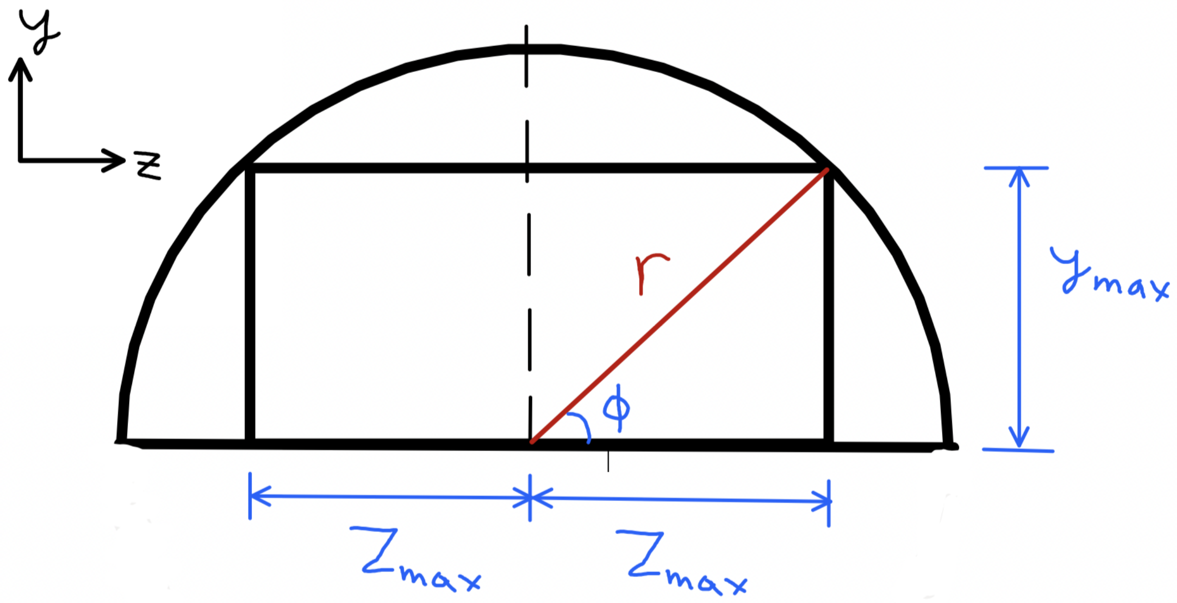


Figure 17. Paintable surface to be optimised.

To give the user the maximum surface area to paint, the rectangle shown in Figure 17 was optimised. The radius, *r*, was calculated using basic trigonometric functions, which is obvious from looking at Figure 16. Knowing *r*, an expression for the surface area in terms of only the angle φ could be used to optimise the area, which would give y_{max}, and z_{max} as shown in the following equations.

$$A = 2z_{max}y_{max}$$

$$A = 2(r \cdot \cos(\varphi))(r \cdot \sin(\varphi))$$

$$A = r^2\sin(2\varphi)$$

$$\frac{dA}{d\varphi} = 0 = 2r^2\cos(2\varphi)$$

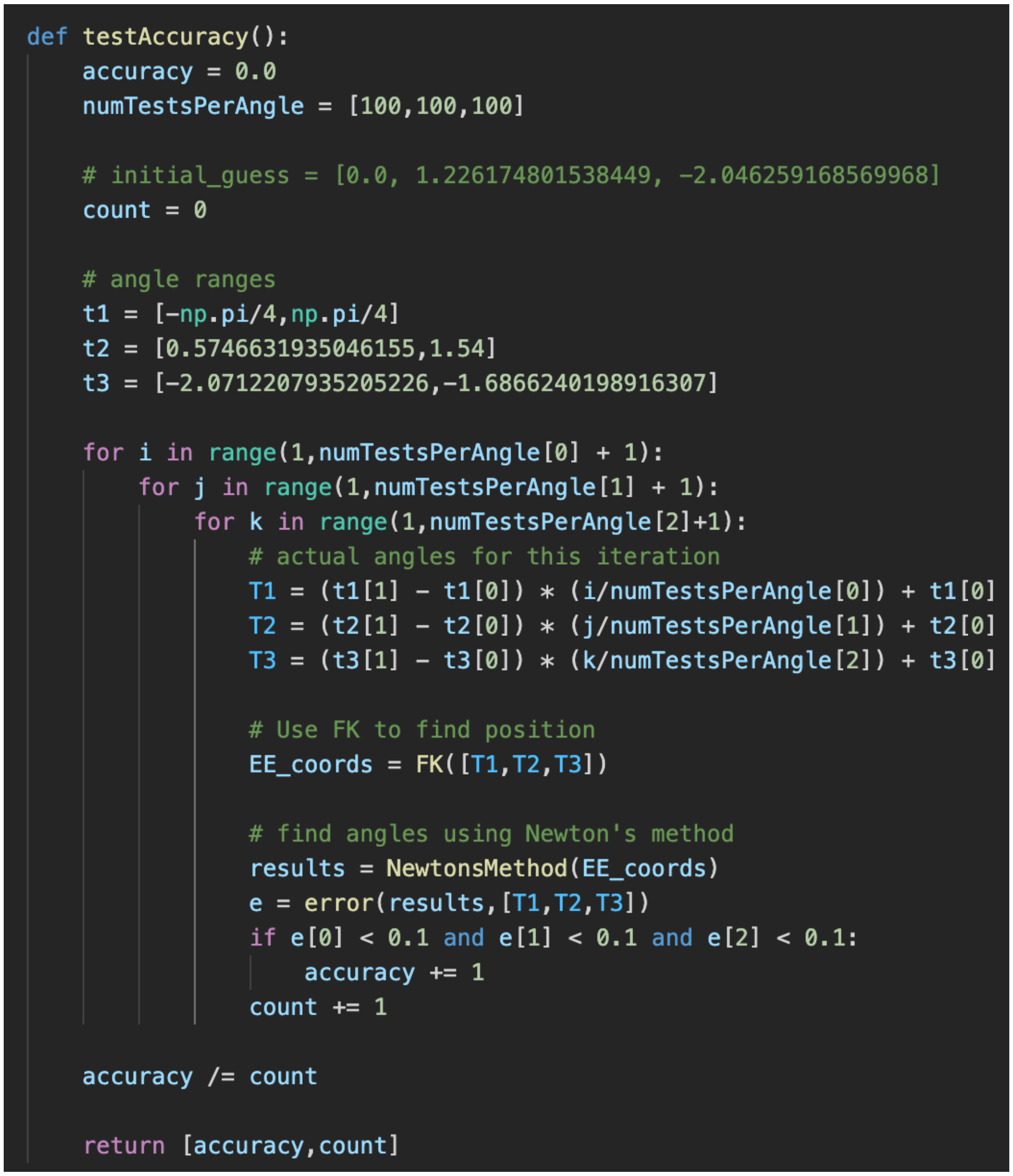
$$\varphi = \frac{\pi}{4}$$

Because φ is π/4, y_{max}, and z_{max} are equal.

With the ranges for the three angles known, the function test accuracy was coded. The function works as follows:

1. Divide the angle ranges and establish the ranges of the angles.
2. Using three for loops, generate a new set of angles ($T_{1,ijk}$, $T_{2,ijk}$,$T_{3,ijk}$) for each iteration of the loops.
3. Using the function *FK* (for forward kinematics), calculate the coordinates of the end-effector at the given angles ($T_{1,ijk}$, $T_{2,ijk}$,$T_{3,ijk}$).

4. Use the coordinates obtained from *FK* as inputs for Newton;s method to solve for the angles numerically using IK.
5. Use the function *error* to compare the actual angles with the angles calculated by Newton's method.
    a. If the angles calculated using IK are correct, then increment variable *accuracy* by
6. At the end of each test case, increment the variable *count* by 1 to keep track of the number of test cases.
7. Finally return the percentage of correct cases by dividing *accuracy* by *count*.



```
def testAccuracy():
    accuracy = 0.0
    numTestsPerAngle = [100,100,100]

    # initial_guess = [0.0, 1.226174801538449, -2.046259168569968]
    count = 0

    # angle ranges
    t1 = [-np.pi/4,np.pi/4]
    t2 = [0.5746631935046155,1.54]
    t3 = [-2.0712207935205226,-1.6866240198916307]

    for i in range(1,numTestsPerAngle[0] + 1):
        for j in range(1,numTestsPerAngle[1] + 1):
            for k in range(1,numTestsPerAngle[2]+1):
                # actual angles for this iteration
                T1 = (t1[1] - t1[0]) * (i/numTestsPerAngle[0]) + t1[0]
                T2 = (t2[1] - t2[0]) * (j/numTestsPerAngle[1]) + t2[0]
                T3 = (t3[1] - t3[0]) * (k/numTestsPerAngle[2]) + t3[0]

                # Use FK to find position
                EE_coords = FK([T1,T2,T3])

                # find angles using Newton's method
                results = NewtonsMethod(EE_coords)
                e = error(results,[T1,T2,T3])
                if e[0] < 0.1 and e[1] < 0.1 and e[2] < 0.1:
                    accuracy += 1
                count += 1

    accuracy /= count

    return [accuracy,count]
```

Figure 18. Python code for the *testAccuracy* function, which tests the accuracy of the IK solver.

An accuracy of 100% indicates that the IK solution in 3D works and can be integrated to use to control the stepper motor through Arduino. As a qualitative test to ensure that the solutions obtained were physically reasonable, the movement of the arms were animated as shown in the following videos.

Animation 1
Animation 2

In the first video, the arm is being viewed from the side in the x-y plane as it is being raised. In the second video, the arm is being viewed from the back in the y-z plane as it moves in the motion of a star, followed by a circle. These animations were created solely based on the angles calculated from Newton's method. By visual inspection, the motion of the arm looks reasonable and as expected, which validates the accuracy of 100% calculated by the *testAccuracy* function.

In the second animation, the arm moves in a star and circle pattern. This proves that given the correct coordinates, the arm can use IK to adjust the joint angles accordingly to produce the specified patterns. The next steps in the testing of the code will be to interface between Python code and Arduino code using the pyfirmata library, and to interpret a set of coordinates extracted from an SVG file using a Python library: possible svglib.

3.2.4 Usability of User Interface

Tests U4-U6 were conducted to determine the accuracy of the backend of the user interface. The test results for the UI shown in Section 3.1.4 show that the UI was able to successfully store information and handle incorrect inputs from the user. From the results, it can be determined that the backend of the user interface can run efficiently without any major issues

3.2.5 Structural Strength of the End-Effector

The end-effector was able to hold the mass of a paintbrush and pencil even after suffering structural damage. This leads the group to assume that similar masses should be acceptable with a structurally intact end-effector. However, during the tests slippage did occur while holding a paintbrush and pencil. Less slippage occurred with a cell phone. These results may be caused by the different centres of mass of the various objects as well as the shape contrast.

3.2.6 Compressive Strength of the End-Effector

The end-effector was able to hold a pencil and paintbrush in place, however there was slippage that occurred. The end effector will be modified to minimise possible slippage. A cell phone was also held by an end-effector. This test will not be conducted while attached to the arm due to the mass of the average cell phone in the group exceeding the maximum load capacity of the arm (without including the mass of the end effector).

3.2.7 Robotic Arm Supporting End-Effector and Object

The robotic arm was able to be successfully connected to the end-effector. While attached the end-effector was easily able to open and close to accommodate for the placement or removal of an object. The robotic arm was able to withstand the weight of the end-effector in addition to the object the end-effector was holding.

4.0 Current Bill of Materials

Some small changes have been made to the bill of materials bringing the total down to 17.20$ which is under the 50$ limit. At the time it was created, an arduino mega and shield were included; they have since been taken out after discovering that they are given in class. See spreadsheet for details.

5.0 Prototyping Test Plan

Prototyping is important because it allows specific design elements to be isolated and tested. The approach chosen for this project is to conduct many tests that are smaller. This approach will allow the team to ensure that each part of the design works before building a more complicated part. To ensure that all tests can be completed on time, the following table will provide a structured plan that outlines the details of the prototype testing. Since there are several subsystems, preliminary tests are not necessarily dependent on each other.

Table 1. Prototype Test Plan

Test ID	**Test Objective**	**Description of Prototype used and of Basic Test Method (What)**	**Description of Results to be Recorded and how these results will be used (How)**	**Estimated Test duration and planned start date (When)**
Software				
S3	Compute the correct values for angles.	Python code will numerically compute a solution of a 2D inverse kinematics problem where the end-effector must trace a line to simulate painting of a line.	-Compute an analytical solution of a static end-effector at multiple nodes (points on the line) and check if the angles are correct when the end-effector passes through each point -If possible, make an	Duration: 15 minutes Test date: March 14

			animation of the arm drawing and check whether the line is correct and the robot movement is feasible	
S4	Compute the correct values for angles.	Python code will numerically compute a solution of a 2D inverse kinematics problem where the end-effector must trace a circle to simulate painting of a circle.	-Compute an analytical solution of a static end-effector at multiple nodes (points on the circle) and check if the angles are correct when the end-effector passes through each point -If possible, make an animation of the arm drawing and check whether the circle is correct and the robot movement is feasible	Duration: 1 minute Test date: March 14
S5	Compute the correct values for angles.	Python code will numerically compute a solution of a 3D inverse kinematics problem where the end-effector must trace a line on a 2D surface to simulate painting of a line.	-Compute an analytical solution of a static end-effector at multiple nodes (points on the line) and check if the angles are correct when the end-effector passes through each point -If possible, make an animation of the arm drawing and check whether the line is correct and the robot movement is feasible	Duration: 1 minute Test date: March 15
S6	Compute the correct values for angles.	Python code will numerically compute a solution of a 3D inverse kinematics problem where the end-effector must trace a circle on a 2D surface to simulate painting of a line.	-Compute an analytical solution of a static end-effector at multiple nodes (points on the circle) and check if the angles are correct when the end-effector passes through each point -If possible, make an animation of the arm	Duration: 1 minute Test date: March 15

			drawing and check whether the circle is correct and the robot movement is feasible	
S7	Ensure that the Arduino can interpret and use Python code to perform calculations.	Python code will be interpreted by Arduino and cause a motor to rotate. The pyfirmata library will be imported to allow Python and Arduino to communicate.	If the motor moves, the test is successful because it is evidence that the python code is able to be used by the Arduino. If no movement occurs, the code did not work and debugging of code or investigation of other communication methods will be required.	Test duration: 10 minutes Test date: March 16
Hardware				
H1	Ensure that the Arduino can communicate with the motors.	Motors and the Arduino will be connected to a breadboard. Arduino code will instruct the motors to rotate.	If the hardware components can communicate, the motor will move. Once this test is passed, Test 9 can take place and aim for a more precise result.	Test duration: 5 minutes Test date: March 16
H2	Check whether the Arduino can instruct the motors to rotate precisely to a specified angle.	The Arduino will make each stepper motor rotate by 90º. A straight object, such as a stick, will be attached to the motor to help measure the accuracy of the rotation.	The angle of rotation will be measured using a protractor. The start and end points of the stick will be marked as a reference for the measurement.	Test duration: 30 minutes Test date: March 16
H4	Test the kill switch	The Aruduino will be rotating a motor indefinitely. The kill switch button will be connected to the Arduino and if it is pressed, the Arduino should stop the motor from rotating.	If the motor stops rotating, the kill switch works. Otherwise, there is a hardware problem or a software problem that must be identified and corrected to test the switch again.	Test duration: 10 minutes Test date: March 16

User Interface (UI)				
U7	Test if stop button on UI is able to stop system	Design a simple application using Python (Tkinter) that has a ‘stop’ button. The user will hit the stop button and check to see if the system outputs a specific value (“the value to stop the code”)	If the output from the system on the IDE is a specific value (“0” for off), the stop button is able to stop the program	Test Duration: 5 minutes Test Date: March 15
U8	Test if user input can be used to control a hardware system.	Design a simple application using Python (Tkinter) that takes in a user input and passes it on to the Python code. The user will indicate how many times a motor should rotate and change direction.	If the motor rotates and changes direction the same number of times that the user indicated through the user interface, then this result confirms that user input can be passed through Python and the Arduino to specify the action of hardware.	Test duration: 20 minutes Test date: March 16
Overall Functionality				
F1	Test if the code can instruct the end-effector to move to a precise position.	A position in 3D space will be passed into the code, which should make the motor rotate by the angles calculated using inverse kinematics to place the end-effector in the correct location.	To confirm that the end-effector moves to the correct location, the location in 3D space will be identified using a measuring tape. A physical marking will be placed in the location. If the end-effector moves to the location of the physical marking, the test is successful. If the robot fails the test, the error must be identified and fixed before repeating the test.	Test duration: 45 minutes Test date: March 16
F2	Test if the robot can draw a straight line	The test conductor will place a marker in the end effector clamp and measure the distance	The robot should be able to successfully draw the line on the surface. The success of the test will be	Test duration: 45 minutes Test date:

	on a surface using a marker.	from the base of the robot to the surface to draw on. The conductor will then input the distance in the user interface and press start. The robot should then draw the programmed line on the surface. The specifications of the line will be coded so that the user does not have to pass a vector drawing into the user interface.	based on whether the line is of the distance specified in the code and if the line is drawn in the correct location. If this comprehensive test of a simple drawing is successful, more complicated comprehensive tests can be conducted, such as drawing more complicated shapes and receiving drawing instruction through the user interface in the form of vector drawings.	March 17
F3	Test if the robot can draw a circle on a surface using a marker.	This test is identical to Test F2, but the robot will draw a circle.	The results of the test will be processed identically as in Test F2, but the radius of the circle and the position of its centre will be measured.	Test duration: 45 minutes Test date: March 18
F4	Determine whether the robot can draw a drawing given a vector drawing passed into the user interface.	This test is identical to Test F2, but instead of a preprogrammed line, a vector drawing will be passed into the user interface and processed to instruct the robot to draw the given diagram.	If the software can process the vector drawing in a timely manner without any errors and the arm can correctly draw the line given by the vector drawing, then the test is successful. The same measuring techniques in Test 18 will be used to validate the line.	Test duration: 45 minutes Test date: March 19
		End Effector		
E7	Test if anti slip pads for furniture will create adequate friction to	Someone's hand will open the end-effector with the newly installed anti-slip pads and place a paint brush and a cell phone in the clamp and	If the end effector can hold the object (paint brush, camera, or nozzle) without it slipping then this test is successful	Test duration: 1 minute Test date: March 16

	stop an object from moving in the end effector.	the spring force will hold the objects in place once the person lets go of the end-effector.		

6.0 Materials for Each Prototype

This section contains a table that outlines materials required to build each prototype and conduct its corresponding test. The numbers in Table 1 correspond to the numbers in the "Test ID" column of Table 2.

Table 2. Materials for each prototype test

Test ID	Materials Needed
S3	Python
S4	Python
S5	Python
S6	Python
S7	Python
H1	Python Aruino Pyfirmata library Motors
H2	Motors Arduino Breadboard
H4	Arduino Motors Switch
U7	Python
U8	Python
F1	Robotic arm Arduino Motors End effector

F2	Marker End effector Robotic arm Arduino Motors
F3	Marker End effector Robotic arm Arduino Motors
F4	Python Marker End effector Robotic arm Arduino Motors Inkscape

7.0 Risks and Contingency Plans

Table 3. Risks and Contingency plans

Risk	**Issue**	**Contingency plan**
1. High Costs	- The budget for this project is 50$, certain electronics cost a fair amount of this budget. This could be an issue moving forward.	- A detailed bill of materials must be created and followed. Things must be ordered in advance to avoid extra shipping costs.
2. Parts not coming in	- If parts don't come in on time it could greatly affect the progress of our arm.	- To avoid this issue, we must have a detailed list of materials and order online the parts as soon as possible
3. Lack of clarity	- Changes to the project and the responsibility of the arm will pose a great risk to the	- If something is not clear specific questions must be directed to the client

	overall project.	about what the arm must accomplish.
4. Scheduling issues (prototype)	- There is not much time to develop each prototype, this could lead to time management issues down the line.	- The schedule that has been created on Wrike must be adamantly followed to make sure the project stays on track. If a team member falls behind on their portion they must reach out and ask for help.
5. Technology risks	- Technology risks such as the 3D printing machines or laser cutting machines not working could pose a threat to the success of the prototypes.	- The prototype plan must be followed. The plan will make sure that the construction of the end effector is completed early, this way if the machines are not working we will still have plenty of time to complete the task.
6. Scheduling issues (software)	- Software is difficult to schedule because we will not know how long it will take to do some of the parts of it.	- To avoid this we will take time to estimate how long each portion will take and divide it up equally. If someone realizes that their section is going to take longer than originally planned they will reach out to the group and another team member can help them out.
7. Time issues (3D printing)	- 3D printing takes a considerable amount of time and if we do not take the time to calculate how long the 3D printing will take,	- To make sure we have plenty of time to complete the sections of the end effector using 3D printing the prototype test plan

	it will be a big problem.	must be followed. It must allow time for the 3D printing to be completed and it must not be attempted the night before.
8. Code Issues	- Bugs and errors could be a result of rushing near the end of the project to get the software done on time.	- Frequently checking the code for bugs and errors can prevent this. Codes should also not be left with error messages to make sure nothing is forgotten.

8.0 Conclusion

The Department of National Defence has a need for a robotic arm that uses inverse kinematics to paint surfaces. In this deliverable, the first prototype and second test plan in Deliverable F were used to create a second prototype and third prototype testing plan as well as an updated bill of materials necessary for developing the final product.