

Project Deliverable G - Prototype II and Customer Feedback

GNG 1103 Group C03

Benoit Tremblay (300236751)

Rebeca Poulin (300236741)

Jess Beardshaw (300227707)

Isabelle Barrette (300228564)

Sarah Alkadri (300245117)

University of Ottawa

March 13th, 2022

Introduction	4
1. Client Feedback	5
2. Prototype II	6
2.1 Camera End-Effector	6
2.1.1 3D Printed Products	6
Figure 1. Second 3D printed prototype for camera end effector with posterior and anterior views	6
2.1.2 3D CAD Model	7
Figure 2. CAD model of prototype II of the camera end effector	7
2.2 Corrosion Remover and Painting End Effector	7
Figure 3. String of commands in Command Prompt to reach desired folder directory	8
2.3 Coding and Inverse Kinematics	10
2.3.1 Kinematics concept and search algorithm	10
2.3.2 Inverse Kinematics code implementation	10
Figure 7. Screenshots of sections of IK code	11
Figure 9. Screenshots of code compilation message	13
Figure 10. Screenshots of Servo motor connections	13
2.3.4 Code Files (OneDrive Links)	13
2.4 User Interface code	14
Figure 11. Screenshot of the user interface code	14
2.4.1 Definitions of variables	15
2.4.2 Main Body code	15
Figure 12. Compiled user interface code and image of user interface	15
Figure 13. Functioning entry box to input the distance the robot is from its workspace	16
Figure 14. Camera end effector control page	16
Figure 15. Programming Interface once “Take Picture” button is pressed	17
Figure 16. Water and Paint end effector control page	17
2.5 Safety	17
2.5.1 Emergency Stop	17
Figure 17. Emergency stop function within the user interface	18
2.5.2 Motion Detection Sensors	18
Figure 18. Motion detection sensors functioning	18
3. Prototype Test plans	19
3.1 Prototype II Test Plan	19
Table 1. Prototype II Test Plan with results	19
3.2 Prototype III Test Plan	20
Table 2. Prototype III Test Plan with expected results	20
4. Updated Bill of Materials	22

Table 3. Bill of materials with total product cost estimate	22
5. Target Specifications	23
Table 4. Target Specification of all aspects	23
Conclusion	24
Bibliography	25

Wrike Link:

<https://www.wrike.com/workspace.htm?acc=4975842&wr=20#path=folder&id=824968763&c=list&vid=64521612&a=4975842&t=830081719&so=5&bso=10&sd=0&st=nt-1>

Introduction

As our design day approaches, we begin to invest more time and effort into making prototypes with feasible concepts. We are beginning to conceptualize ways that our prototypes will come as one to form our final functional product that will be presented on a design day. During this week's deliverable, we have gone more and more in-depth about the specific functions of our different prototype concepts. We have ameliorated our previous prototypes after encountering the actual robot and seeing its dimensions and motor locations.

Our first prototype is the camera end-effector which we have 3D-printed from a newly augmented CAD model. Our coding and inverse kinematics prototype has been refined to work with available servo motors on the robot arm. Our corrosion remover AI has started the development stage and will be able to provide a particular and new element to our final product. Notably, our user interface code has dramatically evolved and started to connect to different robot functions. Finally, we have spent much time ameliorating the arm's safety measures with our buzzer and camera feed.

1. Client Feedback

At the previous client meeting, the client did not give our group feedback regarding the development of the end effectors, user interface or inverse kinematics software. However, the client expressed his interest in our corrosion detection software, and he encouraged our solution's continued development.

At the next client meeting, our group is planning to further our prototypes by testing our inverse kinematics code and end effectors on the robot arm and the user interface and the corrosion detection software.

2. Prototype II

As we continue testing, we will have to update and change each subsystem significantly as we get new feedback, better ideas, and encounter technical problems. As our knowledge expands, we are constantly changing our game plan, referring to the target specifications and solving the client's needs most efficiently.

2.1 Camera End-Effector

After printing the first end effector, we changed the overall design completely and finalized which camera we were deciding to use. A central feature is quickly removing the back piece to access the camera inside. We realized this feature is unnecessary as the client does not need to access the camera.

We decided to remove the clips feature and instead create a simple box shape where the camera slides inside. The back piece will still be bolted to the finger using spacers and nuts but will also be bolted directly to the anterior piece of the end effector. This eliminates our concern of the clips malfunctioning, not fastening correctly, or falling off when the arm is moving. Following that, the semi-circle shape is replaced with a box shape, following the camera's shape. The camera lens protrudes out the front and is secured interiorly using four bolts securing each corner of the camera. The bolts extend through the posterior side and are fastened with nuts.

2.1.1 3D Printed Products

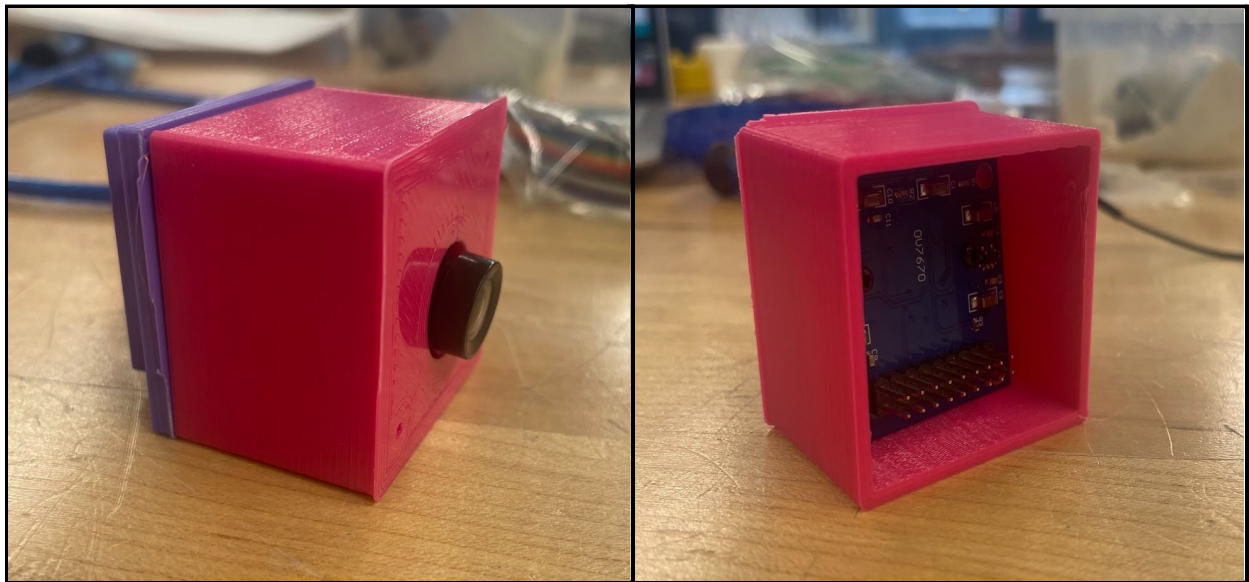


Figure 1. Second 3D printed prototype for camera end effector with posterior and anterior views

2.1.2 3D CAD Model

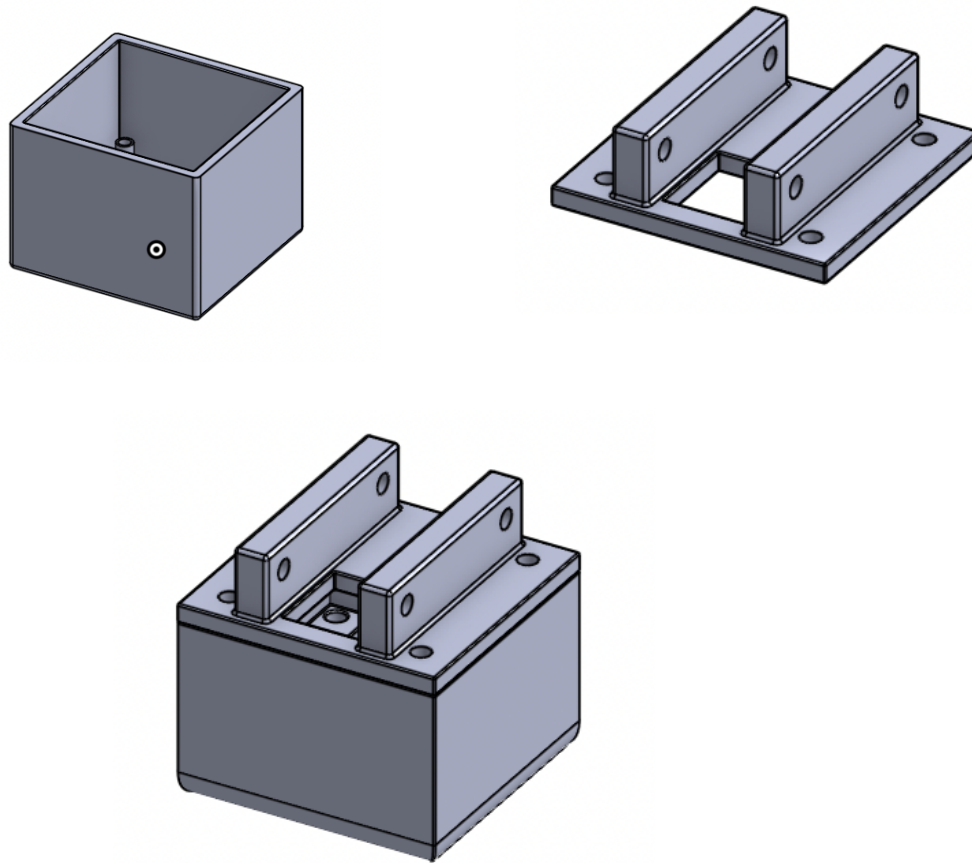


Figure 2. CAD model of prototype II of the camera end effector

2.2 Corrosion Remover and Painting End Effector

The primary issue with trying to understand and run a program written in python with our group is that none of us are super familiar with the program or coding in general. Another issue with the current corrosion AI is that it is a decently old program, five years old, so most of the libraries have updated and changed since, and some of the functions are either invalid or changed.

Firstly, when looking at the ReadMe file given by the code creator, it said to run a python command from the src directory (the file containing all the python files) to run them all at once. As a very inexperienced programmer, this was a big challenge. After doing research, we learned that python needed to be set up with a command prompt to run codes from the directory. To find the specific directory needed, we need to use "cd NameOfFolder" several times to set up the directory to the src folder.

```

Command Prompt
Microsoft Windows [Version 10.0.19044.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\btrem>cd\

C:\>cd python

C:\Python>cd Corrosion detection2

C:\Python\Corrosion detection2>cd corrosion-detection-master

C:\Python\Corrosion detection2\corrosion-detection-master>cd src

C:\Python\Corrosion detection2\corrosion-detection-master\src>

```

Figure 3. String of commands in Command Prompt to reach desired folder directory

The following error encountered was due to old code. The error that occurred had to do with the scikit-image library. Essentially the function `skimage.util.pad` gave an error message saying it does not have the `pad` attribute. After looking at all the changelogs in the past three years, we discovered that the `thread` `skimage.util.pad` was removed from scikit-image, the patch notes stated that `numpy.pad` can be used for the same effect. After pip downloading the NumPy library, the error message was fixed.

```

File "C:\Python\Corrosion detection2\corrosion-detection-master\src\imageTools.py", line 90, in pad
    return skimage.util.pad(arr,tuple(width),'edge')
AttributeError: module 'skimage.util' has no attribute 'pad'

```

Figure

Figure 4. Error message received because of scikit-image problems

The third encountered issue is found within the `imageTools.py` file, and it says `Assertion Failed`. We have researched and asked colleagues and friends about the issue, and it is most likely because of the `+directory+` portion of the code above the red-underlined code line. To fix this issue, we will most likely need to change the directory to suit the files directory in the laptop.

```

Command Prompt
C:\Python>cd Corrosion detection2
C:\Python\Corrosion detection2>cd corrosion-detection-master
C:\Python\Corrosion detection2\corrosion-detection-master>cd src
C:\Python\Corrosion detection2\corrosion-detection-master\src>python main.py test
Initializing classifier...
rust.10.jpg
Traceback (most recent call last):
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\main.py", line 50, in <module>
    clf = Classifier(directories)
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\classifier.py", line 59, in __init__
    self.pos = getImgs(directories["pos"],"pos",maxsize,pos)
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\classifier.py", line 48, in getImgs
    return loadImgs(directory,group,maxsize=maxsize)
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\classifier.py", line 21, in loadImgs
    files.append(Image(fname,directory,group,maxsize=maxsize))
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\image.py", line 29, in __init__
    self.rgb = getImg(name,direc,maxsize,img)
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\image.py", line 19, in getImg
    return loadScaled(name,directory=directory,maxsize=maxsize)
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\imageTools.py", line 41, in loadScaled
    file = load(name,directory)
  File "C:\Python\Corrosion detection2\corrosion-detection-master\src\imageTools.py", line 32, in load
    return cv2.cvtColor(cv2.imread(*_data/"directory+"/"+name),cv2.COLOR_BGR2RGB)
cv2.error: OpenCV(4.5.5) D:\opencv-python\opencv-python\opencv\modules\imgproc\src\color.cpp:182: error: (-215:Assertion failed) !_src.empty() in function 'cv::cvtColor'

C:\Python\Corrosion detection2\corrosion-detection-master\src>

```

Figure 5. Error message from imageTools.py about cv2 (OpenCV)

To fix this I had to change the directory to the first image in the file it is trying to read, so:

C:\Python\Corrosion detection2\corrosion-detection-master\data\test\(name of first image)(ex:picture.jpg)

This fixed the issue, and the program finally imported and classified the images, which brought us to the following error. Once this error was fixed, everything started to come together. The images in the files were being shown as scanned and imported into the command prompt.

```
C:\Python\Corrosion detection2\corrosion-detection-master\src>python main.py test
Initializing classifier...
rust.10.jpg
rust.10.xml
rust.11.jpg
rust.11.xml
rust.13.jpg
rust.13.xml
rust.15.jpg
rust.15.xml
rust.16.jpg
rust.16.xml
rust.17.jpg
rust.17.xml
rust.18.jpg
rust.18.xml
rust.19.jpg
rust.19.xml
rust.2.jpg
rust.2.xml
rust.20.jpg
rust.20.xml
rust.23.jpg
rust.23.xml
rust.24.jpg
rust.24.xml
rust.25.jpg
rust.25.xml
rust.28.jpg
rust.28.xml
rust.29.jpg
rust.29.xml
rust.30.jpg
rust.30.xml
rust.31.jpg
rust.31.xml
rust.32.jpg
rust.32.xml
rust.35.jpg
rust.35.xml
rust.36.jpg
rust.36.xml
rust.4.jpg
rust.4.xml
rust.45.jpg
rust.45.xml
rust.46.jpg
rust.46.xml
rust.48.jpg
```

Figure 6. Images from the test folder being processed within the program.

The final error in the code was a logic error. In the Image.py file, there is an issue on line 99 and 126. The error message is: if self.hsHist != None and hist!= None: ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all(). This was the most confusing error of all because, without programming knowledge, we could not understand the logic of the sentence. Luckily, our friend came to the rescue and told us to add .any() after each hist in lines 99 and 126. The code finally ran and displayed the final message following the change: Done! The code is now fully functional and reads images located in the test file, and writes the names of the files containing corrosion inside the ResultsTest text document. A video showing the program fully functional is shown below.

The program does not seem very accurate and will say that pictures without rust contain rust. There is also the option to set Hue Saturation and Edge threshold values, but currently, it does not work correctly and shows an error message. This is our next task for the corrosion detection program.

<https://youtu.be/xPfUXLRoVfo>

The before last error was once again an issue with scikit-images changelogs. When using the .remove_small_holes function in the program, the operation min_size was no longer a function that existed. After searching through various coding threads, we discovered that using area_threshold instead fixed the critical issues and brought us to the final result.

Upon further review, there is also a way to run the code inputting your Hue Saturation and Edge Threshold values. This section of the code still yields errors when the values are inputted. Pankaj helped us with this, and now the code runs entirely without errors.

The last part needed is calibration because it is inaccurate and says that pictures without corrosion contain corrosion. We need to look into this section of the code.

```
def writeResults(clf):
    res = open("../results/results_"+testDir+".txt", "w+")
    for im in clf.test:
        if im.label == 1:
            res.write(im.name+"\n")
    res.close()
```

The error is believed to lie within `clf` and the `==1` part, we need to figure out what they are and what they do.

2.3 Coding and Inverse Kinematics

We have refined the inverse kinematics code for our second prototype to fit the new test robot with servo motors and drivers. We will also use forward kinematics to find coordinates of corrosion spots (with corrosion detection software and camera program) and then acquire those coordinates to use for inverse kinematics.

2.3.1 Kinematics concept and search algorithm

Our forward kinematics derived concept will be to search a specific area using known angles to scan for corrosion using the software from section 2.2. The main concept for this part of the robot's functioning is to have a search pattern where at each "increment," the corrosion detection code described in section 2.2 will run through the user interface as described in section 2.4. The user interface will essentially store values into the code that can later be used in the inverse kinematics code from section 2.3.2. As for basic kinematics, movement will demonstrate servo motor rotations allowing the arm to move to specific coordinates.

2.3.2 Inverse Kinematics code implementation

To refine our code, we will need to ensure that the robot can get to an (x,y,z) coordinate with accuracy and precision. As opposed to our second prototype, we used a different analytical approach. This will essentially rely on finding triangles to form relationships between the joint variables and the x,y position. The triangle is drawn in purple (in the figure below), which tells us the end-effector's position in the base frame. We will be using three triangular relationships to solve IK from this new approach.

First off, we use the Pythagorean theorem for our right triangles. Secondly, we will use *SOHCAHTOA* to use the sine, cosine and tangent relationships. Thirdly, we can use the law of cosines when we do not have a right triangle.

We have simplified the code to fit a palletizing robot arm meaning that the angle that the Elbow servo motor performs stays constant and partly independent of the Shoulder servo motor rotation. The variables that required essential setup functions and analytical explanations are here below.

```
int i;

#define PI 3.14159265

float BaseLength = 3.229;
float ShoulderLength = 9.555;
float ElbowLength = 9.859;
float Hypot;
float x0 = 5; //Target end point TEST
float y0 = -8; //Target end point TEST
float z0 = 4; //Target end point TEST
float A, B, C,
    Theta,
    Alpha,
    Alpha_temp,
    Alpha_i,
    Beta_temp,
    Beta;
```

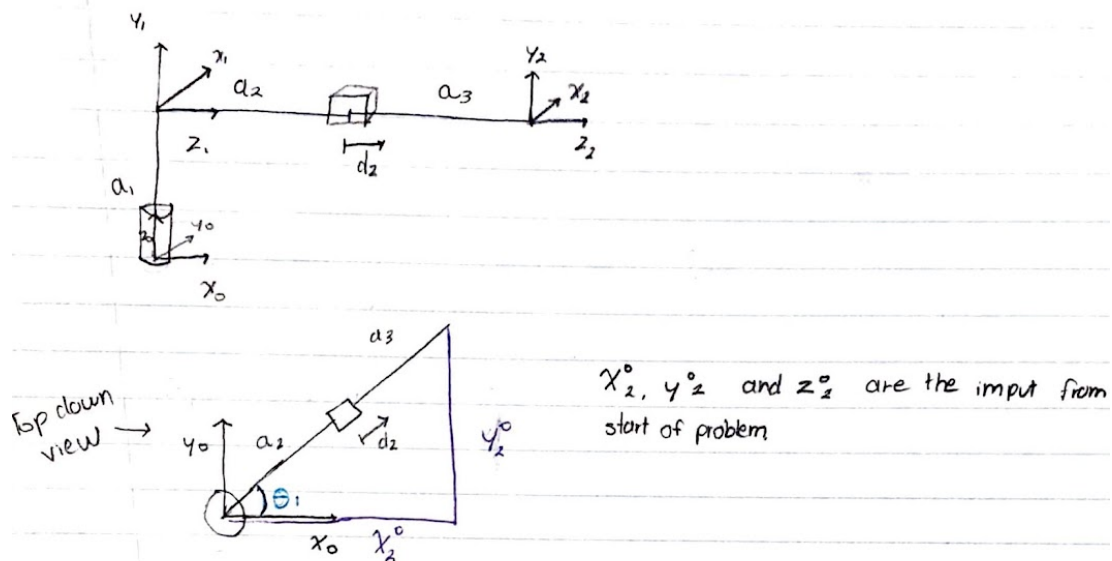
```
void setup()
{
    Serial.begin(9600);

    pinMode( 8, OUTPUT); //Base pin.
    pinMode( 9, OUTPUT); //Shoulder pin.
    pinMode( 10, OUTPUT); //Elbow pin.

    BasePan.attach(8); // attaches the servo on pin 9 to the servo object
    ShoulderTilt.attach(9); // attaches the servo on pin 10 to the servo object
    ElbowTilt.attach(10); // attaches the servo on pin 8 to the servo object.

    //end setup.
}
```

Figure 7. Screenshots of sections of IK code



To find Base Pan servo object rotation.

Value can be \ominus or \oplus

$$\tan \theta_1 = \frac{y_2^0}{x_2^0}$$

$$\theta_1 = \tan^{-1}(y_2^0/x_2^0)$$

taking this and applying it to our problem:

we will use the x and z coordinates for remainder of our problem.

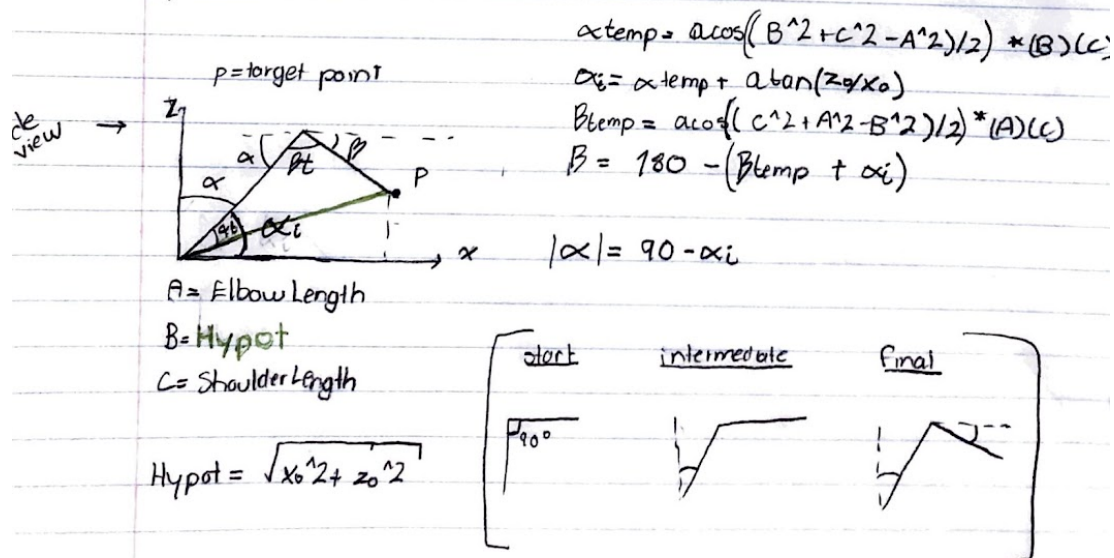


Figure 8. Image of IK mathematical concept

Output

```
Sketch uses 5048 bytes (15%) of program storage space. Maximum is 32256 bytes.  
Global variables use 261 bytes (12%) of dynamic memory, leaving 1787 bytes for local variables. Maximum is 2048 bytes.  
  
-----  
Compilation complete.
```

Figure 9. Screenshots of code compilation message

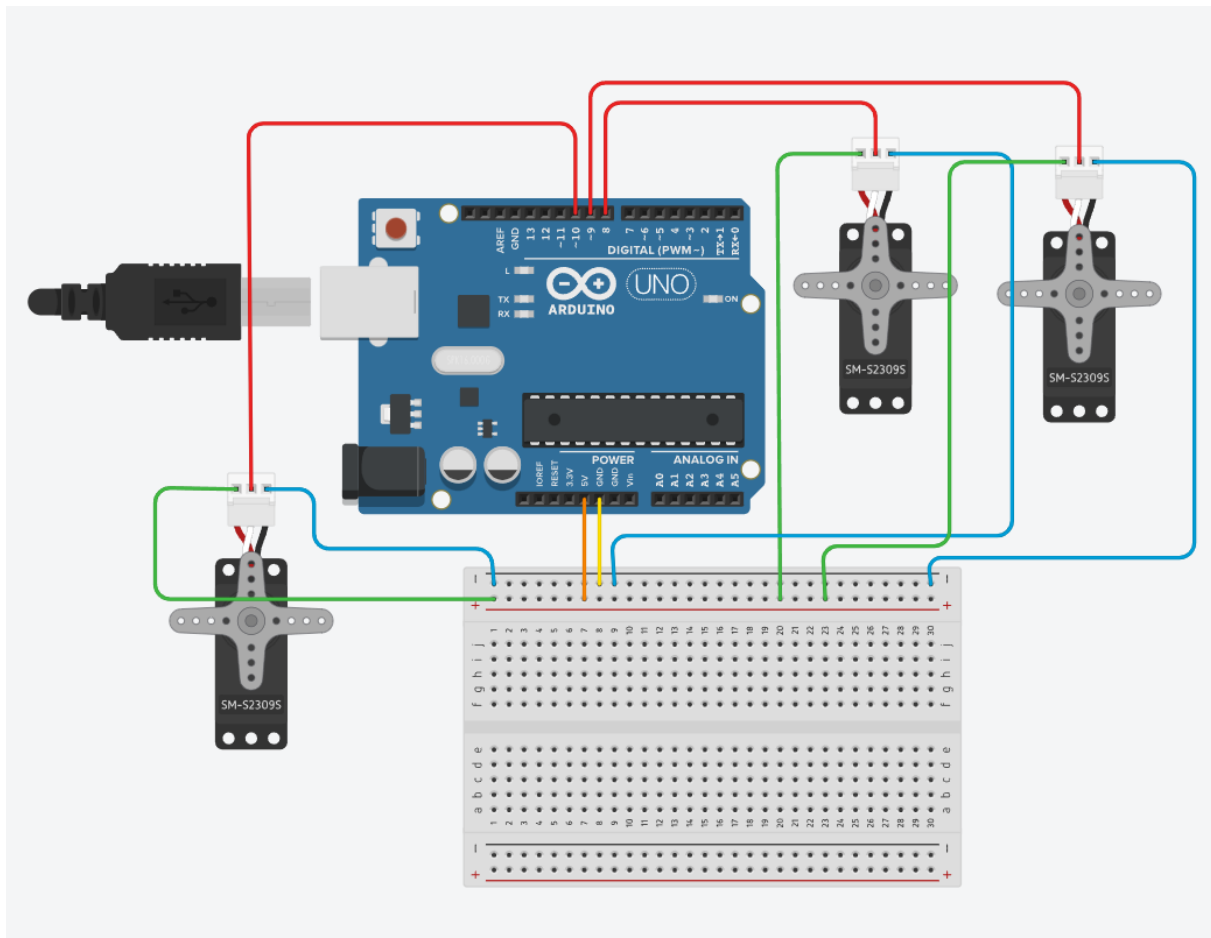


Figure 10. Screenshots of Servo motor connections

2.3.4 Code Files (OneDrive Links)

IK :

- [IK_CODE_palletizing.ino](#)
- [IK_CODE_original.ino](#)
- [IK_CODE_beta.ino](#)

2.4 User Interface code

This week's prototype was very different from last week, and we developed a much more complex and a more aesthetically pleasing experience for our GUI. Starting with a completely new base for our code discovered by following a tutorial online linked in the bibliography below, we have acquired knowledge of many new functions and deepened our minimal existing knowledge of the Python programming language. Using all of this, we have come up with the following code, which will be explained below.

```
>>> import tkinter as tk
... import imageio
... from PIL import ImageTk, Image
...
... HEIGHT = 500
... WIDTH = 600
...
... def emergency_stop():
...     print("Stopped")
...
... def startcamcode():
...     print("Code Running *Beep Boop Beep Boop*")
...
... def distance_wall(entry):
...     print("This is the distance in meters:", entry)
...
... def takepicture():
...     print("Say Cheese!")
...
... def open_cameraeffector():
...     for widgets in frame.winfo_children() and lower_frame.winfo_children():
...         widgets.destroy()
...     stop_button = tk.Button(frame, text="STOP", fg='white', font=20, bg='#EE0000', command=emergency_stop).place(relx=0, relwidth=0.49, relheight=1)
...     picturebutton = tk.Button(lower_frame, text="Take Picture", fg='white', font=20, bg='#F08080', command=takepicture).place(relx=0, rely=0.9, relwidth=1, relheight=0.1)
...     start_button = tk.Button(frame, text="START", fg='white', font=20, bg='#7CFC00', command=startcamcode).place(relx=0.5, relwidth=0.49, relheight=1)
...
... def open_wpeffector():
...     for widgets in frame.winfo_children() and lower_frame.winfo_children():
...         widgets.destroy()
...     stop_button = tk.Button(frame, text="STOP", fg='white', font=20, bg='#EE0000', command=emergency_stop).place(relx=0, relwidth=0.49, relheight=1)
...     start_button = tk.Button(frame, text="START", fg='white', font=20, bg='#7CFC00', command=startcamcode).place(relx=0.5, relwidth=0.49, relheight=1)
...
... root = tk.Tk()
... root.title('User Interface')
...
... canvas = tk.Canvas(root, height=HEIGHT, width=WIDTH)
... canvas.pack()
...
... background_image = ImageTk.PhotoImage(Image.open("C:\\Users\\isasb\\Pictures\\arm.png"))
... background_label = tk.Label(root, image=background_image)
... background_label.place(relwidth=1, relheight=1)
...
... frame = tk.Frame(root, bg='#80C1FF', bd=5)
... frame.place(relx=0.5, rely=0.1, relwidth=0.75, relheight=0.1, anchor='n')
...
... entry = tk.Entry(frame, font=40)
... entry.place(relwidth=0.49, relheight=1)
...
... button = tk.Button(frame, text="Input distance from wall", font=5, command=lambda: distance_wall(entry.get())).place(relx=0.5, relwidth=0.49, relheight=1)
...
... lower_frame = tk.Frame(root, bg='#80C1FF', bd=10)
... lower_frame.place(relx=0.5, rely=0.25, relwidth=0.75, relheight=0.6, anchor='n')
...
... cam_button = tk.Button(lower_frame, text="Camera End-Effector", fg='white', font=30, bg='#FF6A6A', command=open_cameraeffector).place(relx=0, rely=0, relwidth=1, relheight=0.49)
...
... wp_button = tk.Button(lower_frame, text="Water/Paint End-Effector", bg='#836FFF', fg='white', font=30, command=open_wpeffector).place(relx=0, rely=0.5, relwidth=1, relheight=0.49)
...
... root.mainloop()
```

Figure 11. Screenshot of the user interface code

2.4.1 Definitions of variables

The code begins by importing the necessary libraries to execute the code downloaded and installed beforehand. Once everything is set up, the code begins reading the defined variable, which in this case is "emergency_stop," which currently has the pressing of the stop button print "Stopped on the run code screen, "startcamcode" which prints "Code Running" when its associated button is pressed. We also have "distance_wall" which is the button pressed to print the entered distance from the wall in meters onto the code screen, as well as "takepicture" which prints "Say Cheese!" onto the same screen. These four functions will eventually be linked to all other codes, such as the inverse kinematics one for the distance function, the camera code and other Arduino codes for the things such as the interrupt function for the stop button and the video feed to be shown on the camera end-effector, but since we are still quite early on in the prototyping process, they have yet to be connected. Our final definitions are the end effector button controls, "open_cameraeffector" which clears the home screen. The addition of all the new functions necessary for the camera end effector such as the start, stop and take picture button that is mentioned previously and the "open_wpeffector" which has those same start and stop buttons. These definitions with built-in functions are the embellishments of this whole user interface experience and their functions, but the main code will be explained next.

2.4.2 Main Body code

The main code contains a mixture of labels, frames, buttons and an entry box, which, combined with all their conditions, create the user interface that you see when the code is running. The buttons are each assigned a command defined and explained in the previous paragraph. The labels and frames are coloured and filled with images to make the user an enjoyable and easy experience. When run, the code will pop up the following window, which contains the buttons and entry box of the home page.

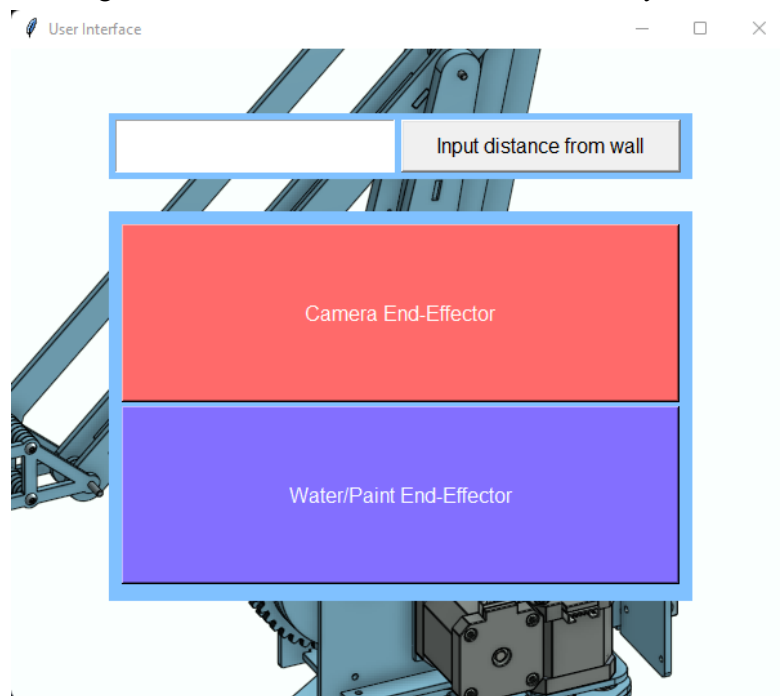


Figure 12. Compiled user interface code and image of user interface

Once the measurement is input in the entry box and the button beside it is pressed you will see the following.

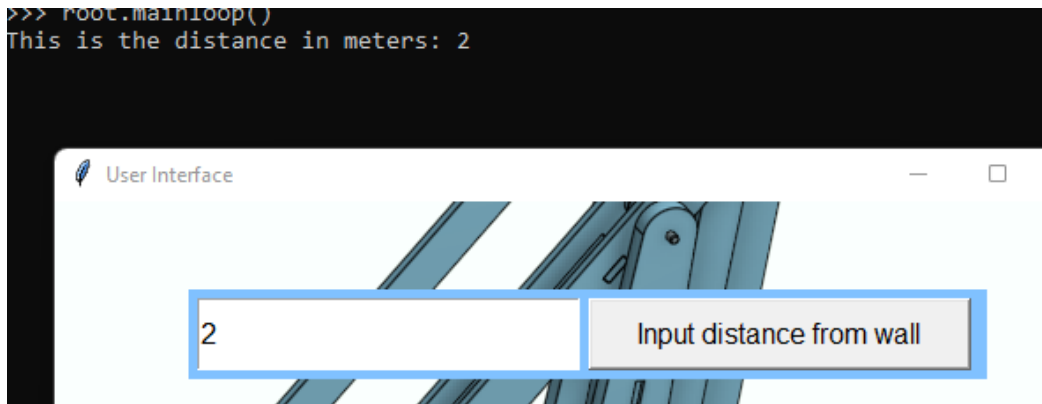


Figure 13. Functioning entry box to input the distance the robot is from its workspace

This means you can now pick your first end effector, if going in order you would start with the camera one which will look like so.

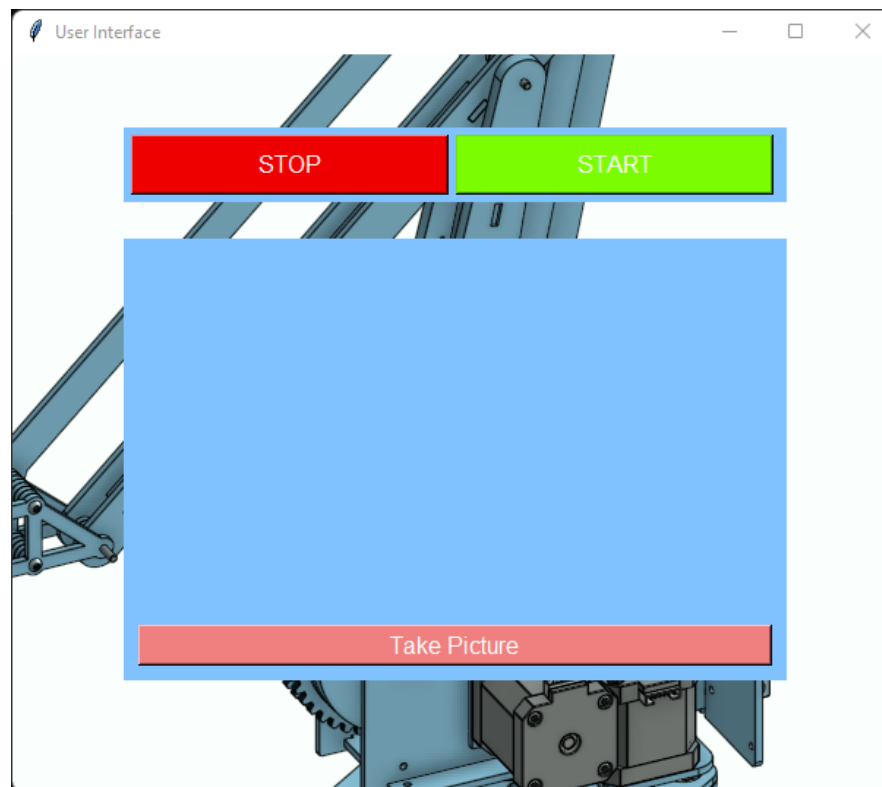
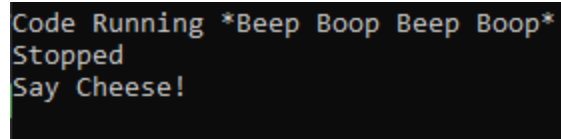


Figure 14. Camera end effector control page

The blue area will eventually be filled with the live feed from the camera end effector, but at the moment, we do not have those connected. As mentioned above, the start, stop and take picture buttons print text in the programming interface.



```
Code Running *Beep Boop Beep Boop*
Stopped
Say Cheese!
```

Figure 15. Programming Interface once “Take Picture” button is pressed

The final face of the user interface is the water/paint end-effector one which is used by pressing the water end effector button on the home screen. There will be some back button or a new button in the final product or the next prototype to select the second end effector from the first one since there is no easy way to do so. This is the water end effector interface which will most likely be modified in the future.

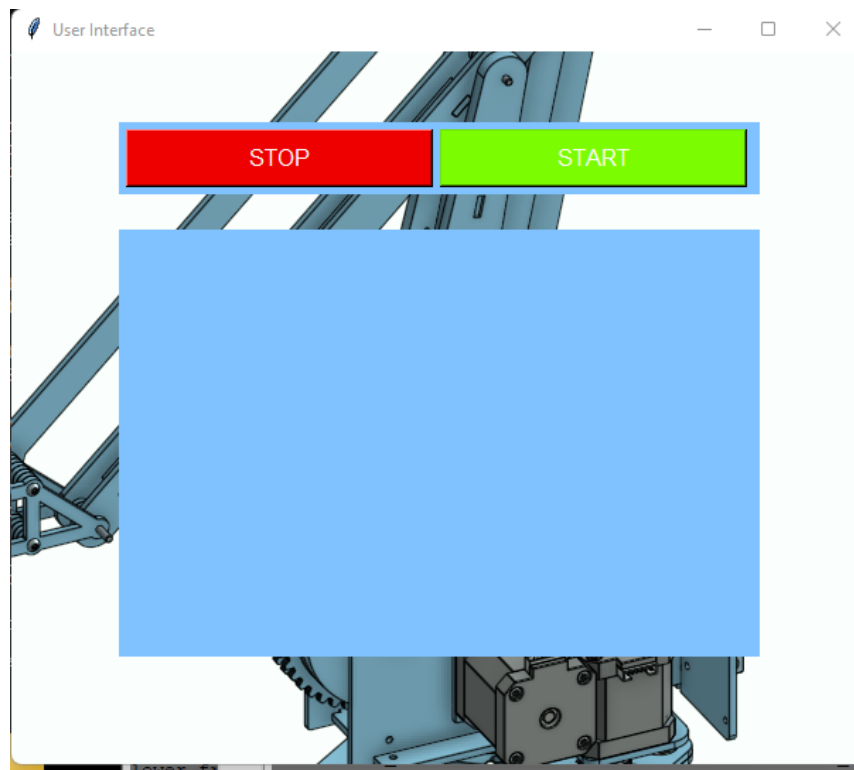


Figure 16. Water and Paint end effector control page

Similar to the camera end-effector, the water/paint end-effector has the start and stop buttons and will most likely have something in the currently empty box, but there is not as of this prototype.

In conclusion, the user interface for our project is still very much a work in progress but is moving along nicely and will soon be bringing everything together into a simple and easy-to-use GUI.

2.5 Safety

2.5.1 Emergency Stop

Our team will be implementing an emergency stop function within the user interface that will stop all robot operations in light of a malfunction of the robot. This function will be implemented within the user interface to ease use for those operating the robot. Considering that those operating the robot will

have limited technical knowledge, the design of the function will be in the corner of each page with large lettering to be visible to the user at all times.

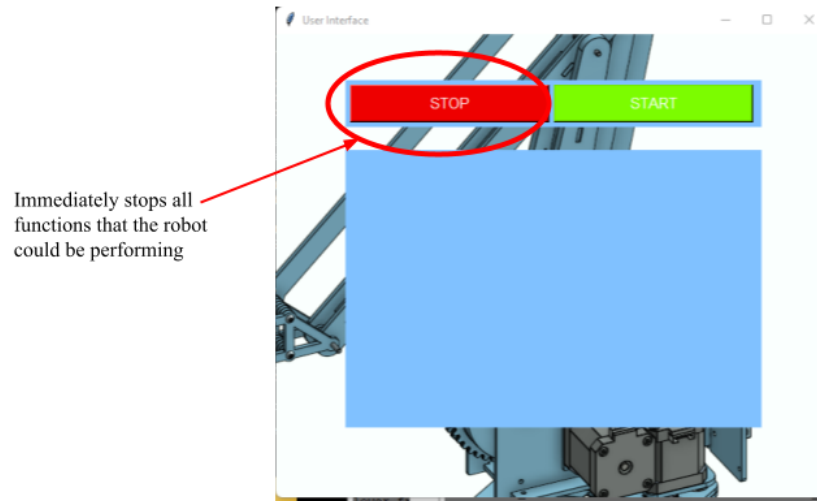


Figure 17. Emergency stop function within the user interface

2.5.2 Motion Detection Sensors

The motion detection sensors have been implemented and tested to detect human motion presence within 7 meters of the sensors. This will alert the passerby that the robot functions with a buzzer sound. The red led light on the breadboard also indicates when the sensors have detected motion. Once the sensors are triggered, an interrupt function on the arduino will pause the movement of the arm for a timed duration. Then, assuming the sensors are not re-triggered, the arm will resume the performing function.

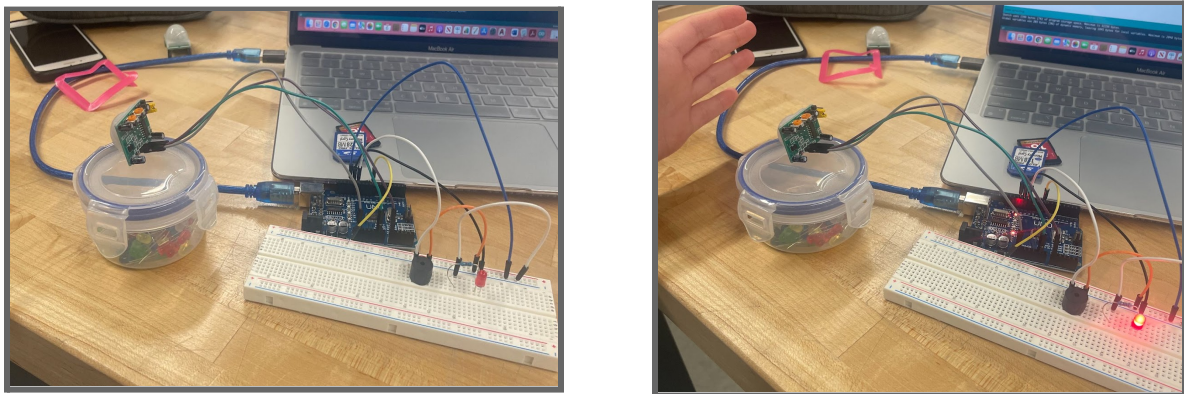


Figure 18. Motion detection sensors functioning

3. Prototype Test plans

3.1 Prototype II Test Plan

Table 1. Prototype II Test Plan with results

Test #	Objective	Description and Test Method	Test Duration and Date	Results
1	Test arm movement code on arm	The algorithm and code for the inverse kinematics movement of the arm should be completed, and it will be tested on the arm as soon as the opportunity presents itself so that any issues are discovered and it can be modified accordingly quickly	1 day First session with robot arm	Will be tested once the robot arm is completed March 14th, 2022
2	Paper or cardboard quick prototype	Quickly make a 2D and/or 3D tangible model of end-effectors as a size comparison to the actual robot and objects that will be used with them to be sure of our dimensions	> 1 day First session with robot arm	Will be tested once the robot arm is completed March 14th, 2022
3	Retouch engineering drawing and 3D modeling of end-effectors	Any miscalculations or wrong dimensions are discovered through the previous tests and now the drawings and models can be readjusted to accommodate our new discoveries	1 day After first session with robot	Will be completed one day after the first session with the robot, March 15th, 2022.
5	3D printed model of what we have designed so far	The 3D model is adjusted and can now have the pieces printed and assembled for testing on the robot. If the previous analysis and prototyping were effective, this should be done once or twice to minimize the number of materials used and the overall cost	1 or 2 days Second session with robot arm	Will be completed during second session with the robot arm, March 17th, 2022

3.2 Prototype III Test Plan

Table 2. Prototype III Test Plan with expected results

Test #	Objective	Description and Test Method	Test Duration and Date	Expected Results
1	Test code and user interface with newly 3D printed pieces and arm	Pieces are printed and the end-effectors are assembled, everything can be wired and plugged into the Arduino in its respective place, and the code can be tested on the arm and the user interface. If any errors occur, the code and user interface will have to be modified accordingly	Consistent with prototype testing. Five consecutive test trials with no errors.	1 or 2 days Once robot and 3D printer is accessible
2	Make sure attachments and necessary scenarios are compatible with end-effectors and code	The final test will entail putting all pieces together for one last test, running multiple scenarios with the user interface, arm and all the end effectors to simulate the users' experience and ensure that it is possible, simple and easy to understand for the high school students who will most likely be running the interface and interchanging the end effectors	Consistent prototype testing. Five consecutive test trials with no errors.	3 days Second last step, leave time to fix mistakes and get feedback
3	Adjust all necessary things and create the final versions of end-effectors, code and user interface	Once the group and client have settled on a final version and has been through the tests previously mentioned, it is time to bring it to life and create the final version of everything necessary, test it on the robot arm and if all goes well, there will no longer be any need for prototyping	Either run out of time or be satisfied with the final product before design day	1 to 3 days Last step, must be before design day
4	Test Corrosion detection program and fix bugs to properly detect corrosion	Currently the code struggles with assigning values according to whether there is corrosion or not in the picture. With the TA and other people we will try and fix the current issue	Either run out of time or until the error is fixed within the code	1-2 weeks Lots of files to go back and forth as well as midterms this week will cause delays

5	Camera pictures running through Corrosion Program	Now that the camera is complete we can save the pictures it takes into the input file for the corrosion code so it scans those pictures taken	Code needs to work properly to detect properly but we can test the input method now that the camera works and the code runs	1 day Only need to set up camera onto laptop with Corrosion detection program
---	---	---	---	--

4. Updated Bill of Materials

Table 3. Bill of materials with total product cost estimate

Item Name and Link	Quantity	Cost (\$)	Justification
Camera OV7670 VGA CMOS Camera	1	23.68	The camera chosen needs to be compatible with Arduino software in order to access the data (live video feed) and send it to other devices.
PIR motion sensors PIR motion sensors	1	14.99	These sensors will be added to different end-effectors to ensure safety while operation. (soldered)
3D printing materials		0.00	Since most of our end effector components will be 3D printed, we will be using the machines and materials provided in the Maker Lab.
Arduino kit and wires	1	0.00 (Free at Maker Lab)	The Arduino will be useful for the spray guns in order to connect the sensors and triggers to a specific output in our software. This kit includes a breadboard and some resistors in order
Soldering kit	1	0.00 (Free at Maker Lab)	Used to mend our wires together and solder them to our things like our camera and sensors to ensure that they will not be easy to break off or to fall apart simply by moving.
Total product cost (w/o taxes or shipping)		38.67	
Total product cost (including taxes and shipping)		41.16	

5. Target Specifications

Table 4. Target Specification of all aspects

Robot Mechanics	
Degrees of Freedom	3
Weight supported by end effector (kg)	1.00
Weight of robot (kg)	9.07
Camera end-effector	
Back and Main piece diameter (mm)	60.00
Back depth (mm)	2.90
Main piece depth (mm)	22.00
Clips length	10.00
Weight (kg)	0.024
Camera	
Weight (kg)	0.016
Camera Resolution	VGA 640 x 480 at 30 fps
Working Power	60mW/15fps
Pixel coverage	3.6um x 3.6um
Painter/corrosion remover end-effector	
Main piece dimensions (mm)	(49.00 x 37.40 x 35) + 2(9.0 x 4.0 x 4.0)
Adjustable piece dimensions (mm)	(44.198 m x 22.0 x 34.5) + 2(32.5 x 4.8 x 4.8)
Weight (kg)	0.046

Conclusion

In summary, we are working out all the kinks in our prototypes, but everything is moving forward, and things are getting better every day. We have had our fair share of difficulties with this prototype from our many failed attempts at end-effectors and multiple coding troubles. However, our plan is slowly but surely coming together, and we will be working just as hard on our following prototypes and the final product to produce the best product possible for our client.

Bibliography

Galli, K., 2019. *How to Program a GUI Application (with Python Tkinter)!*. [video] Available at: <<https://www.youtube.com/watch?v=D8-snVfekto>> [Accessed 13 March 2022].