CS 4850 Section 02 Fall 2023

SP-1 Robots

Corey Griffin

Michael Grunkemeyer

Sharon Perry

12/2/23

Website: https://robotssp1blocksortingmv.github.io/Robot_Code/

GitHub: https://github.com/RobotsSP1BlockSortingMV/Robot_Code

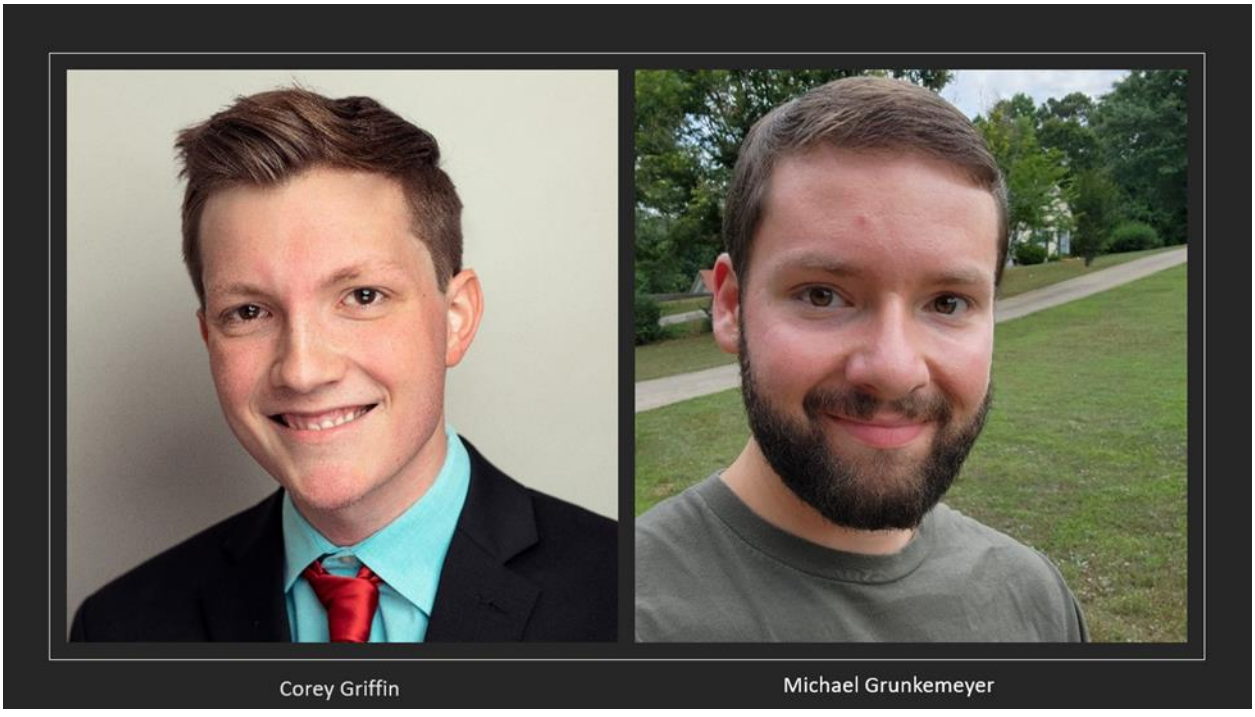


Table of Contents

Introduction

At the beginning of this project, we had many ambitious goals to achieve but ultimately, we decided that our goal was to have the robot identify and pick up a colored object with a fully customized hand with an electromagnet. Along the way, we faced many challenges with the robot, and we decided to also include an introductory guide to using the robot so that the next group would not face the same struggles that we did.

History of the Robots at Kennesaw State University

KSU for many years has had two UXA-90 robots. These were used in various projects years ago and then put away from sight in a storage room until they were relocated. The first group to do a project with the robots after rediscovering them was the RESTful Robots group. When this group chose the robot project, they quickly found out the onboard operating system was unable to be updated and inadequate to build new projects.

This group decided to use the REST API to create the RESTful API that would allow them to communicate and control the robots through a raspberry pie. This project was ultimately successful, and they ended up using a raspberry pie with a docker image flashed to it. Using the IP address of the raspberry pie they were able to command all basic actions along with individual motor requests.

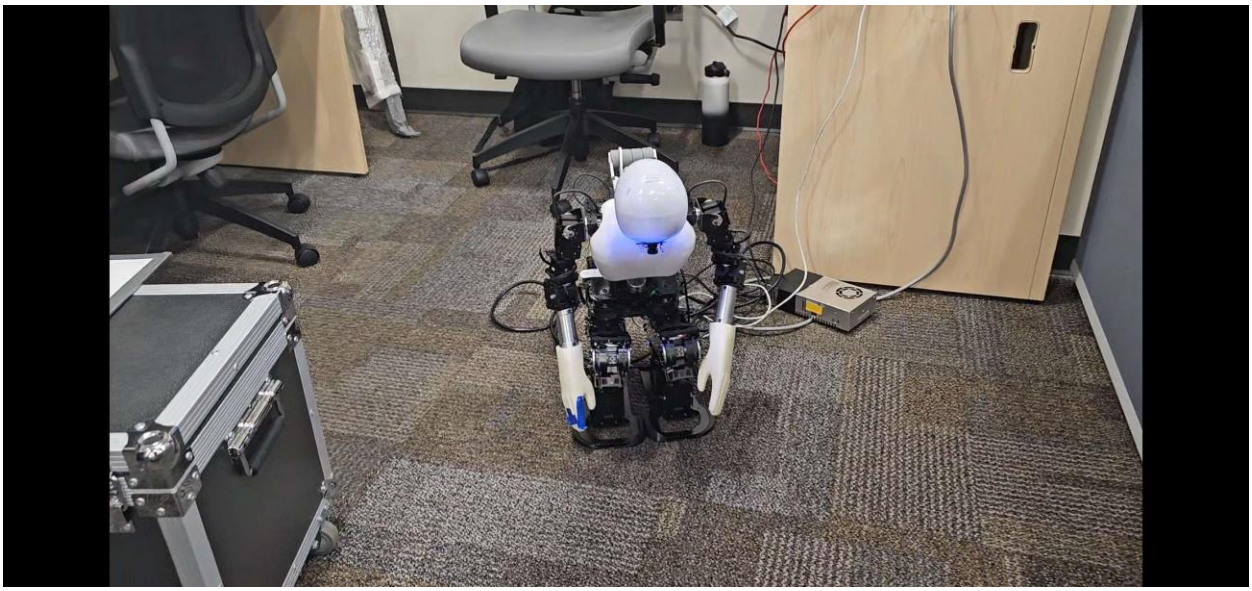
The next group to do a project with the UXA-90 robots was the VR movement group. This group utilized the RESTful robots API, unity, and an Oculus Rift VR headset to complete their project. They were able to make API calls with the VR headset controls and see the robot's vision through a camera connected to an additional raspberry pie through the VR headset. This group faced a lot of challenges in using the RESTful API as it was not properly documented.

Guide to Using the Robot

In this section, we will discuss the initial setup steps to get your robot project up and running. The first group to do a project with these robots as mentioned previously found that the original operating system was too difficult to upgrade and interface with. They ended up creating an API to interface with the robots called RESTful Robots API or REST API for short. See the previous group's documentation for more information. The previous group did a really good job in developing an API to interface with the robots however the documentation is sparse and leads to a lot of confusion in how to command the robot for other projects. Through trial and error, we have discovered how the robot works and we will save you a lot of time with this guide.

We filmed two videos, one discussing how to get the robot out of the box and another video walking through the steps of turning the robot on as well as commanding the robot. I will share the steps here in the text as well. The two videos will be listed on our project's website as "Remove the Robot From the Case Video" and "Initial Robot Setup Video".

The first thing you will need to do is open the robot's case. Professor Perry will provide you with the code needed to unlock the case. Remove all the materials on top of the robot and locate the two white cords. With two people preferably carefully lift the robot out of the case and place it on the floor. Next, pick up the robot and put the robot into the sitting position see the picture below:



Once in the sitting position, you can plug the power block into the robot. Locate the gray raspberry pie and plug the power cord into it. Locate the white mini-USB cable and plug the cable into the back of the robot underneath the gray handle. Plug the USB side into the gray raspberry pie. There are two power switches on the robot. One near the power cable and one underneath the robot's chest. Ensure these are flipped in the correct position.

Press the power button behind the robot's head to power the robot on. Once the robot is running press the power button on the gray raspberry pie and look for the red light on the front to indicate that it is powered on. You will need a keyboard and mouse to capture the IP address of

the gray raspberry pie. Once the gray raspberry pie is powered and fully loaded the robot will stand up by itself. Press enter on the keyboard attached to the gray raspberry pie. When you see the login text on the screen the username and password for the gray raspberry pie is username: uxa90 password uxa90. Once logged in look on the screen for two IP addresses.

The screen will display text like this:

Ubuntu 22.04.2 LTS uxa90 tty1 uxa90 login: [41.336739] cloud-init [1788]: Cloud-init v. 23.1.2-0ubuntu0~22.04.1 running 'modules:config [43.875159] cloud-init [1816]: Cloud-init v. 23.1.2-0ubuntu0~22.04.1 running 'modules:final' at Thu, 05 0 [44.296411] cloud-init [1816]: Cloud-init v. 23.1.2-0ubuntu0~22.04.1 finished at Thu, 05 Oct 2023 20:14:0 uxa90 login: uxa90 Password: Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-1027-raspi aarch64)

- Documentation: https://help.ubuntu.com/
- Management: https://landscape.canonical.com/
- Support: https://ubuntu.com/advantage

System information as of Thu Oct 5 16:04:08 EDT 2023 System load: 3.11865234375 Processes: Usage of /: 14.5% of 58.23GB 24% 0% Users logged in: 187 0 Memory usage: IPv4 address for dockero: 172.17.0.1 Swap usage:

IPv4 address for wlano: 10.101.142.174 Temperature: 53.6 C

- Strictly confined Kubernetes makes edge and IoT secure. Learn how Microk8s just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge Expanded Security Maintenance for Applications is not enabled. 433 updates can be applied immediately. 145 of these updates are standard security updates. To see these additional updates run: apt list --upgradable 24 additional security updates can be applied with ESM Apps. Learn more about enabling ESM Apps service at https://ubuntu.com/esm The list of available updates is more than a week old. To check for new updates run: sudo apt update Last login: Thu Oct 5 16:04:10 EDT 2023 on tty1 uxa90@uxa90: ~$ cd .ssh uxa90@uxa90:~/.ssh$ ls authorized_keys cloud_key cloud_key.pub id_ed25519 id_ed25519.pub known_hosts known hosts.old uxa90@uxa90:~/.ssh$ cat cloud_key.pub ssh-rsa AAAABNC1403500000

Notice the highlighted text above that says IPv4 address for wlano: 10.101.142.174

This is the IP address you will need to use in your code to interact with the gray raspberry pie to send commands to the robot. This IP address can change but sometimes it does not. There are other ways to get the IP address, but this is the simplest way.

Another thing to note is if you do not see wlano listed then reboot the Raspberry pie and try again sometimes it does not show on the first attempt.

If you go to our GitHub and look at the file Movement.py you will see several movement commands listed here.

```
141  def movement_command(select_command,position):
142      session = requests.Session()
143      client = session
144      base_address = '10.101.148.223'
145      baseIP = 'http://' + base_address
146      port = 50000
147      baseIP = 'http://' + base_address + ':'+str(port) + '/'
148      robot_head = baseIP + 'motor?id='
149      robot_arm = baseIP + 'motor?id='
150      robot_motion = baseIP + 'motion/'
151      if select_command == 'walk_left':
152          walk_left(robot_motion)
153      elif select_command == 'walk_right':
154          walk_right(robot_motion)
155      elif select_command == 'walk_forward_short':
156          walk_forward_short(robot_motion)
157      elif select_command == 'turn_right':
158          turn_right(robot_motion)
159      elif select_command == 'turn_left':
160          turn_left(robot_motion)
161      elif select_command == 'sit_down':
162          sit_down(robot_motion)
163      elif select_command == 'standing_position':
```

On line 144 set the base IP address to the one found on the gray raspberry pie. Do not change the port number, it must be port 50000.

To make API calls we use a method called async def APICall(ip_address).

```python
async def APICall(ip_address):
    try:
        # Make an asynchronous GET request using requests library
        response = await asyncio.to_thread(requests.get, ip_address)

        # Check if the response status code is in the 2xx range (indicating success)
        if 200 <= response.status_code < 300:
            # Read and return the response body as a string
            return response.text
        else:
            # Handle non-successful responses here if needed
            print(f"Request failed with status code: {response.status_code}")
            return None
    except Exception as e:
        # Handle exceptions here if needed
        print(f"An error occurred: {str(e)}")
        return None
```

This is in the Movement.py file as well.

Essentially what you are doing to command the robot is you need to make an asynchronous call to the IP address of the gray raspberry pie in the form of a string.

For example, to make the robot sit down we would use the string:

http://10.101.148.223:50000/sit_down

The command convention is like this 'http://' + IP_address + ':' + port_number + '/' + command.

With typical prebuilt commands like sit down, walking position, and standing position you can just make the asynchronous API call with the string and the robot will follow the command.

If you want to command individual motors you need to take one additional step.

```python
def move_left_upper_shoulder(robot_arm,position,robot_motion):
    movement = robot_arm + '12&position=' + position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")

```

This command for example to move the left upper shoulder you will need to modify the string a little bit.

Instead of using http://10.101.148.223:50000/sit_down

You would use http://10.101.148.223:50000/motor?id=12&position=50&torq=4.

The command convention is now 'http://' + base_address + ':'+str(port) + 'motor?id=' + '12&position=' + position + '&torq=4'

Each motor has its own motor id list below:

{"data":[{"id":0,"name":"left-foot","description":"foot tilt in/out","position":127,"min":115,"max":140,"default":127,"inverted":true},

{"id":1,"name":"right-foot","description":"foot tilt in/out","position":127,"min":110,"max":140,"default":127,"inverted":false},

{"id":2,"name":"left-heel","description":"foot up/down","position":127,"min":77,"max":175,"default":127,"inverted":true},

{"id":3,"name":"right-heel","description":"foot up/down","position":127,"min":78,"max":175,"default":127,"inverted":false},

{"id":4,"name":"left-knee","description":"shin front/back","position":204,"min":77,"max":204,"default":204,"inverted":true},

{"id":5,"name":"right-knee","description":"shin fron/back","position":50,"min":50,"max":180,"default":50,"inverted":false},

{"id":6,"name":"left-front-upper-leg","description":"","position":127,"min":0,"max":255,"default":127,"inverted":true},

{"id":7,"name":"right-front-upper-leg","description":"","position":127,"min":0,"max":255,"default":127,"inverted":false},

{"id":8,"name":"left-back-upper-leg","description":"","position":127,"min":0,"max":255,"default":127,"inverted":true},

{"id":9,"name":"right-back-upper-leg","description":"","position":127,"min":0,"max":255,"default":127,"inverted":false},

{"id":10,"name":"left-hip","description":"","position":127,"min":0,"max":255,"default":127,"inverted":true},

{"id":11,"name":"right-hip","description":"","position":127,"min":0,"max":255,"default":127,"inverted":false},

{"id":12,"name":"left-upper-shoulder","description":"","position":180,"min":10,"max":254,"default":180,"inverted":true},

{"id":13,"name":"right-upper-shoulder","description":"","position":180,"min":1,"max":254,"default":180,"inverted":false},

{"id":14,"name":"left-lower-shoulder","description":"","position":115,"min":135,"max":254,"default":135,"inverted":false},

{"id":15,"name":"right-lower-shoulder","description":"","position":138,"min":1,"max":120,"default":120,"inverted":true},

{"id":16,"name":"left-bicep","description":"twist in","position":127,"min":1,"max":254,"default":127,"inverted":true},

{"id":17,"name":"right-arm","description":"twist in","position":127,"min":1,"max":254,"default":127,"inverted":false},

{"id":18,"name":"left-eblow","description":"max is up","position":210,"min":75,"max":210,"default":210,"inverted":true},

{"id":19,"name":"right-elbow","description":"min is up","position":40,"min":40,"max":180,"default":40,"inverted":false},

{"id":22,"name":"hip","description":"rotate left/right","position":127,"min":95,"max":165,"default":127,"inverted":false},

{"id":23,"name":"neck","description":"look left/right","position":127,"min":1,"max":254,"default":127,"inverted":false},

{"id":24,"name":"head","description":"look up/down","position":127,"min":100,"max":160,"default":127,"inverted":false}],"message":"getAllMotors"}

Each motor also has its own range of motion, and you can apply a torque setting for how forcibly the robot does this action. When you make a call to an individual motor if you are using our methods, it will print out the current position of the motor and its range of motion, better known as the min and max value of that motor. Some of the motors are inverted as well so be aware of that.

One other important point is before you make an individual motor call make sure the robot is in the walking position. Also, ensure that you make an API call with the convention of:

http://10.101.148.223:50000/pc_control

Before making the actual individual motor request command.

To command the individual motor, make sure you follow this convention:

response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))

response_body = asyncio.run(APICall(movement))

Commanding this address first http://10.101.148.223:50000/pc_control

Which tells the robot you would like to access an individual motor.

Then you can make the individual motor call next with the string

http://10.101.148.223:50000/motor?id=12&position=50&torq=4

If you do not follow this convention the robot will not move.

You are not required to use our Python code. This can be done in any programming language where you can make asynchronous API calls. For simplicity, you can use the code we have created in movement.py to develop new projects with Python if you do not want to duplicate the effort we have already put in.

One other thing to note. The robot named Clank's mini-USB connector popped off during testing as Clank's knee always overheats after 15 minutes causing the robot to fall to the floor. We resoldered the connector to the robot but it is not flat enough so we could not make a connection with that robot. I would suggest using a soldering iron to remove the mini-USB connector and try to replace it again so that both robots will be fully functional. This concludes the introductory guide to using the robot.

Sorting Colored Blocks with a UXA-90 Robot

Movement.py file

As previously stated in the introduction our goals were very ambitious at the beginning of the project ultimately, we reduced the complexity down to the goal of not sorting colored blocks with the UXA-90 robot but just picking up a colored object with a fully customized hand that features an electromagnet embedded inside of the hand.

The first thing we needed to do was write code to command the robot to perform different actions such as standing, walking, or moving individual motors so that we could move the arm toward an object we would like to pick up. In the first half of the semester through trial and error, we developed the movement.py Python file. This file includes all the initial settings and methods needed to send API calls to the gray raspberry pie which will in turn command the robot to perform such actions. If you want to see the details of how these methods work, please see the Guide to Using the Robot section at the top of this document.

Now that we can command the robot to perform any action required, we began the development of the minimize distance algorithm which will be used to move the hand toward a colored object and pick up said object.

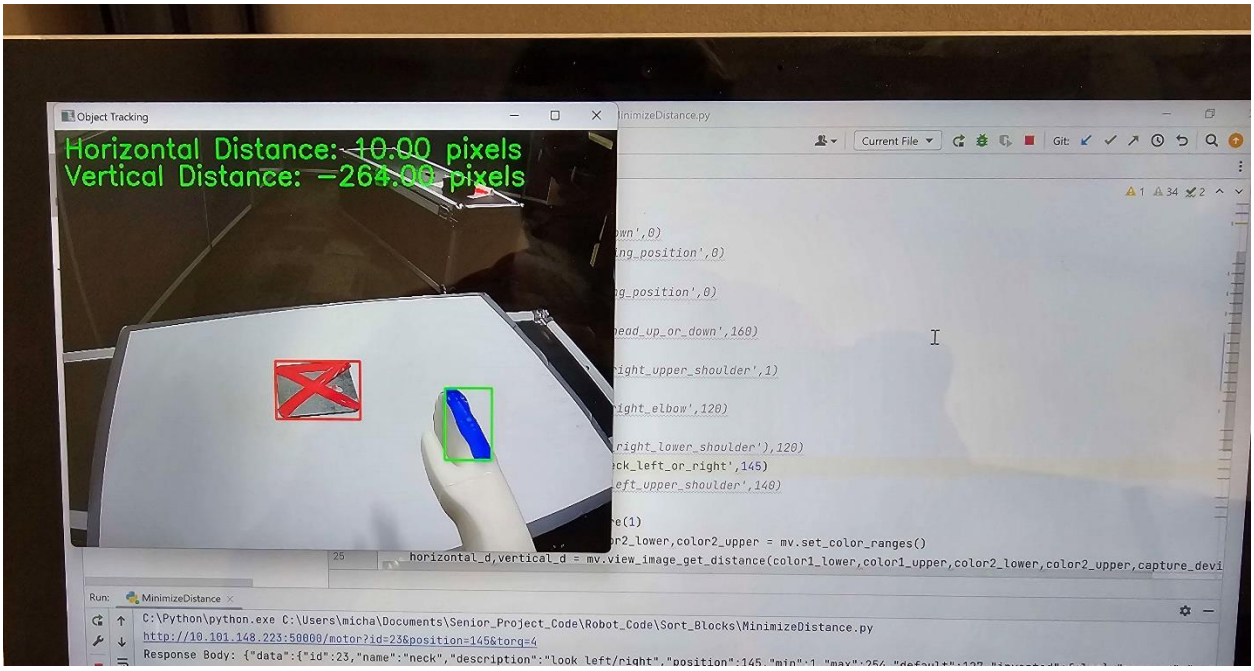
MinimizeDistance.py

The minimize distance algorithm depends on the Movement.py file, MachineVision.py file, and Relay.py file. We will discuss the MachineVision.py file and the Relay.py file here as movement.py is discussed in the Guide to Using the Robot section.

The minimize distance algorithm works by calculating the horizontal and vertical pixel distance between the hand and the object we want to pick up. In the MachineVision.py file, we used a Python library called open CV and selected two colors red and blue. The red color would represent the object we wanted to pick up and the blue color would represent the hand. This method shows a live video feed of what the robot could see through a camera we purchased on Amazon. See the details of the camera below:

We ended up not being able to use the robot's onboard camera as there was no way to connect to it. Being able to plug the camera directly into the laptop running the algorithm greatly simplified this step.

When running the code in MachineVision.py this is what you will see on your laptop screen:



With this code, we were able to track the robot's hand and the object that we wanted to pick up. You can see the horizontal and vertical pixel distance displayed on the screen as well. This helped in developing the minimize distance algorithm as we had live video of the robot's vision.

Now that the robot has vision, we began creating the minimize distance algorithm. The machine vision algorithm would run 300 iterations and then return the current horizontal and vertical pixel distance between the object and the hand. This algorithm was tracking anything red or blue so we needed to ensure there were no objects in view of the robot that the algorithm would track otherwise the distance metrics would become skewed. Running 300 iterations before sending back the distance metrics greatly helped to get accurate figures for the minimize distance algorithm.

The minimize distance algorithm has a method to get the robot into the initial position to pick up objects. This method makes the robot stand up, go into walking position, move the head down, move the right upper shoulder, right elbow, right lower shoulder, and neck into the correct position to track the hand and the object we want to pick up.

Then with a loop, we begin capturing the distance between the hand and the object. Depending on the location of the object the robot will move its hand up or down and left or right. Once the hand is close enough to the object to be picked up the movement part of the algorithm will end. In Relay.py we have Python code to interact with a USB relay.

We purchased the USB relay in question from Amazon:

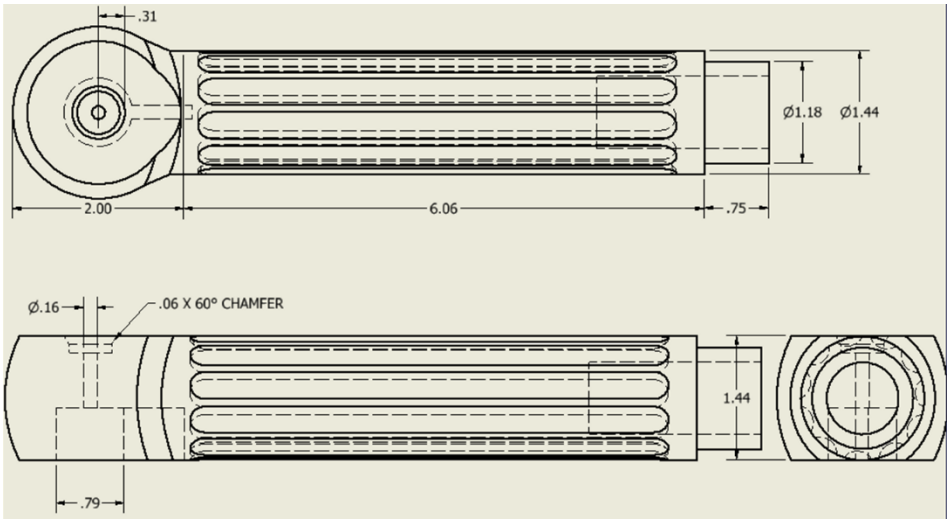
The code will command this relay to go into the closed position which will activate the electromagnet in the customized hand. Please see the next section for more details on the customized hand.

Once the magnet is active, if the robot has correctly placed its hand over the object the magnet will pick up the object. The Relay.py code will then command the robot to move its hand in another direction to show that the object is currently being held by the robot. Afterwards, the Relay.py code will open the circuit thus dropping the object thus ending the algorithm.

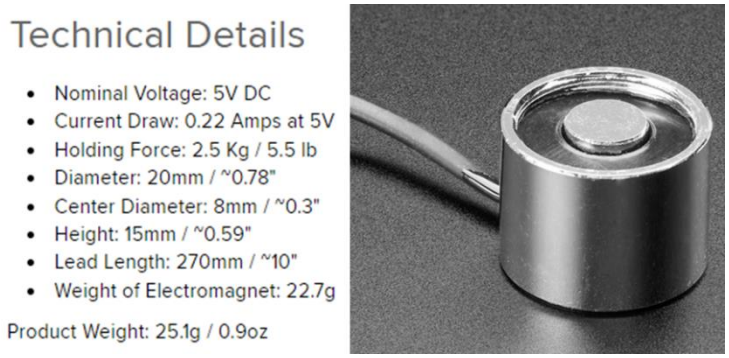
Keep in mind that the USB relay will retain the state that it is last set to. If the algorithm crashed after closing the circuit the magnet would remain powered and active which could cause issues if left unattended.

Hand Replacement Augmentation and Documentation

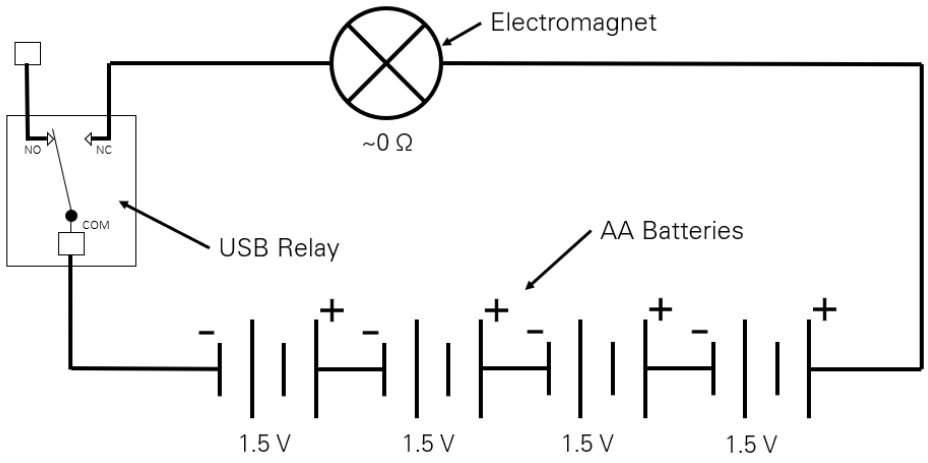
The provided hand for the UXA-90 was a rigid 3D-printed model of a hand in a slightly cupped position. As this provided an inopportune area of contact, we decided to design a new hand replacement that could be contoured to meet any specifications we might need. The replacement was designed in Autodesk Inventor based on caliper measurements taken of the original hand insert, which the model is provided on our website. If any future team needs to adapt our design of the new arm or modify it, the Autodesk Inventor software is free for KSU students by linking their Autodesk account to their KSU account. This new hand was given a longer length to grant the robot a further reaching distance, and an inset for the end of the hand to house an electromagnet and its mounting requirements. The electromagnet can imitate gripping objects in a similar manner to a crane. The hand was 3D printed with the thermoplastic polymer, Poly-Lactic Acid (PLA), by the KSU 3D Center; this material will degrade over time as it is biodegradable but should last long in its current storage environment.



The electromagnet used in the arm works by providing an electric current through a coil of wire wrapped around a ferromagnetic material such as iron. When powered, the electromagnet is theoretically able to exert a holding force of 5.5 pounds, although this level of performance would only be possible under near-perfect conditions and in practice should be expected to reliably lift with a force of ~2 pounds. It's worth mentioning that since electromagnets innately have such a low degree of electrical resistance, it is unadvisable to leave the electromagnet consistently active for periods of time longer than a minute or two, or else the magnet risks overheating.



The circuit used to power the electromagnet was connected to a USB relay (as mentioned in MinimizeDistance.py) to control turning the circuit off/on. As a power source, we used 4 AA batteries connected in series to provide ~6 volts to power the ~ 5 volt electromagnet.



Version Control

We created a new GitHub account to collaborate on this project. Our GitHub has all the code required to reproduce this project as well as the HTML code used to create our website. All team members utilized GitHub during the development of this project.

Challenges

There were several challenges that we faced during this project. The main issue that has been discussed was the previous group's documentation did not feature much about how to use the API to command the robot. The first group did a great job creating the API for the robot, but it was extremely complicated, and they left no instructions for us to get started. The next group faced the same issues as we did and left some vague instructions on how they controlled the robot but ultimately it greatly delayed our project.

Some of the missing artifacts included:

- The initial setup of the robot
- How to get the IP address of the gray raspberry pie
- How to make asynchronous API calls to command the robot
- How to make individual motor requests to the robot
- The login information for the gray raspberry pie

Additionally, having such a large element of our project being tied to a physical system limited our windows of time to work on the robot and test it. In a similar vein, we encountered some logistical problems coordinating with KSU's 3D Center. The Center was undergoing renovations and didn't reopen until November 1st. After which the arm took an estimated time of a week before it would be ready (this accounts for both printing and our team's physical alterations onto the hand to make it compatible with the robot's socket). This narrow window left us without time to make changes to the arm's structural design since we'd lack the time to reprint it. Such was a problem when we discovered that the weight of the new hand exceeded what the arm's motors were able to lift, as a result, we had to alter the testing to limit the arm's movement to not rely on lifting the arm above its current position.

Conclusion

Our project, even in reduced scope, was ultimately successful. We faced many challenges and pitfalls that previous groups also encountered and solved in manners unique to their projects. Not only did we finish our project, but we also documented the procedure required to get the robot in a ready state for future groups. This way as they develop projects with the UXA-90s, they won't have to sacrifice valuable time unraveling and rediscovering how to work with the robot. As a result of the time they'll save, they'll be capable of expanding the scope of their projects beyond our current limits and consequentially create more elaborate projects.

We developed an invaluable resource for future groups in the form of a Python file called movement.py that has all the required groundwork laid out to command the robot to do any action it is capable of, whether that be a complex hardwired animation like "DANCE" or a call to a particular single motor. With the ability to issue instructions for the robot, to follow, we created the MinimizeDistance.py algorithm which uses a front-mounted camera to track the robot's hand and target object we aim for it to pick up. This algorithm moves the augmented appendage towards the object in incremental amounts, activates the electromagnet when overtop the target, and then moves the hand whilst still gripping the object. While our demonstration may be somewhat brief, it serves as proof that our concept works and can be refined to perform more elaborate operations.

Code Used in this Project

Machine Vision.py

```
import cv2
import numpy as np
def set_capture(capture_device):
    # Initialize video capture from the camera
    cap = cv2.VideoCapture(capture_device)
    return cap

def set_color_ranges():
    # Convert RGB color (20, 0, 255) to HSV
    rgb_color1 = np.uint8([[[20, 0, 255]]])  # RGB color
    hsv_color1 = cv2.cvtColor(rgb_color1, cv2.COLOR_BGR2HSV)

    # Define a threshold range based on the HSV color
    color1_lower = np.array([hsv_color1[0][0][0] - 10, 100, 100])  # Adjust the -
10 to fit your desired range
    color1_upper = np.array([hsv_color1[0][0][0] + 10, 255, 255])  # Adjust the
+10 to fit your desired range
    color2_lower = np.array([100, 100, 100])
    color2_upper = np.array([120, 255, 255])
    return color1_lower,color1_upper,color2_lower,color2_upper

def
view_image_get_distance(color1_lower,color1_upper,color2_lower,color2_upper,cap):
    count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Convert the frame to HSV color space
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Create masks for each color range
        mask1 = cv2.inRange(hsv, color1_lower, color1_upper)
        mask2 = cv2.inRange(hsv, color2_lower, color2_upper)

        # Find contours in the masks
        contours1, _ = cv2.findContours(mask1, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        contours2, _ = cv2.findContours(mask2, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
        # Draw bounding boxes and calculate centroids
        for contour in contours1:
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
            centroid1 = (x + w // 2, y + h // 2)

        for contour in contours2:
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            centroid2 = (x + w // 2, y + h // 2)

        # Calculate distance between centroids
        if 'centroid1' in locals() and 'centroid2' in locals():
            horizontal_distance = centroid1[0] - centroid2[0]
            vertical_distance = centroid1[1] - centroid2[1]
            cv2.putText(frame, f'Horizontal Distance: {horizontal_distance:.2f}
pixels', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            cv2.putText(frame, f'Vertical Distance: {vertical_distance:.2f}
pixels', (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            if count == 300:
                return horizontal_distance, vertical_distance

        # Show the frame
        cv2.imshow('Object Tracking', frame)
        count += 1
        if count > 1000:
            return 10000000,10000000

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the video capture and close all windows
    cap.release()
    cv2.destroyAllWindows()
```

MinimizeDistance.py

```
import MachineVision as mv
import Movement as rbaction
import time
import Relay as ry
horizontal_d = 10000000
vertical_d = 10000000

right_elbow_current = 120
```

```
right_lower_shoulder_current = 120

#Move arm into inital position
#rbaction.movement_command('sit_down',0)
def initial_position():
    rbaction.movement_command('standing_position',0)
    time.sleep(15)
    rbaction.movement_command('walking_position',0)
    time.sleep(15)
    rbaction.movement_command('move_head_up_or_down',160)
    time.sleep(8)
    rbaction.movement_command('move_right_upper_shoulder', 1)
    time.sleep(8)
    rbaction.movement_command('move_right_elbow',110)
    time.sleep(8)
    rbaction.movement_command(('move_right_lower_shoulder'),120)
    time.sleep(3)
    rbaction.movement_command('move_neck_left_or_right',145)

initial_position()

#Change the outerloop depending on location of item to pickup
for i in range(0,2):
    capture_device = mv.set_capture(2)
    color1_lower,color1_upper,color2_lower,color2_upper = mv.set_color_ranges()
    horizontal_d,vertical_d =
mv.view_image_get_distance(color1_lower,color1_upper,color2_lower,color2_upper,ca
pture_device)
    print(horizontal_d,vertical_d)
    if vertical_d > 0:
        right_elbow_current -= 10
        rbaction.movement_command('move_right_elbow',(right_elbow_current))
        time.sleep(3)
    elif vertical_d < 0:
        right_elbow_current += 10
        rbaction.movement_command('move_right_elbow',(right_elbow_current))
        time.sleep(3)
    if horizontal_d > 0:
        right_lower_shoulder_current -= 10
        rbaction.movement_command('move_right_lower_shoulder',right_lower_shoulde
r_current)
        time.sleep(3)
    elif horizontal_d < 0:
        right_lower_shoulder_current += 10
```

```
        rbaction.movement_command('move_right_lower_shoulder',
right_lower_shoulder_current)
        time.sleep(3)

ry.grab_object()
```

Movement.py

```
import requests
import asyncio
from urllib.parse import urlparse

def get_ports(http_url):
    # Example HTTP URL
    # Parse the URL
    parsed_url = urlparse(http_url)
    # Get the port number (defaulting to 80 if not specified)
    port_num = parsed_url.port or 80
    # Print the port number
    return port_num

async def APICall(ip_address):
    try:
        # Make an asynchronous GET request using requests library
        response = await asyncio.to_thread(requests.get, ip_address)

        # Check if the response status code is in the 2xx range (indicating
success)
        if 200 <= response.status_code < 300:
            # Read and return the response body as a string
            return response.text
        else:
            # Handle non-successful responses here if needed
            print(f"Request failed with status code: {response.status_code}")
            return None
    except Exception as e:
        # Handle exceptions here if needed
        print(f"An error occurred: {str(e)}")
        return None

def walk_left(robot_motion):
    response_body = asyncio.run(APICall(robot_motion + 'walk_left'))
    if response_body:
        print(f"Response Body: {response_body}")
```

```
def walk_right(robot_motion):
    response_body = asyncio.run(APICall(robot_motion + 'walk_right'))
    if response_body:
        print(f"Response Body: {response_body}")

def walk_forward_short(robot_motion):
    response_body = asyncio.run(APICall(robot_motion + 'walk_forward_short'))
    if response_body:
        print(f"Response Body: {response_body}")

def turn_right(robot_motion):
    response_body = asyncio.run(APICall(robot_motion + 'turn_right'))
    if response_body:
        print(f"Response Body: {response_body}")

def turn_left(robot_motion):
    response_body = asyncio.run(APICall(robot_motion + 'turn_left'))
    if response_body:
        print(f"Response Body: {response_body}")

def sit_down(robot_motion):
    response_body = asyncio.run(APICall(robot_motion + 'sit_down'))
    if response_body:
        print(f"Response Body: {response_body}")

def standing_position(robot_motion):
    print(robot_motion + 'reset')
    response_body = asyncio.run(APICall(robot_motion + 'reset'))
    if response_body:
        print(f"Response Body: {response_body}")

def walking_position(robot_motion):
    print(robot_motion + 'basic_motion')
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(robot_motion + 'basic_motion'))
    if response_body:
        print(f"Response Body: {response_body}")

#Left upper shoulder - id:12 ; min: 10 ; max:254 ; default:180 ; inverted:true ;
min is slightly behind the user, max is straight up
#Right upper shoulder - id:13 ; min: 10 ; max:254 ; default:180 ; inverted:false
; min is straight up, max is
#Left lower shoulder - id:14 ; min:135 ; max:254 ; default:135 ; inverted:false
#Right lower shoulder - id:15 ; min:1 ; max:120 ; default:120 ; inverted:true
```

```
def move_left_upper_shoulder(robot_arm,position,robot_motion):
    movement = robot_arm + '12&position=' + position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")

def move_right_upper_shoulder(robot_arm,position,robot_motion):
    movement = robot_arm + '13&position=' + position + '&torq=4'
    print(movement)
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")

def move_left_lower_shoulder(robot_arm,position,robot_motion):
    movement = robot_arm + '14&position=' + position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")

def move_right_elbow(robot_arm,position,robot_motion):
    movement = robot_arm + '19&position=' + position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))


def move_right_lower_shoulder(robot_arm,position,robot_motion):
    movement = robot_arm + '15&position=' + position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")

def move_neck_left_or_right(robot_head,position,robot_motion):
    movement = robot_head + '23&position='+ position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")
```

```
def move_head_up_or_down(robot_head,position,robot_motion):
    movement = robot_head + '24&position=' + position + '&torq=4'
    print(movement)
    response_body1 = asyncio.run(APICall(robot_motion + 'pc_control'))
    response_body = asyncio.run(APICall(movement))
    if response_body:
        print(f"Response Body: {response_body}")

#use ip address from wlano0
#Boot up gray rasberry pie wait for it to say cloud login
#Press enter a few times then login when it says UXA90 login
#Name,Password is uxa90,uxa90
#IP address will be displayed in the upper right hand corner from the device
wlano0
#Append the IP address as I am below to make API calls to command the robot

def movement_command(select_command,position):
    session = requests.Session()
    client = session
    base_address = '10.101.148.223'
    baseIP = 'http://' + base_address
    port = 50000
    baseIP = 'http://' + base_address + ':'+str(port) + '/'
    robot_head = baseIP + 'motor?id='
    robot_arm = baseIP + 'motor?id='
    robot_motion = baseIP + 'motion/'
    if select_command == 'walk_left':
        walk_left(robot_motion)
    elif select_command == 'walk_right':
        walk_right(robot_motion)
    elif select_command == 'walk_forward_short':
        walk_forward_short(robot_motion)
    elif select_command == 'turn_right':
        turn_right(robot_motion)
    elif select_command == 'turn_left':
        turn_left(robot_motion)
    elif select_command == 'sit_down':
        sit_down(robot_motion)
    elif select_command == 'standing_position':
        standing_position(robot_motion)
    elif select_command == 'walking_position':
        walking_position(robot_motion)
    elif select_command == 'move_left_upper_shoulder':
        move_left_upper_shoulder(robot_arm,str(position),robot_motion)
```

```
    elif select_command == 'move_right_upper_shoulder':
        move_right_upper_shoulder(robot_arm,str(position),robot_motion)
    elif select_command == 'move_left_lower_shoulder':
        move_left_lower_shoulder(robot_arm,str(position),robot_motion)
    elif select_command == 'move_right_lower_shoulder':
        move_right_lower_shoulder(robot_arm,str(position),robot_motion)
    elif select_command == 'move_neck_left_or_right':
        move_neck_left_or_right(robot_head,str(position),robot_motion)
    elif select_command == 'move_head_up_or_down':
        move_head_up_or_down(robot_head,str(position),robot_motion)
    elif select_command == 'move_right_elbow':
        move_right_elbow(robot_arm,str(position),robot_motion)
```

Relay.py

```
import serial

from time import sleep
import Movement as rbaction

#Method to activate the magnet and grab the object
def grab_object():
    ser = serial.Serial(port='com5', baudrate=9600)
    ser.close()
    ser.open()
    ser.write(bytes.fromhex("A0 01 01 A2"))
    print('Circuit closed. Button is pressed')
    sleep(3)
    rbaction.movement_command('move_right_lower_shoulder',120)
    sleep(10)
    ser.write(bytes.fromhex("A0 01 00 A1"))
    print('Circuit Open. Button is released')

#Backup method to turn the magnet off if needed
def drop_object():
    ser = serial.Serial(port='com5', baudrate=9600)
    ser.close()
    ser.open()
    ser.write(bytes.fromhex("A0 01 00 A1"))
    print('Circuit Open. Button is released')
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title> SP-1 Robots Block Sorting Project </title>
<meta charset="utf-8">
<link rel="stylesheet" href="mystyles.css">
<meta name="robots" content="noindex, no follow, noarchive" />
<meta name="author" content="Michael Grunkemeyer & Corey Griffin" />
</head>
<body>
<div id="wrapper">
<header>

  <!-- official KSU colors gold #FFC425 and black #231F20 -->

<p align="center"> <br>
<img src="Images/Robot.jpg" alt="KSU" width="473">
<p align="center">
  <h1> SP-1 Robots Block Sorting Project</h1>
  <p>CS 4850 Senior Project Fall 2023</p>
  <br>
</header>
<nav>


<ul>
  <li> <u>Useful Resources </u> </li><br>
  <li><a
href="https://github.com/RobotsSP1BlockSortingMV/Robot_Code">Github</a></li><br>
  <li><a href="https://www.manualslib.com/manual/1507985/Robobuilder-Uxa-90-
Light.html">UXA-90 Manual</a></li><br>
  <li><a href="https://www.youtube.com/watch?v=uL0eQwdKisg">Remove the Robot From
the Case Video</a></li><p></p>
  <li><a href="https://www.youtube.com/watch?v=E7NZNTmAru4">Initial Robot Setup
Video</a></li><p></p>
  <li><a href="Report/Robots-SP1-FinalReport.pdf" download>Final
Report</a></li><p></p>
  <li><a href= "Arm/Arm CAD mod.stl" download>Arm CAD File</a></li><p></p>
  <li><a href= "https://www.youtube.com/watch?v=-sRnRLmcA1s" > Presentation
Demo</a></li><p></p>
  <li><a href="https://4850-red.github.io/red-site/">Restful Robots
Group</a></li><br>
  <li><a href="https://cs4850-uxa90.github.io/CS4850-Spring23-Robot/">VR
Group</a></li><p>
```

```
  <!-- <li><a href = "mailto: abc@example.com">Email me.</a></li><br>  -->
  <br><p>

</ul>

</nav>
<main>

  <div class="container">
    <div class="grid-container">
      <div class="picture">
        <p align="center">
          <img src="Images/Team_mem.jpg" alt="Robitic KSU" width="473">
          <br>
      </div>
      <div class="Author">
        <h2>Team members: Michael Grunkemeyer & Corey Griffin</h2>
      </div>
      <p>
      <div class="Summary">             
              
      By working with the UXA-90 robot, our team has documented how to connect
and program it, as we display an example of such possible future projects by
having the robot identify and pick up a colored object. <p>
  <br>
  <br>
  <br>
  </div>
</div>
</div>
</main>

<footer>Copyright &copy; 2023 Kennesaw State University<br>

<br>
</footer>
</div>
</body>
</html>
```