Group 8

CSCE 462

Tanner Gao, William Yang, Justin Yap

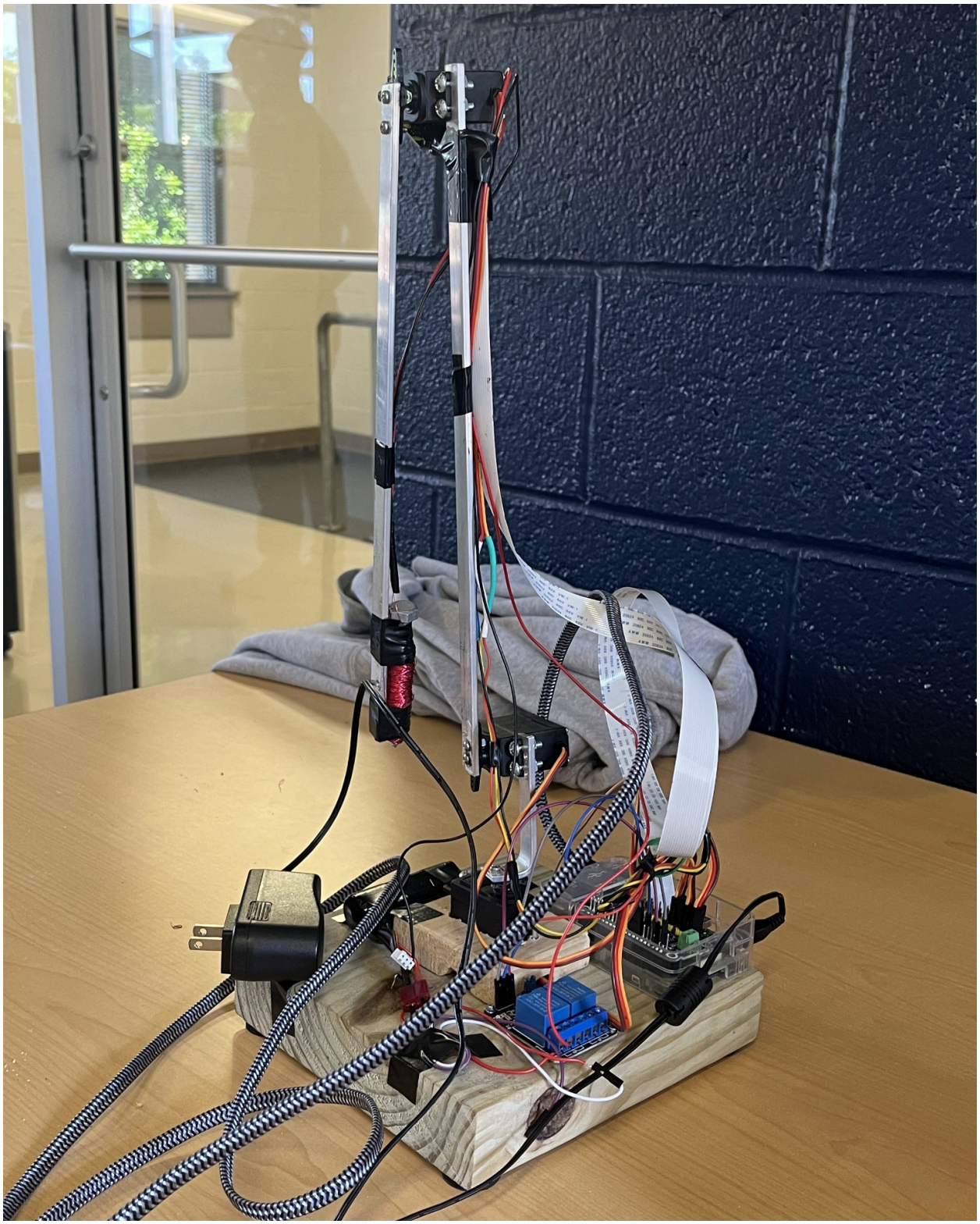
Summary:

The goal of this project was to create a robot arm, capable of cleaning a workplace bench. We decided on this project due to a few factors such as the complexity and physics behind using an arm, utilizing motion through servo motors, and the practicality of the project, being able to be used for keeping a work area clean.

System Design:

The prototype determines “islands” where there are isolated small parts to pick up. This is done through a programmed scan, in which the arm moves to a designated starting point and then slowly moves across the area of vision. If the arm “sees” an isolated part, it will stop moving and move towards the object using one servo, use another servo to get the electromagnet close to the object, and turn on the magnet to pick up said object. It will then go behind the area of vision to drop it in a designated spot by turning off the magnet. This process will repeat until it has finished scanning through the area of vision and cannot see any objects to be picked up.

Physical Design:



A wooden block was hollowed out to use for the base of the block. The servo main is mounted in the block with screws. This block sits on the main platform and is screwed down. This servo is for horizontal rotation. The servo is connected to a straight, sturdy piece of metal approximately 1 ft in length. Another servo is attached to the end of this arm, and another arm of equal length is attached to that servo. For the electromagnet, we utilized a nail and a wire to stimulate the effect of a magnetic field. All servos are connected to the Raspberry Pi through a PCA9685 chip. Some of the servos did not have the wire range to connect to the Pi directly, and so one the servos had wires soldered on to increase their effective length. To get automatic movement, a RaspberryPi camera was used and attached to a servo with tape. We used a mixture of electric tape and zip ties to contain the wires traveling up and down the arm. The Pi itself is mounted to the base alongside the relay, battery, and button for the purpose of starting the robot. The Raspberry Pi has a casing which allows it to be mounted down to the block with the necessary holes needed to allow for the necessary wires.

Software Design:

Upon starting the software, the arm moves into a default position and enters a limbo state, where it will wait for a button to be pressed. Upon button press, the arm will begin its journey to rotate horizontally up to 190 degrees. During its journey, it will stop at every integer angle for a brief amount of time to take an image using its camera. If there is an object detected, the arm will extend and attempt to pick it up with its magnet, drop it off into a designated position, and return to where it was before picking up the object. This will continue until it has swept the entire range of horizontal motion, at which point it will return back to its

default position and await another button press. The software has two main parts that work dynamically to calculate how to move and actually move the arm: Image analysis and servo movement.

Image Analysis

The image recognition software utilizes the object recognition to find the distance away an object is. We utilized OpenCV to help analyze camera images, which were taken with the Picamera library. Images were taken once every rotation of the base servo. Each image is analyzed on the fly through a series of steps. First, the image is turned into grayscale for simplification. This does mean that the objects that we would want to pick up must have a different coloration from their background, and usually mean that the background must not be gray. Then, the

algorithm attempts to construct a new image with only edges it detects. Finally, the findContours function is used to convert collections of edges into proper shapes and objects. For each object detected, the will form a bounding rectangle around what it believes to be the midpoint of the object, and gives adjusted x and y pixel values of its location.

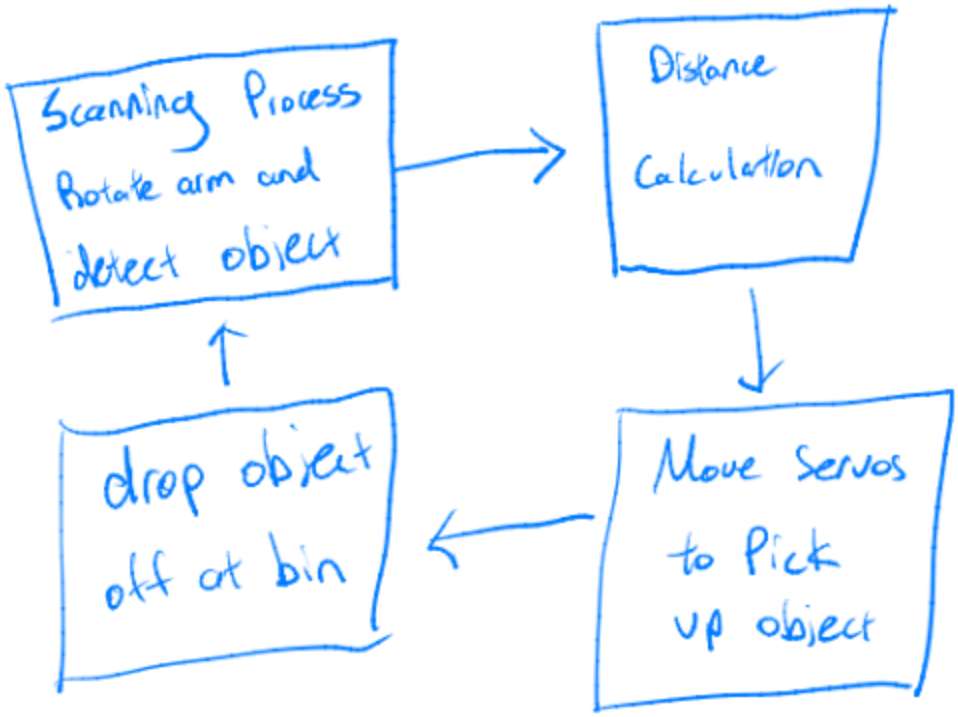
Servo Movement

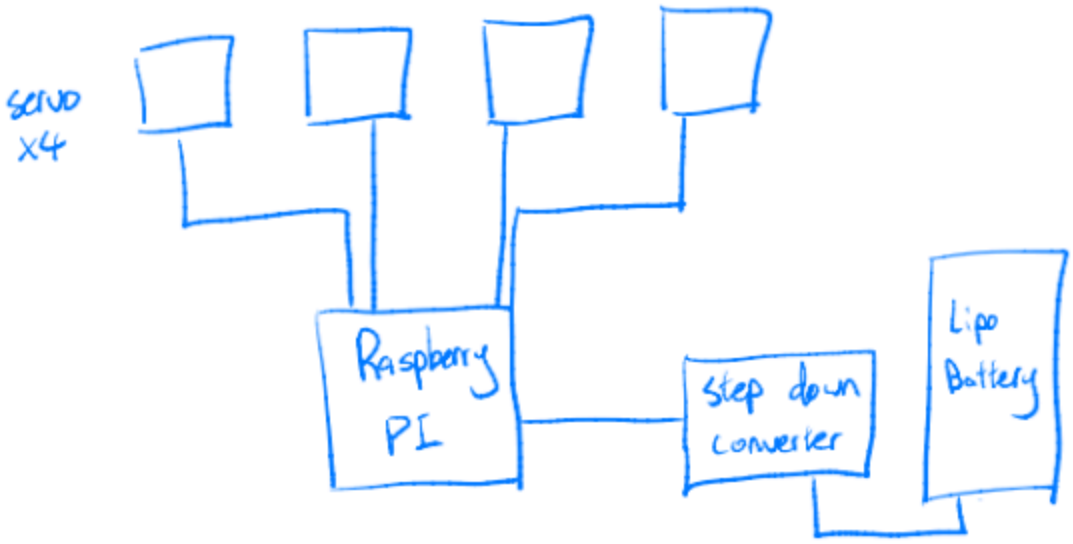
```
#rotates, searching for objects to pick up and picks them up.
def scan_and_rotate_servo():
    start_angle = 10
    end_angle = 150
    step = 1  # Define step size for angle increment

    # Allow the camera and servo to warm up
    time.sleep(0.1)

    # Main loop to capture frames and rotate servo
    for angle in range(start_angle, end_angle, step):
        servo0.angle = angle
        writePosition('0', str(angle))
        time.sleep(0.05)
        # Capture a single frame
        camera.capture(rawCapture, format="bgr")
        image = rawCapture.array

        # Process the image
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 50, 150)
        _, binary = cv2.threshold(edges, 50, 255, cv2.THRESH_BINARY)
        kernel = np.ones((5,5), np.uint8)
        dilated = cv2.dilate(binary, kernel, iterations=1)
        contours, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        height, width = image.shape[:2]
        for contour in contours:
            x, y, w, h = cv2.boundingRect(contour)

            if x > 0 and y > 0 and (x+w) < width and (y+h) < height:
                M = cv2.moments(contour)
                if M["m00"] != 0:
                    cx = int(M["m10"] / M["m00"])
                    cy = int(M["m01"] / M["m00"])
                    if (cy >= 200):
                        continue
                    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
                    cv2.circle(image, (cx, cy), 5, (0, 0, 255), -1)
                    if 140<= cx <= 150:
                        #print("Object detected at 100px along the x-axis")
                        #print('cy', cy)
                        angle1, angle2 = get_servo_angles(cy)
```

The arm is composed of three servos: A base servo and two arm joint servos. The base servo is responsible for horizontal rotation while the two arm joints are responsible for forward movement. The x pixel value given by the image recognition software was calibrated by hand to line up with the base servo's movement. This made it so that the arm would only activate when it was directly lined up with an object.

The y pixel, on the other hand, was used to determine the distance between the arm and the object. Y-pixel to servo angle values were calibrated by manual testing. After a set of such values were obtained from testing, we used a linear interpolation algorithm to produce servo angles on the fly for the arm. As for arm movement itself, the arm moves one servo at a time, with small delays between each movement to account for lasting momentum. The movement speed for all movements aside from the first initial movement is also intentionally slowed down in order to provide stability for the arm, as simple one-movement commands could cause instability. There were two keys to allow for this to happen. First, the exact positions of each servo must be known. This was done by setting the arm to a default position when powered on for the first time, and recording the servo angles in a file. Secondly, piecewise movements were used. Each time a move command was requested, the current angle of the servo would be read from the file. The servo would then move a small fraction towards the requested angle, update its new location in the file, and stop for an incredibly small amount of time. This would be repeated until the servo reached the requested angle . In real time, this gives the illusion of slower, but still continuous movement with little to no jitter.

Pseudocode for the software design is provided below.


Scanning Process
Rotate arm and
detect object
Distance
Calculation
Move servos
to Pick
up object
drop object
off at bin



Challenges & Areas of Improvement:

While developing our design, we had anticipated a number of challenges and problems, due to none of us having a particularly large amount of hardware experience and next to no experience with working with servos. Therefore, we decided to be flexible with the project as our work progressed and we learned what we could and couldn't manage to implement.

Claw Vs Magnet:

One of the most important examples of this is the absence of the claw on the final product. Originally, we planned for the arm tip to hold a claw that would pick any item up instead of an electro-magnet that could only pick magnetic objects up. However, our two claw prototypes suffered from various issues, including excess friction that caused difficulty in moving in one design, and a failure to rotate both claw arms to the same degree in the other. It was suggested to us that 3-D printing

should be used when making parts that move so much, but we did not have enough time nor the experience to make the parts. Therefore, we decided to implement an elector-magnet instead.

Servo Movements:

```
#Moves a servo to an angle.
def move(servo, num, ang):
    initialAngle = float(getPosition(num))
    angleDiff = (abs(ang - initialAngle))
    if (angleDiff < 0.3):
        return
    numStep = max(round(1.2 * angleDiff), 1)
    increment = (ang - initialAngle)/(numStep)
    currentAngle = initialAngle
    for x in range(0, numStep):
        currentAngle += increment
        if (currentAngle < 0):
            currentAngle = 1
        elif (currentAngle > 180):
            currentAngle = 180
        servo.angle = round(currentAngle,1)
        writePosition(int(num), str(round(currentAngle,1)))
        time.sleep(abs(increment/40))
```

Another challenge was the difficulty of calibrating servo movements. Our final software takes pixel values and directly converters them into angles that the servos can use. However, the camera software was not finished for much of the project's duration, and so during that time, the servos' angle calculations were done using trigonometry. The Law of Cosine was utilized in conjunction with solving for an additional angle to lower the arm so the tip of the arm could touch the ground. First, the angles for the two arm servos were found using the known lengths of each arm and a given distance via the Law of Cosine. Next, a triangle was created "below" this triangle, with two of the shorter sides being the given

distance and a set value that was slightly smaller than the distance between the arm and the ground. Trigonometry was used to determine the distance between the near ground location and the bottom arm servo. The Law of Cosine was called again with this distance in mind to produce a final value for the upper arm servo, and finally, the angle from the second triangle was added to the bottom arm servo's angle from the second Law of Cosine to produce the final value for the lower arm servo. This proved surprisingly consistent, but not accurate; the arm would be off by 0.5-2 inches depending on where it was sent, and sometimes it was too high off the ground. Errors likely arose due to the arms not being tightly wound enough to the servos, and the servos themselves not being strong enough to fully support the arm and allowing for a bit of slack when the arm was extended. Another issue was present throughout the course of the project, although its problems were dampened with better software. The servos would sometimes jitter during or after a movement for seemingly no consistent reason. The jittering produced a small amount of sound and constantly distorted the arm's current position by a small degree. Slowing servo movements down and making smaller steps were instrumental in improving the consistency of the arm in this way, although they were not enough to fully solve the problem.

Wiring:

Wiring and attaching items to the base were minor issues towards the latter part of the project. The camera we used had flat and long wiring which could not be folded to any significant degree or else the camera would continuously throw errors. We settled on creating a large loop within the wiring in order to slightly reduce the amount of free wiring while still not bending the wire so much that it would fail. The wire was also sensitive to the electrical tape we used, and so it could only be used sparingly, and alongside zip ties to secure the flat wire. Normal

servo and electromagnet wires were secured to the arms using a combination of electrical tape and zip ties. Our raspberry pi and relay were secured to the wooden base via screws (the former being held in a special casing that was screwed), and the battery had to be taped.

Materials:

Item Name	Source	Count	Cost
7.4V Lithium Ion Battery	Amazon	1	$23.79
Button	Texas A&M University	1	—
Drill	Owned	1	—
Flat Aluminum Bar	Home Depot	1	$6.42
Hack Saw	Owned	1	—
Raspberry Pi	Texas A&M University	1	—
Raspberry Pi Camera	Adafruit	1	$19.98
Raspberry Pi Camera Extension Cord	Amazon	1	$5.49
Relay	Texas A&M University	1	—
Sanding Sponge	Owned	1	—
Screwdriver	Owned	1	—
Servo Driver Module	Amazon	1	$17.99
Servos	Amazon	3	$19.99
Step Down Converter	Amazon	1	$7.65

Conclusion:

Overall, the project was a success despite the large amount of setbacks we faced as a group. Although we strayed from the initial design, we were able to develop a successful project that allowed for the pickup of small metallic objects in a workbench setting. There is still room for improvement but given the time constraint, the project was a success.