Checkpoint #4 Report

[EECN30169] Mobile Robot 2024

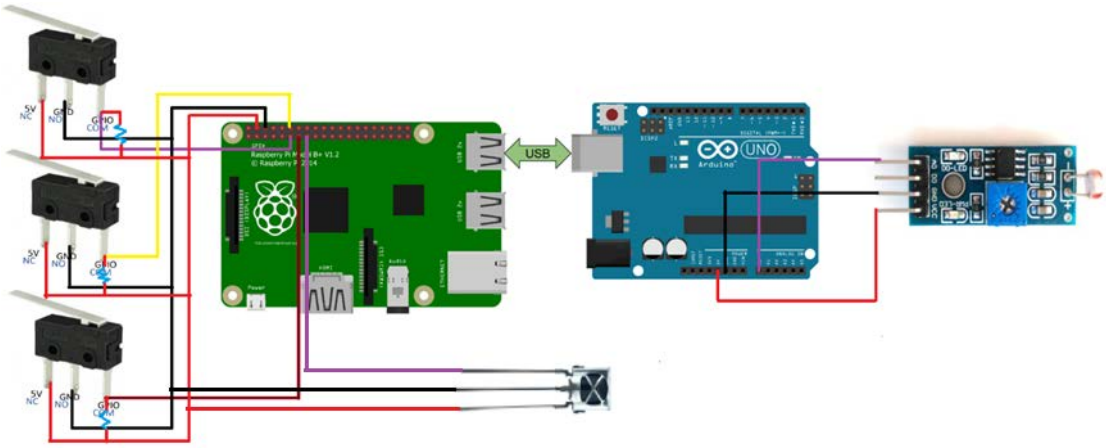
Student ID: 313605019 **Name:** 方敏 **Date:** **2024/11/15**

1. Purpose:

The purpose of this experiment is for the robot to grab the light ball, then use infrared receiver to detect the position of the beacons and push the light ball into the hole with the claw we designed. During the experiment, the robot needs to identify the locations of both beacons simultaneously to ensure accurate task completion.

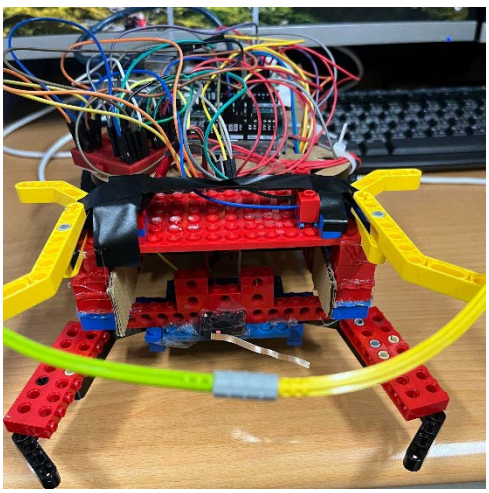
2. Description of Design:

Wiring:

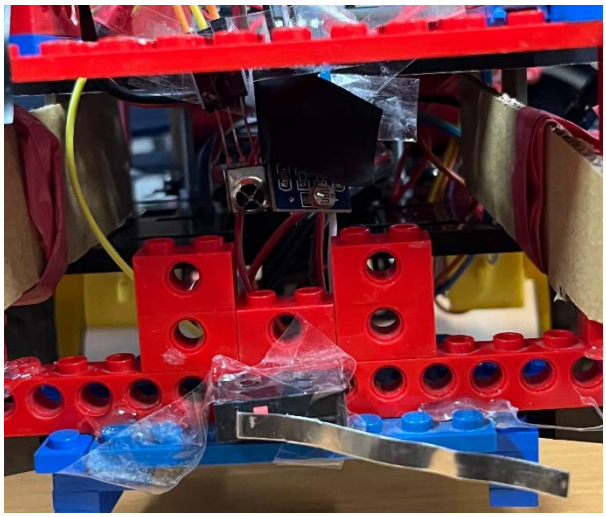


This implementation requires the addition of an infrared receiver. Since only a few pins remain available on the Arduino, we decided to connect the infrared receiver to GPIO3 on the Raspberry Pi. This approach not only allows the main program running on the Raspberry Pi to receive sensor data in real-time but also takes advantage of the many unused pins on the Raspberry Pi, with another reason I will discuss in the **Discussion** section.

Assembling:



Upon testing, we found that the range of the infrared receiver is very wide. To receive signals more accurately, we adopted a similar approach to the light sensor by placing the infrared receiver in a more concealed position. This ensures that when a signal is received, the robot is directly facing to the beacon.



Coding:

Arduino:

On the Arduino side, the program hasn't changed much compared to cp3. Since the infrared sensor is mounted on the Raspberry Pi, the Arduino publisher only needs to output these three values.

```
if (output_counter == OUTPUT_DELAY) {
  output_counter = 0;
  output_msg.data[0] = analogRead(light);
  output_msg.data[1] = encoderA_accumulation;
  output_msg.data[2] = encoderB_accumulation;
  output_pub.publish(&output_msg);
}
```

Raspberry Pi:

The main modification in the program is on the Raspberry Pi side. We created a new background thread to add support for the infrared receiver without blocking the main program, continuously reading the receiver's values. To determine which target signal is detected, the program calculates the ratio of 0s to 1s within a fixed time frame to decide if the target signal has been received. To prevent signal instability, we set a rule that if the target signal is not detected for n consecutive times, then signal interruption is confirmed. Based on our actual testing, we set the threshold to 2.

```
threading.Thread(target=ir_sensor_output).start()
```

```
def ir_sensor_output():
    ir_count0 = 0.0
    ir_count1 = 0.0
    not_find_ir_counter = 0
    while not rospy.is_shutdown():
        if GPIO.input(3) == GPIO.LOW:
            ir_count0 += 1
        else:
            ir_count1 += 1

        if (ir_count0 + ir_count1) >= read_ir_times:
            received_ir_data = ir_count0 / (ir_count0 + ir_count1)
            if (received_ir_data >= beacon_lower_bound[beacon_select]) and (received_ir_data <= beacon_upper_bound[beacon_select]):
                not_find_ir_counter = 0
                find_ir = True
            else:
                not_find_ir_counter += 1
                if not_find_ir_counter >= not_find_ir_threshold:
                    find_ir = False
            ir_count0 = ir_count1 = 0.0

        time.sleep(0.0001)
```

We divided the program into two main parts: one part is searching for the light ball, and the other is searching for the beacon. The light ball search part is similar to the previous checkpoint 3, where the robot moves forward and then rotates right after hitting the wall. However, unlike checkpoint 3, to ensure whether the robot has correctly grabbed the light ball, we didn't hard-code the path. Instead, we let the robot rotate in a full circle and then proceed in the brightest direction. By calculating the motor movement path (using encoderA_accumulation and encoderB_accumulation provided by the Arduino publisher), we confirm if the robot has completed a full rotation and then proceed toward the brightest direction.

```
    while ARotate > encoder_a_plus_b:
        accumulative_motor += encoder_a_plus_b
        if subscribe_data_queue:
            if light_data < min_light_data:
                min_light_data = light_data
                min_light_angle = accumulative_motor
        print("encoder: ", encoder_a_plus_b)
        time.sleep(0.01)

    # Stop rotation
    publish_queue = [{'left': 0, 'right': 0}]
    print("Done rotating")

    if min_light_angle != 0:
        direction = 1 if min_light_angle > (ARotate / 2) else -1
        publish_queue = [{'left': direction * rotate_speed, 'right': -direction * rotate_speed}]
        time.sleep(abs(min_light_angle / rotate_speed))
```

Once the light ball is found, the program will enter the beacon search phase. Due to the wide detection range of the infrared sensing function, and considering that the light ball is not far from the target hole, we modified the code so that after grabbing the light ball, the robot rotates to the left. If the beacon is not detected, it

will move forward after a 90-degree left rotation.

```
while (time.time() - start_time) < 2.0:  # Rotate for approximately 90 degrees (time-based)
    if find_ir:  # If beacon is detected
        found_beacon = True
        break
    time.sleep(0.01)

# Stop rotation
publish_queue = [{'left': 0, 'right': 0}]

if found_beacon:
    print("Beacon found, stopping rotation.")
else:
    print("Beacon not found, moving forward after 90-degree turn.")
    # Go straight after rotating 90 degrees
    publish_queue = [{'left': speed, 'right': speed}]
```

3. Result

In our demo, the times were 15 seconds and 18 seconds, reflecting that our strategy was quite successful.

4. Discussion

Problem 1:

Why didn't we install the infrared sensor on the Arduino?

Explanation 1 :

Initially, we connected the infrared receiver to the Arduino due to its extensive infrared encoding and decoding library. However, we discovered that the reason for the repeated compilation failures was the version of the library we referenced was too new. After switching to an older version, it did run successfully, but we found that using the infrared library caused frequent issues with rosserial. After extensive testing, we identified that the problem was due to rosserial consuming a large amount of dynamic memory. By default, both the send and receive buffers occupy 280 bytes each, totaling 560 bytes, which made the program unable to run on the Arduino, which has only 2K bytes of memory. Although we ultimately did not use this method to connect the infrared receiver to the Arduino, we still spent lots of time troubleshooting this issue.

Problem 2:

Why didn't we rely entirely on the infrared sensor?

Explanation 2:

Because we found that its detection range is too wide, which easily leads to angular errors. To prevent issues, we set a rotate time limit for detecting the infrared signal. In the future, we will try to cover the infrared sensors from other angles to ensure it only detects signals directly in front.

Problem 3:

What improvements can we do to the claw?

Explanation 3:

With the current claw design, the light ball may still probably slip out. We could try adding a servo motor to control the claw. Once the light ball is detected, the servo motor could rotate the claw inward to lock it in place, preventing the light ball from escaping.

In this checkpoint, I learned more about hardware control and software integration for robots. I also came up with many cool ideas for sensors that could be added to the robot. I am grateful for the guidance from the professor and teaching assistants. I hope that the hockey contest and final project will proceed more smoothly.