San Francisco State University

CSC 615 -Fall 2023

12/12/23

Raspberry Pi Car

Team Procrastinators

Github name: vngo00

Github repository link:

https://github.com/CSC615-2023-Fall/csc615-term-project-vngo00

Ameen Safi | 920689065

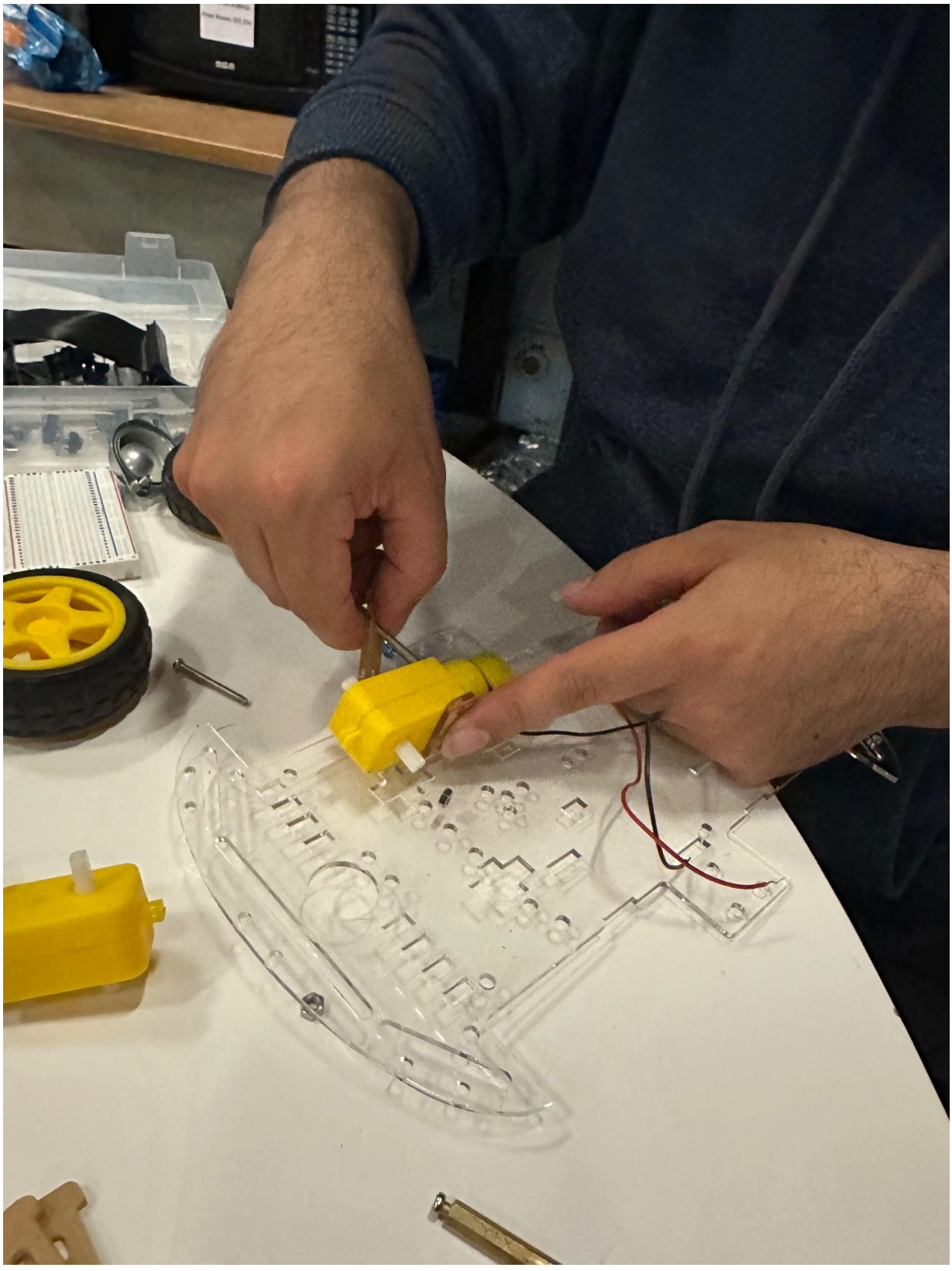
Vinh Ngo | 920984945

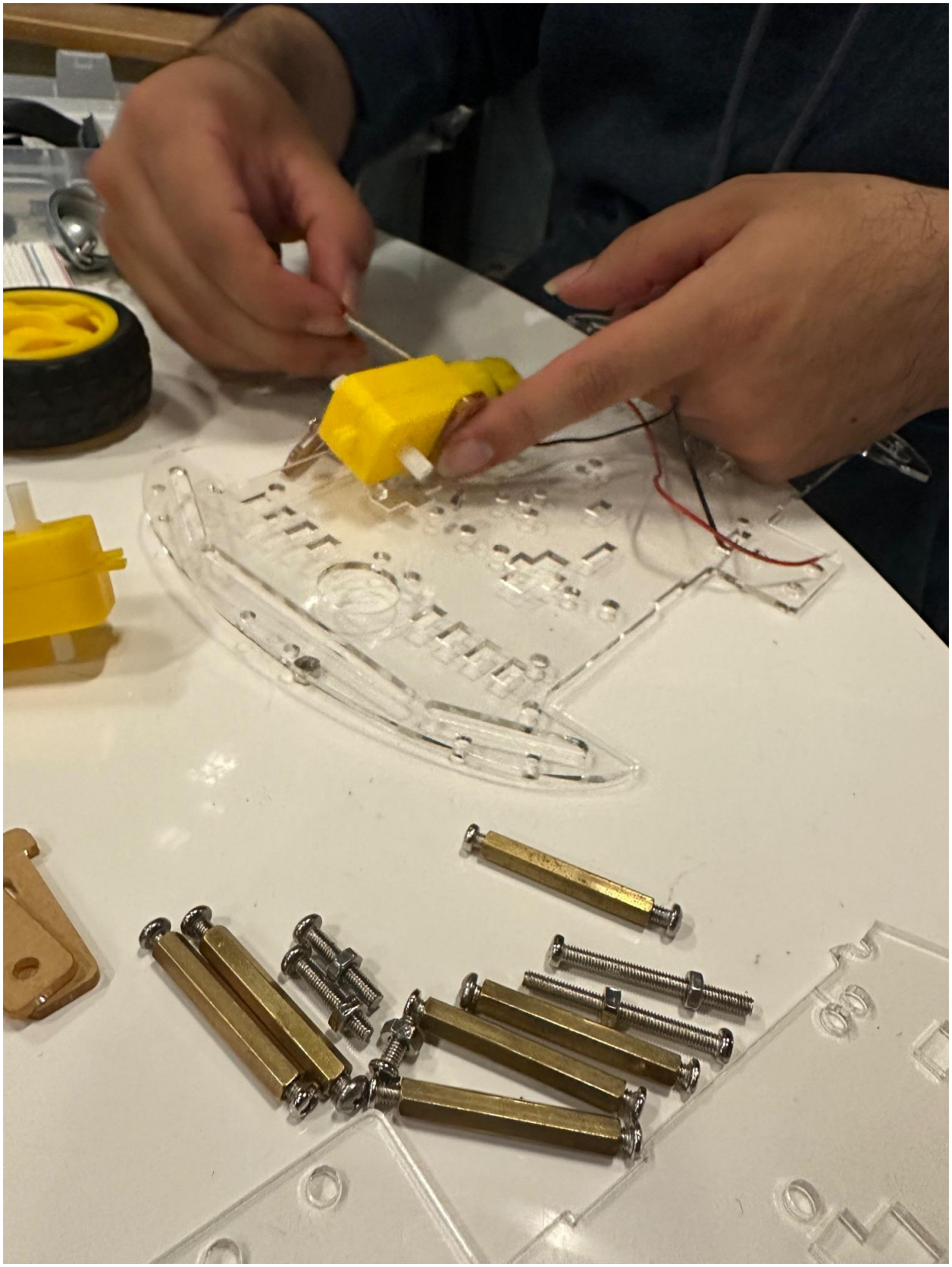
James Donnelly | 917703805

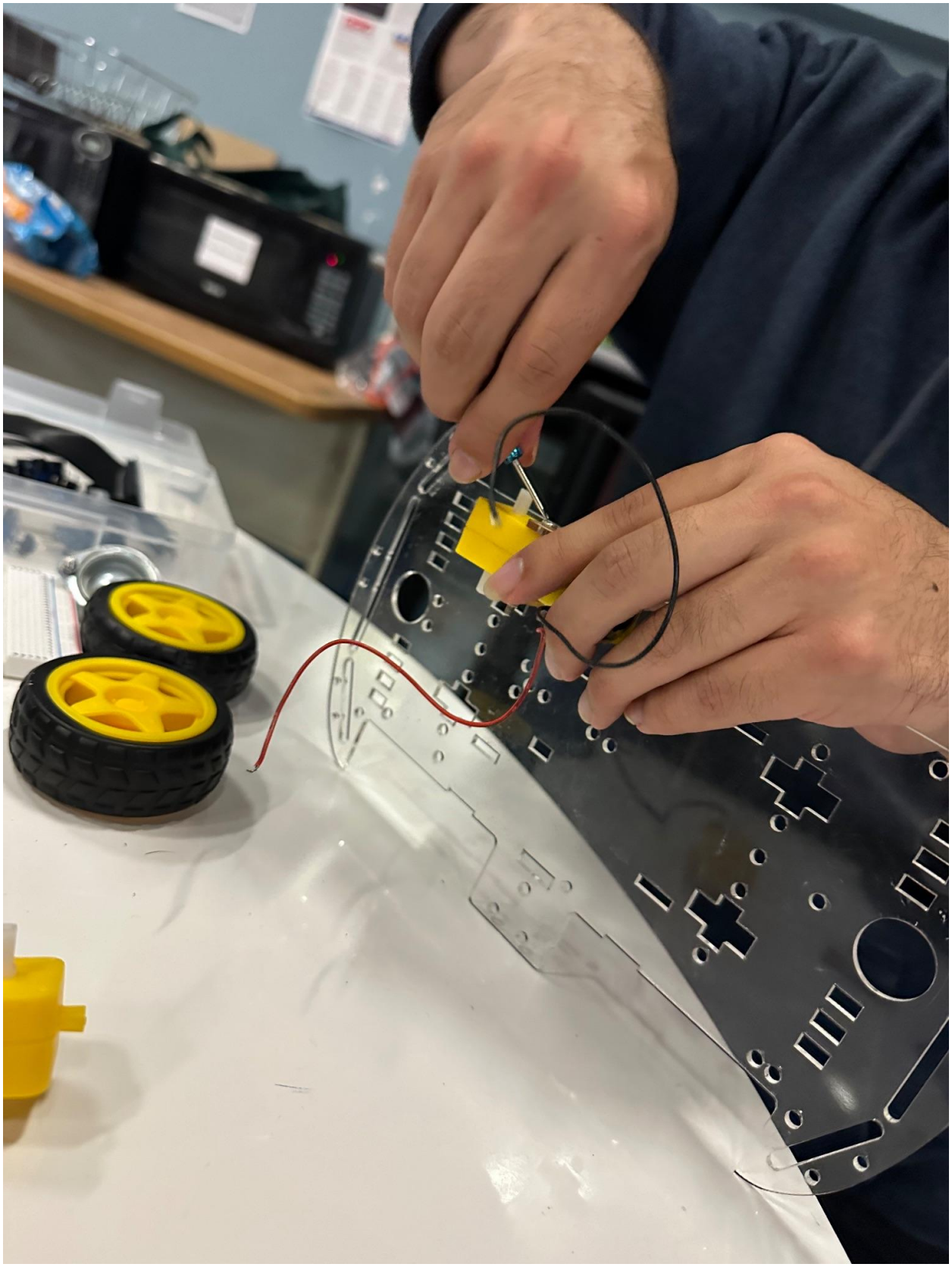
Task Description

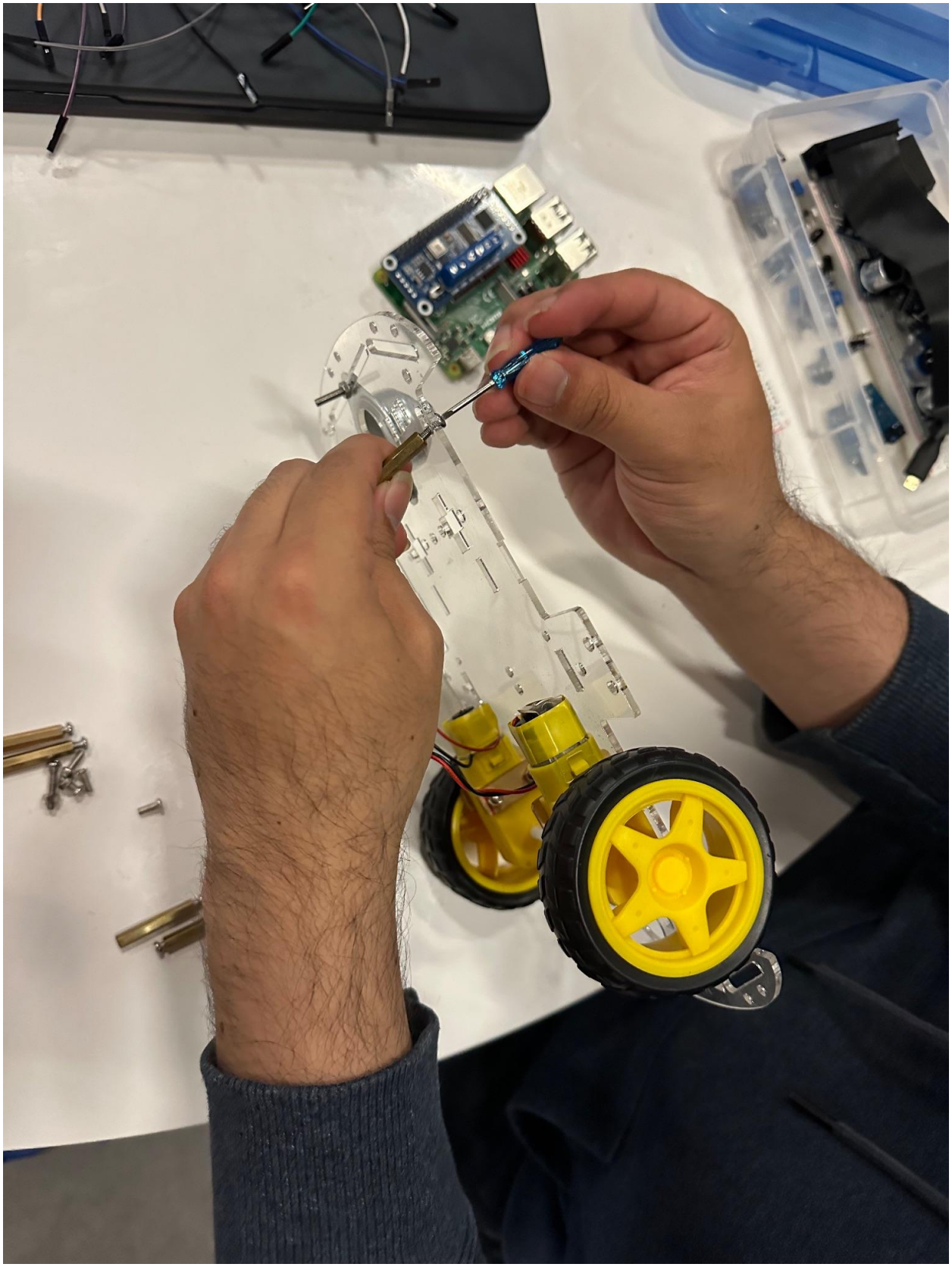
The task is to make a car that runs autonomously and follows the line. With following the line it must be able to avoid obstacles and get back on the line after successfully avoiding the object. In addition we have to design and build the car with our own choices. This includes but is not limited to choosing the power source, how it is started, the wheels to be used if wheels at all, the motors to be used, the sensors used, the body of the vehicle, and the code for the actual process of the car.

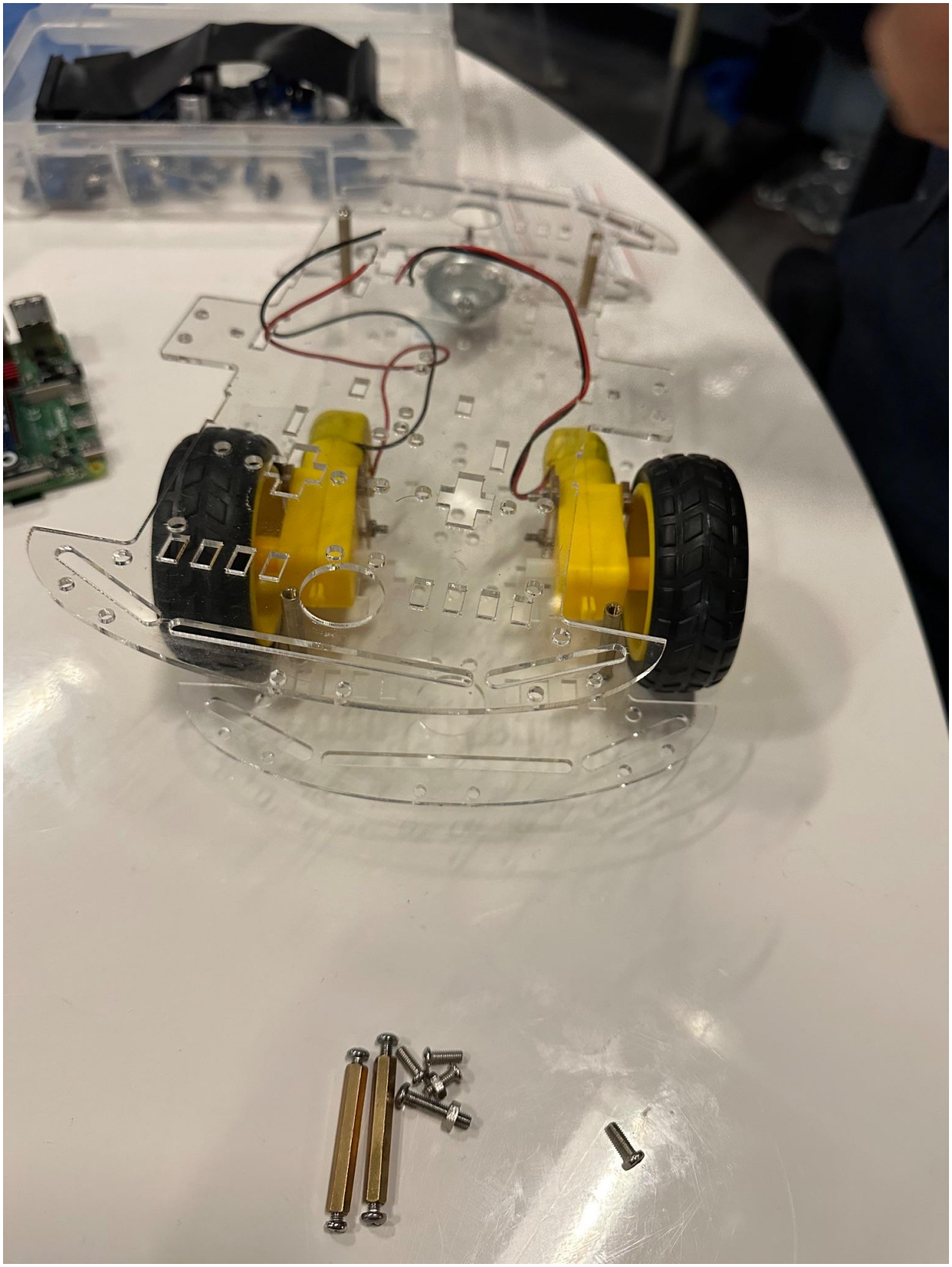
Building the Robot

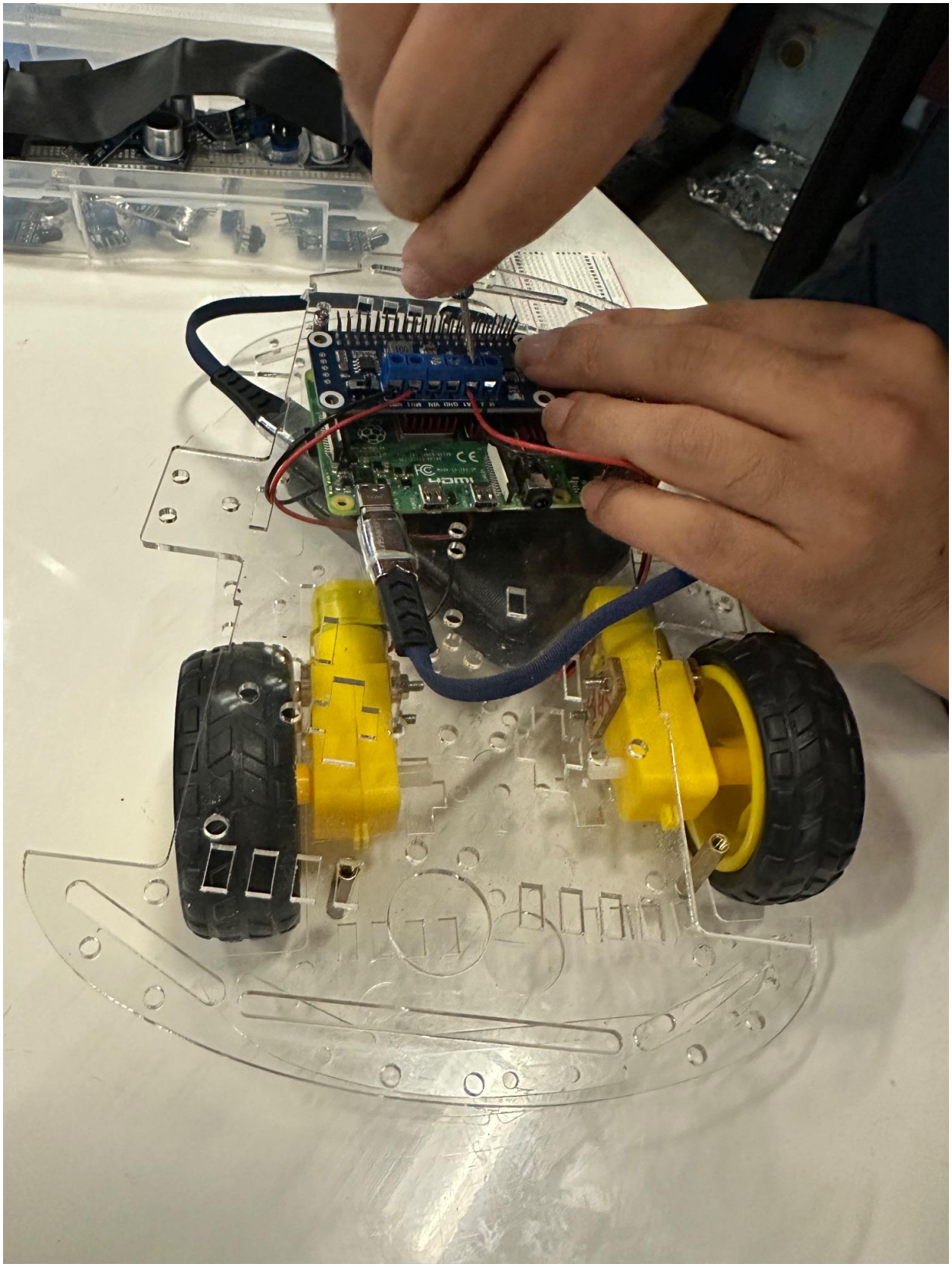


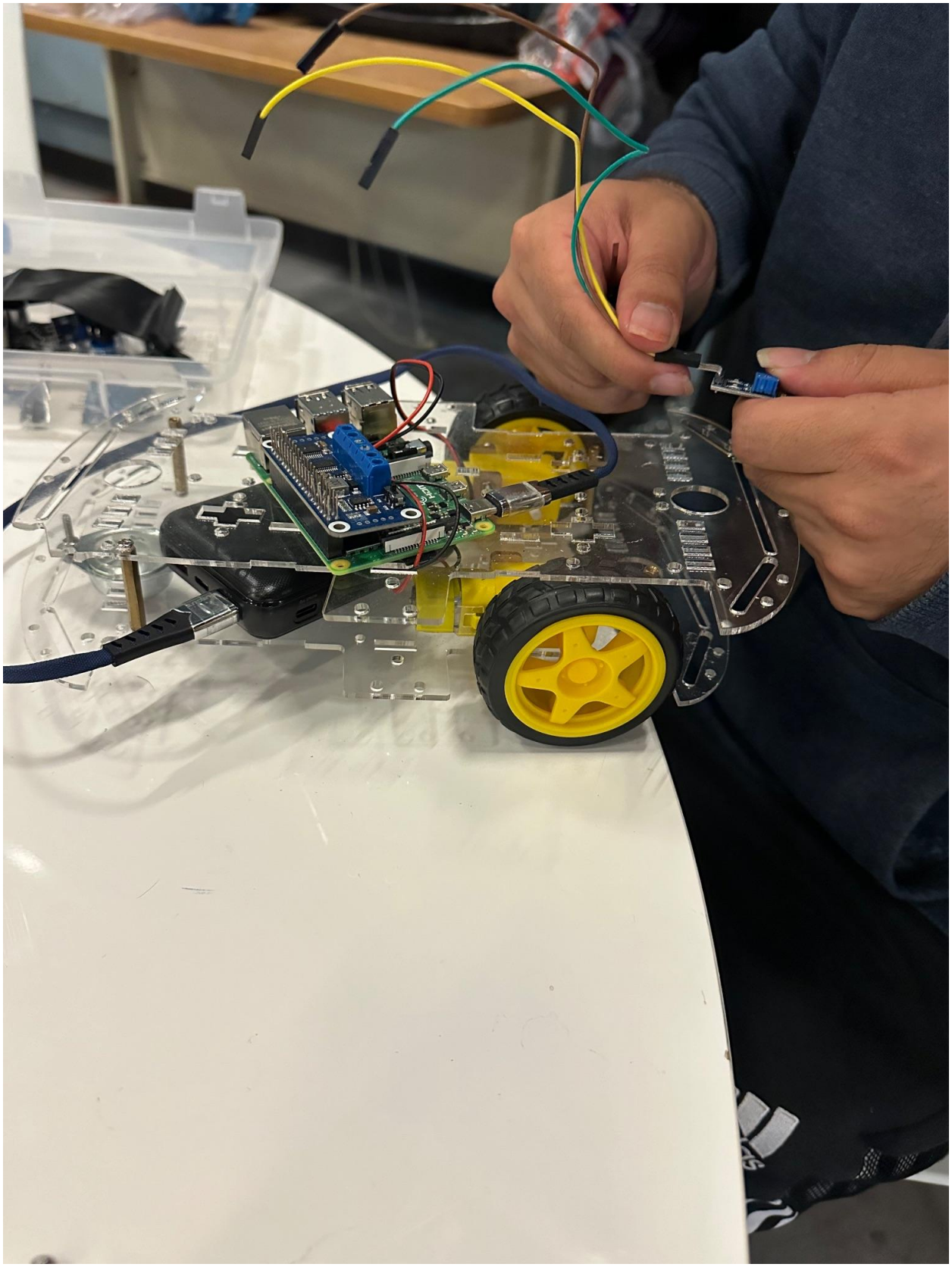


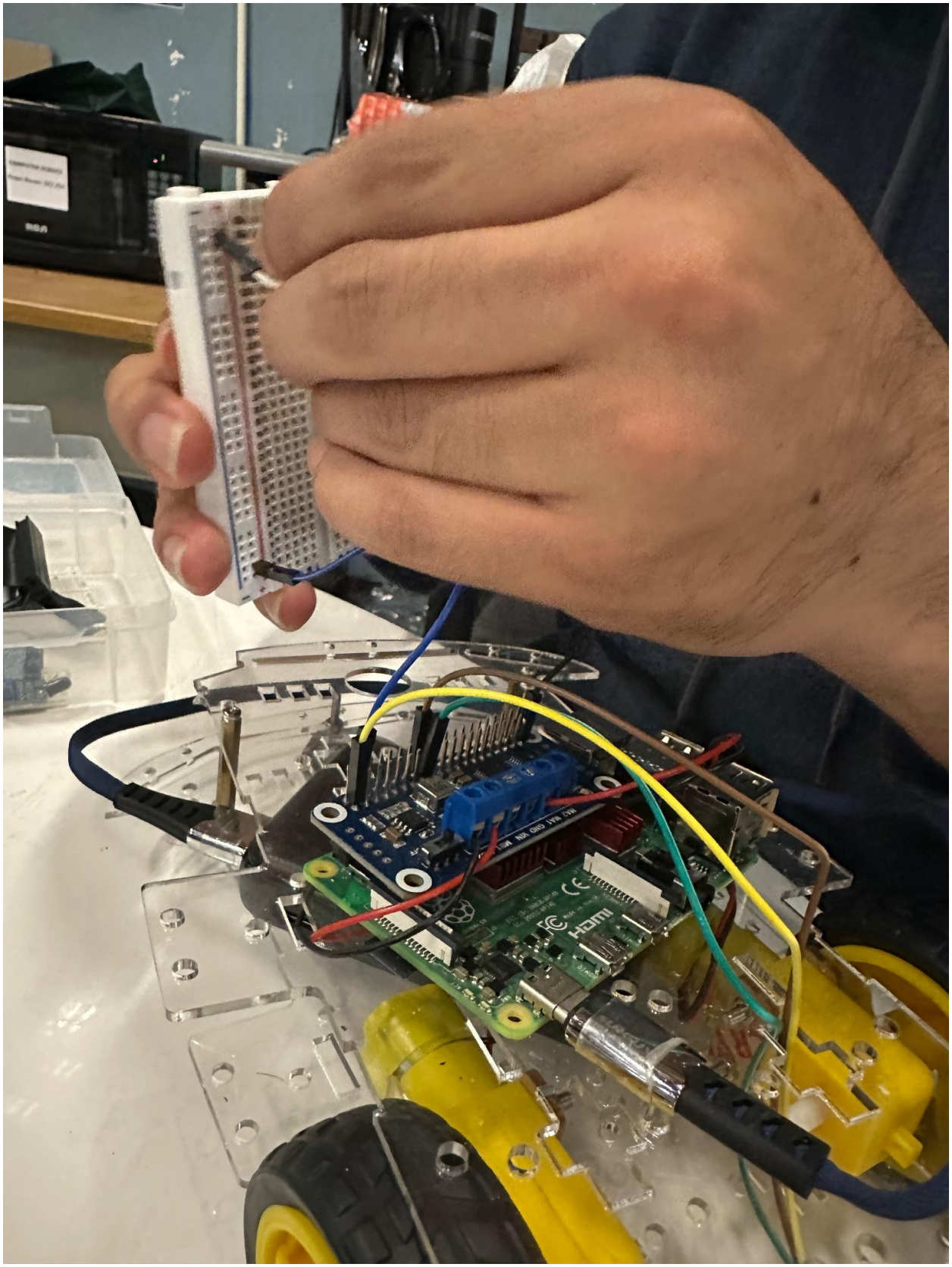


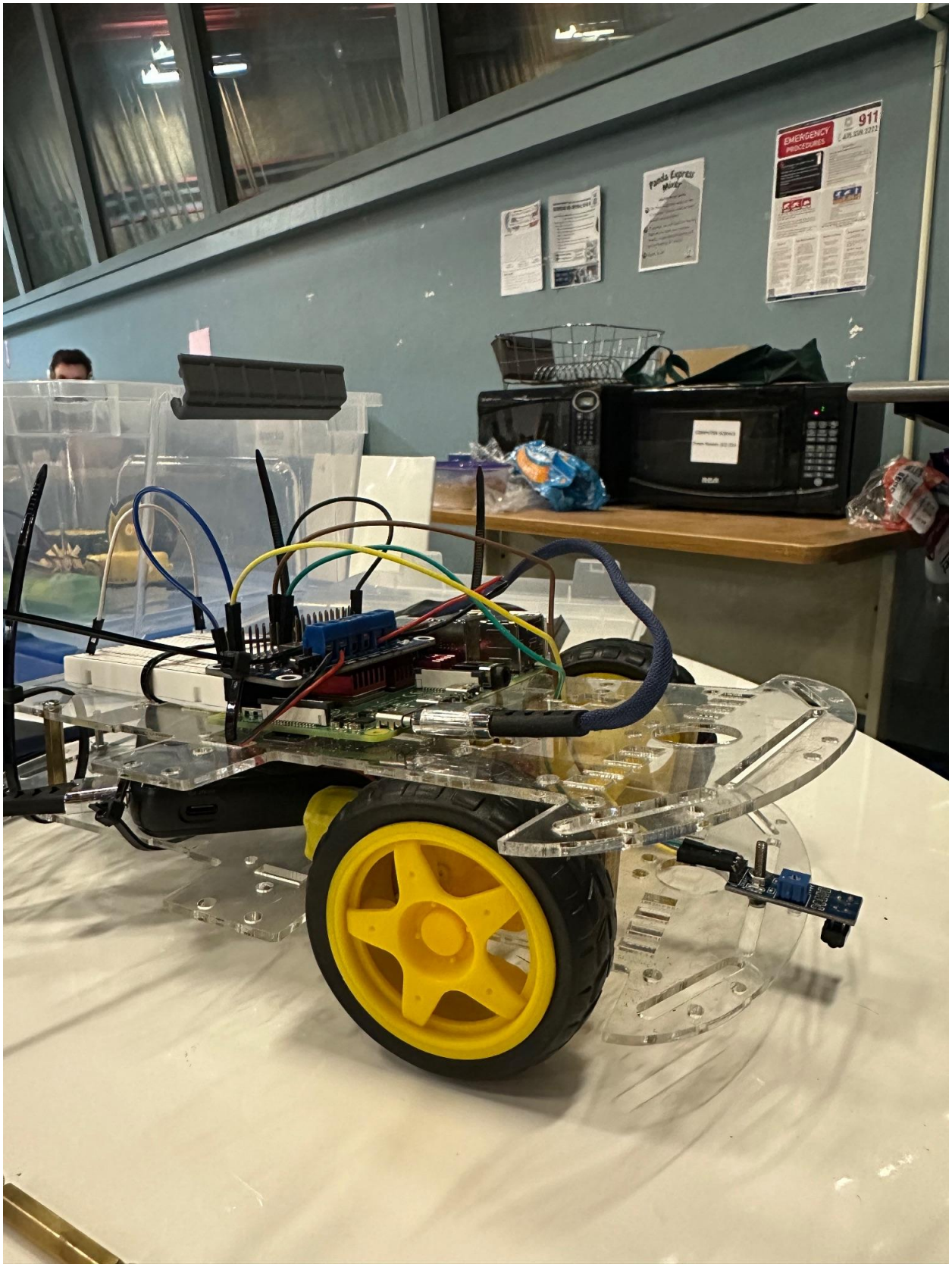


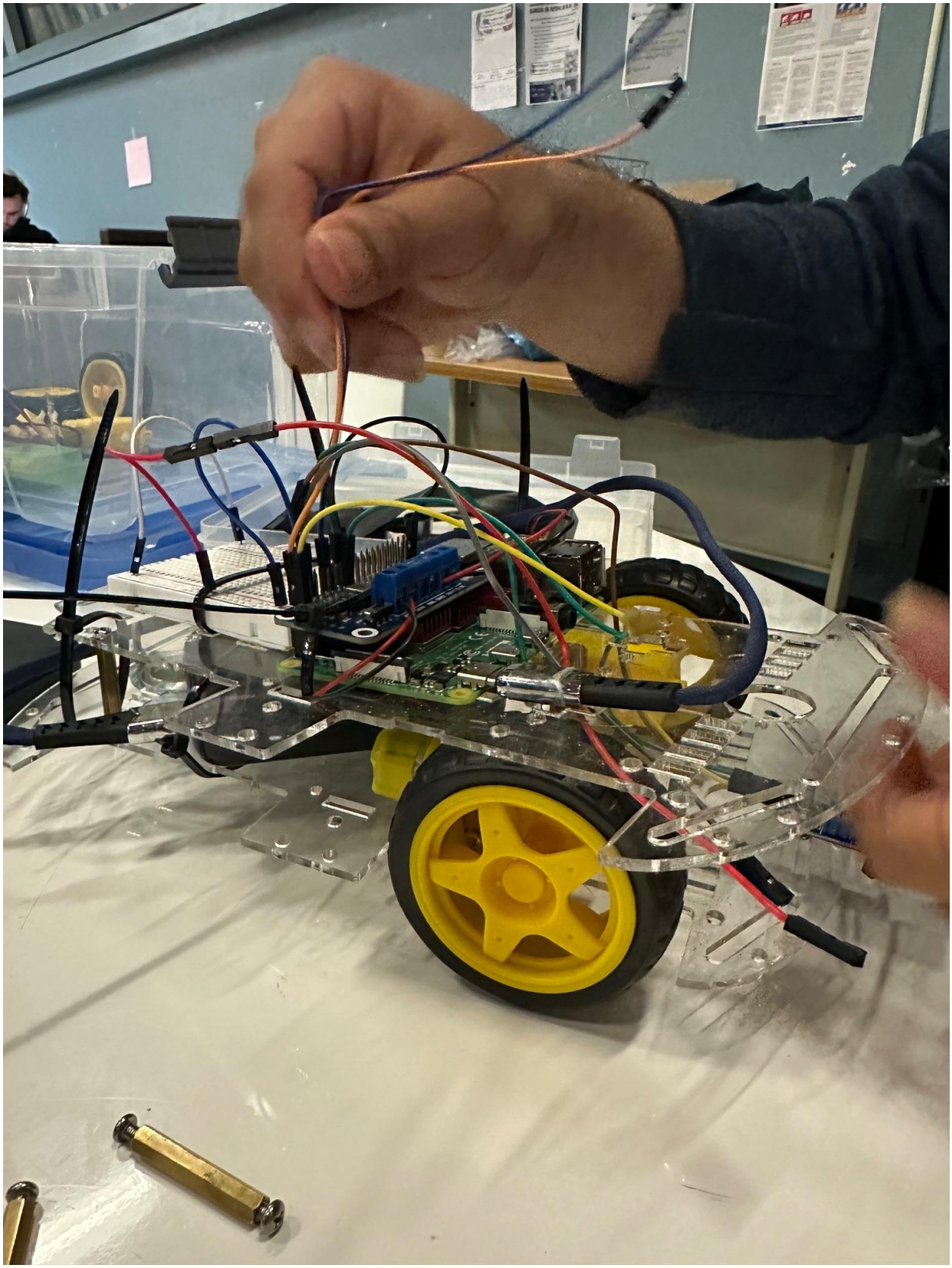


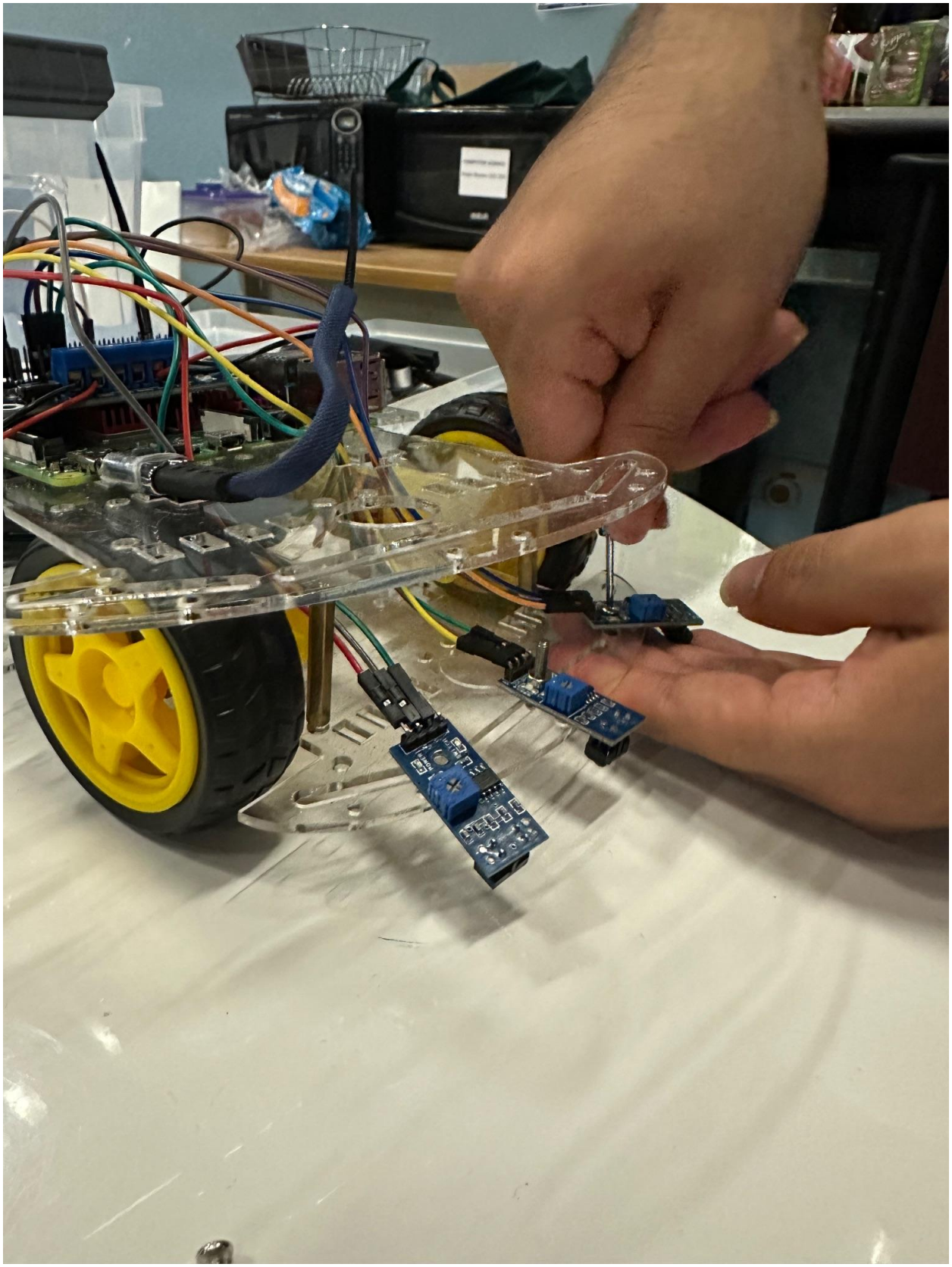


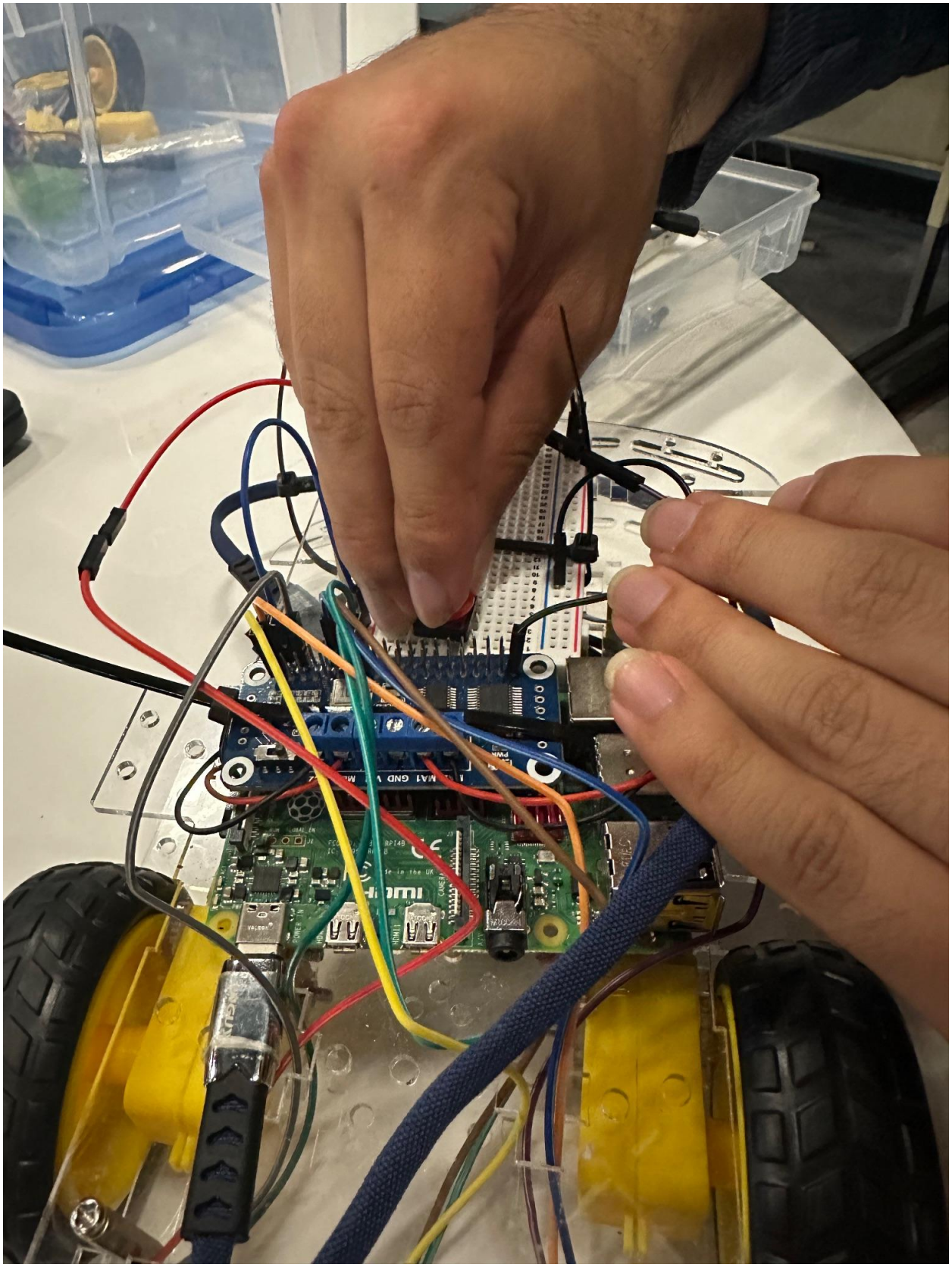


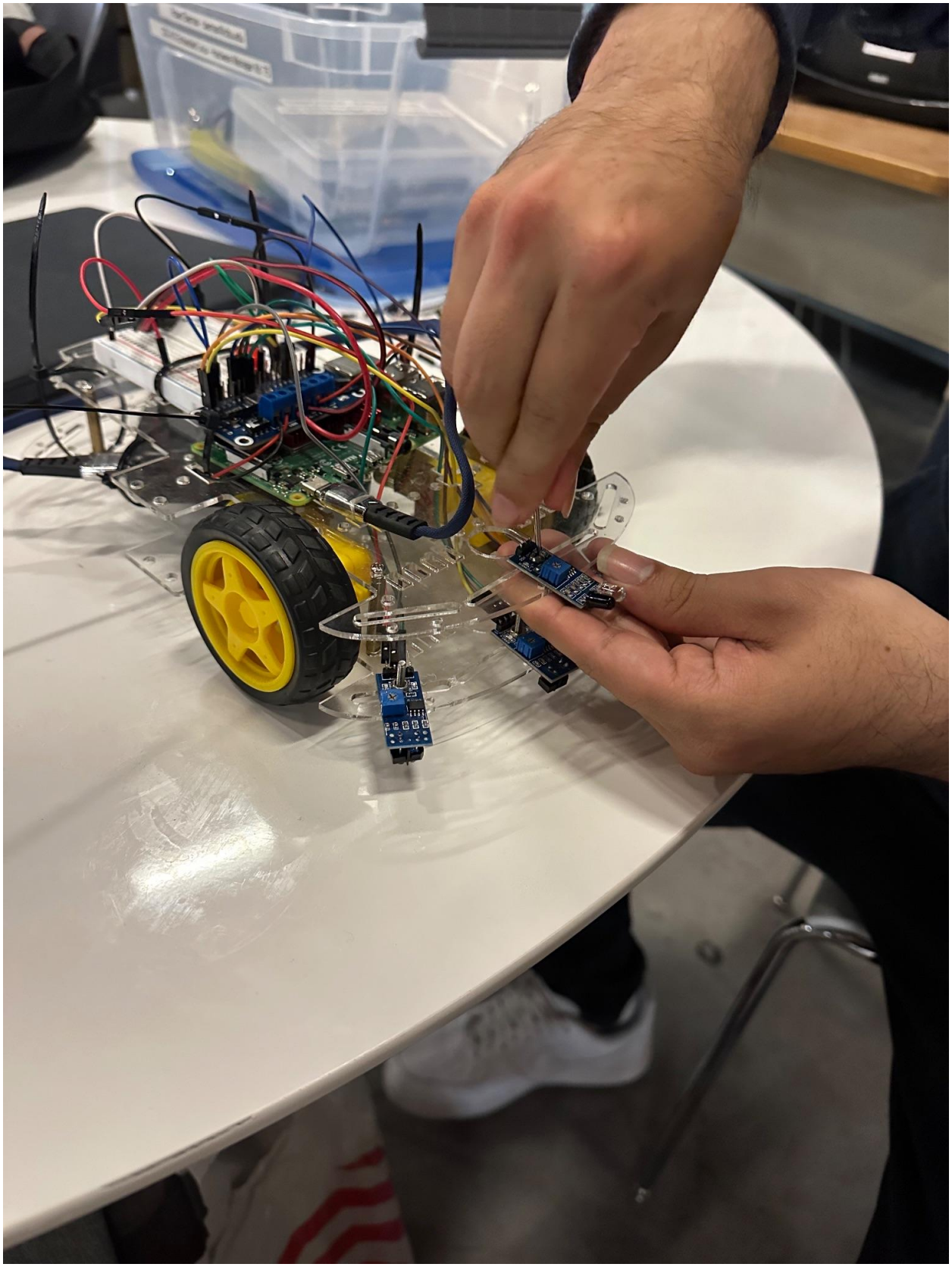


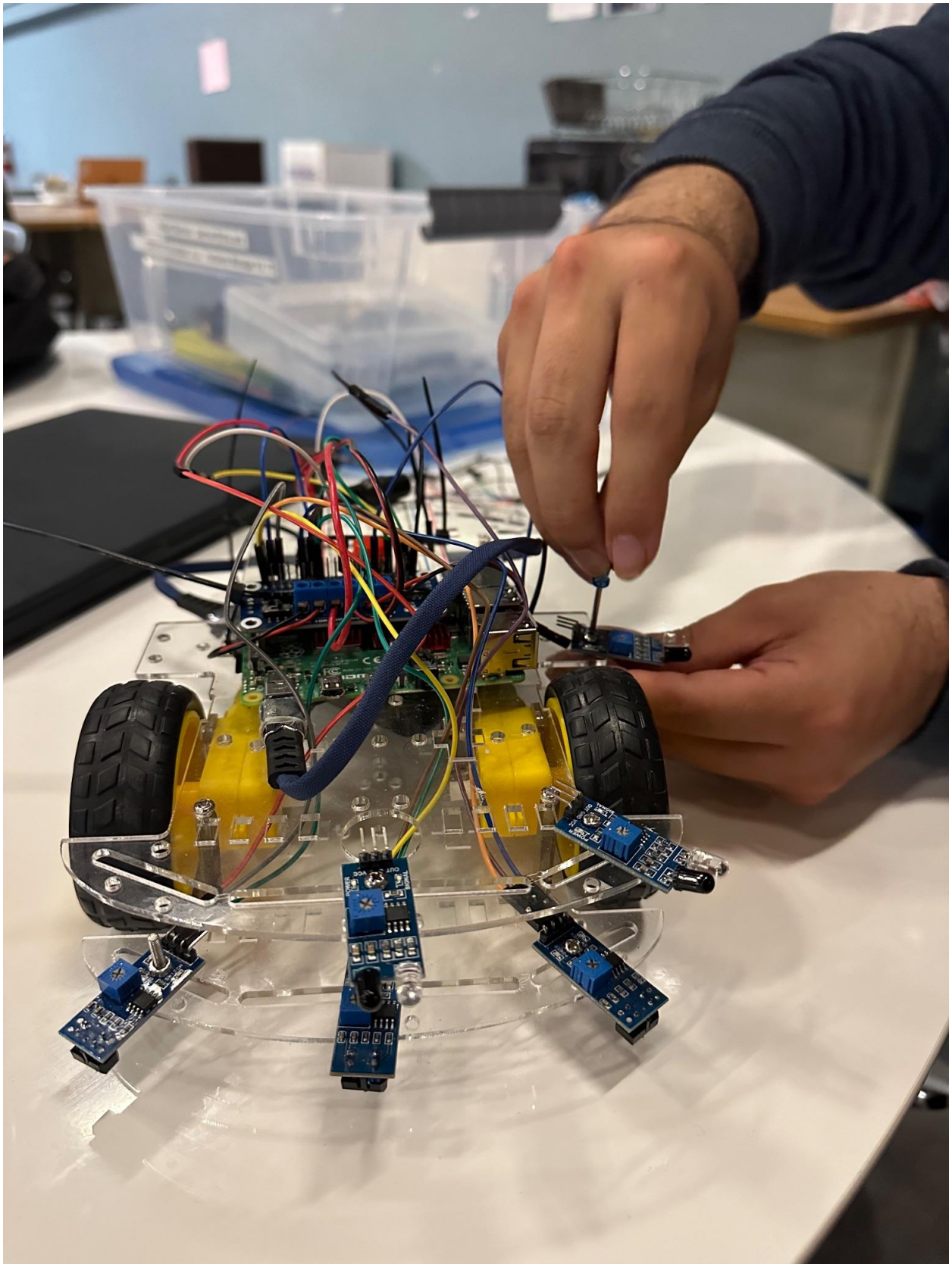

EMERGENCY
PROCEDURES
911

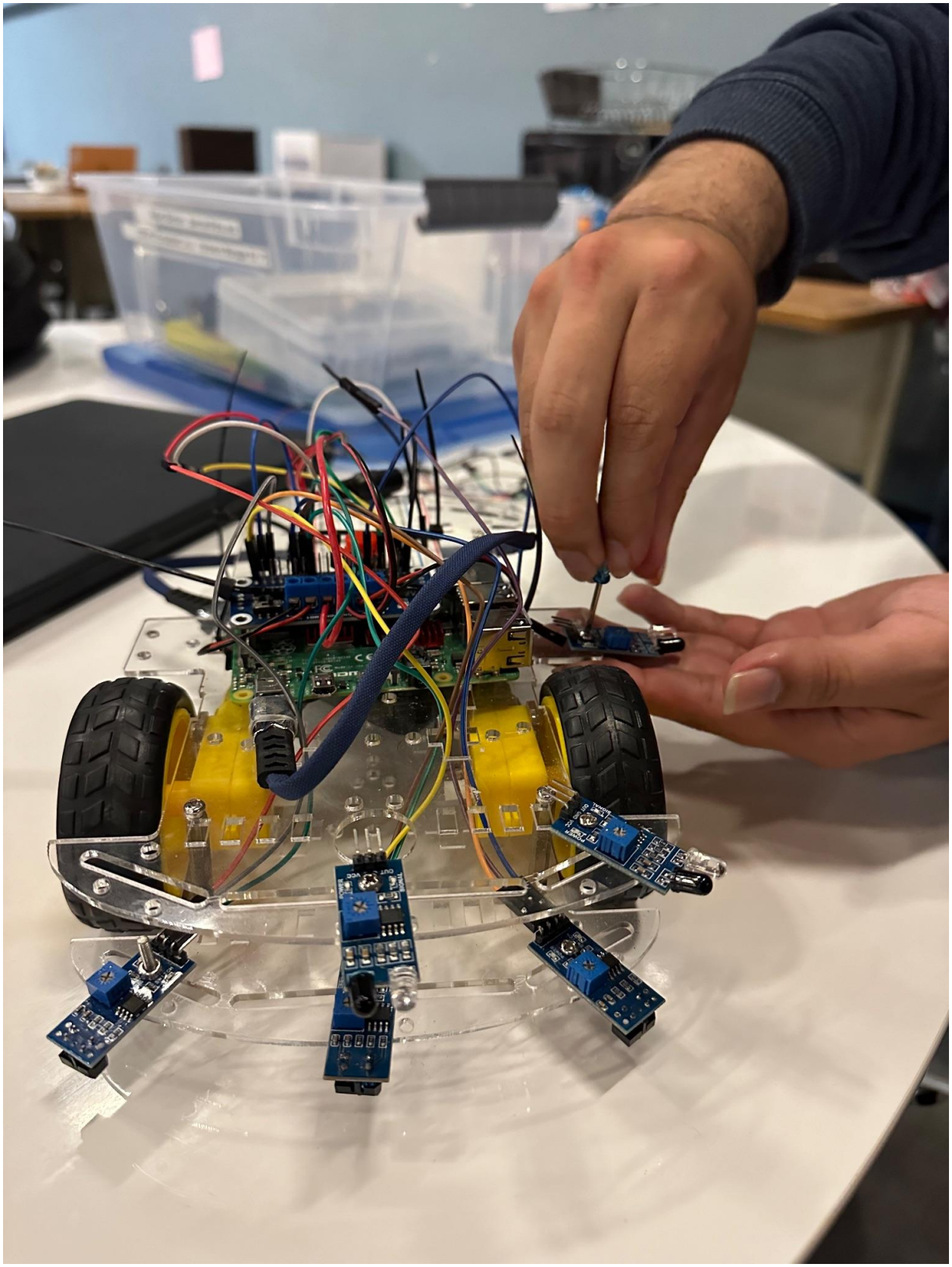


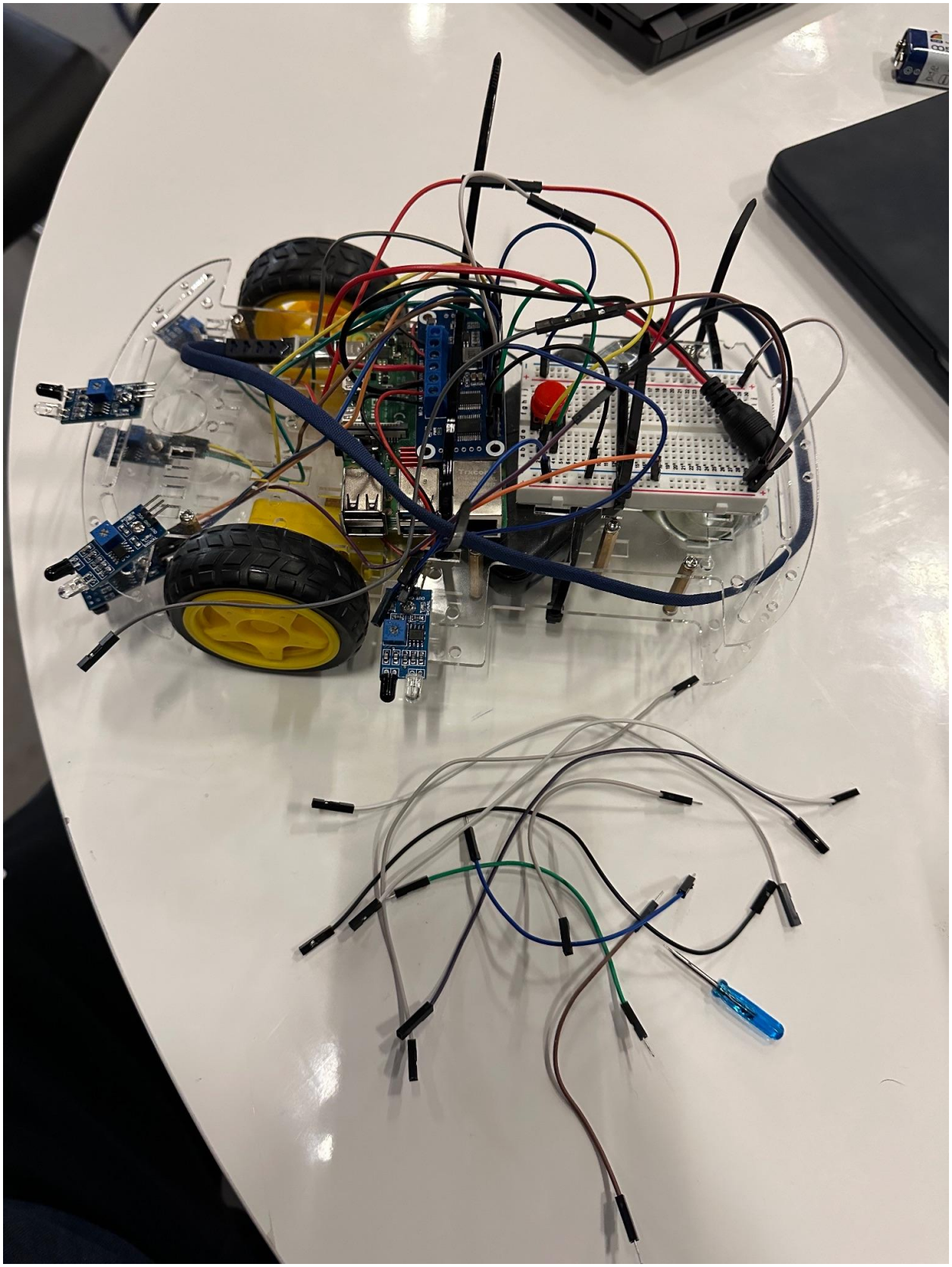


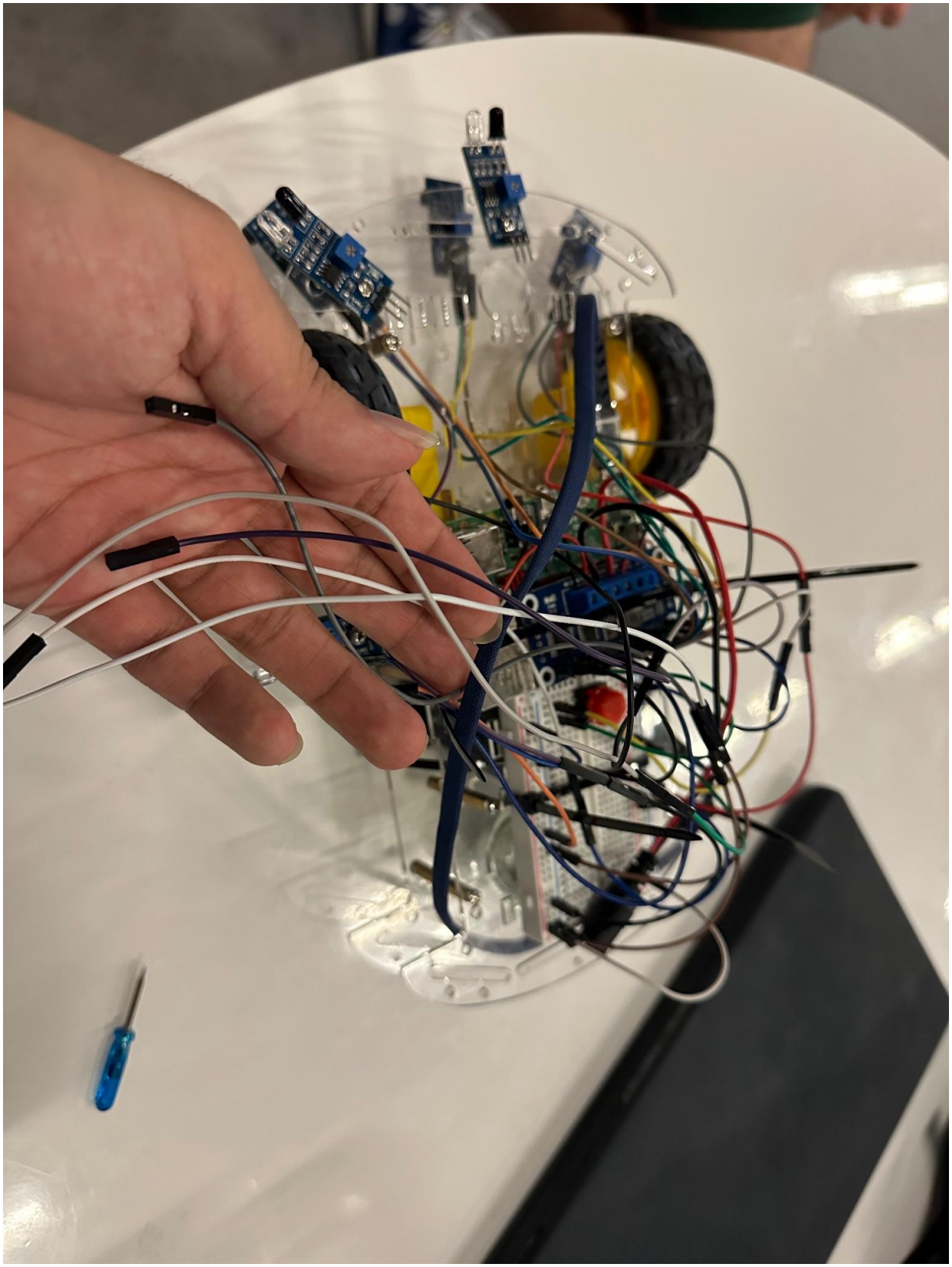


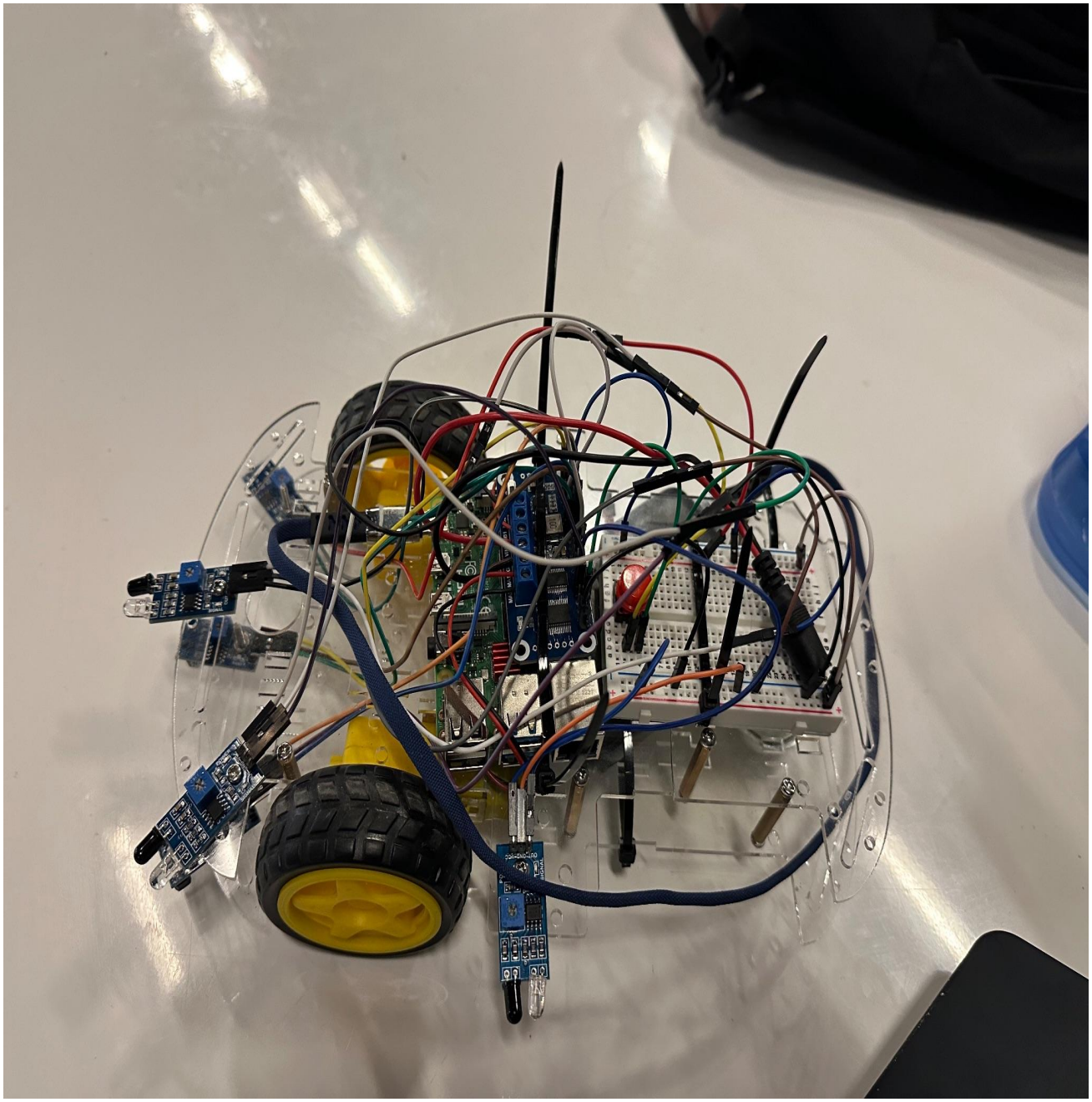






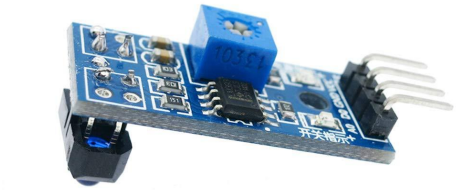




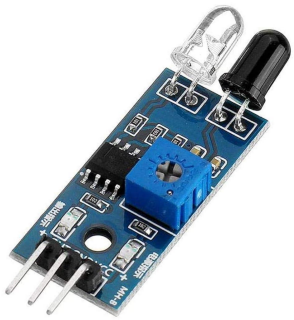


Parts / Sensors used

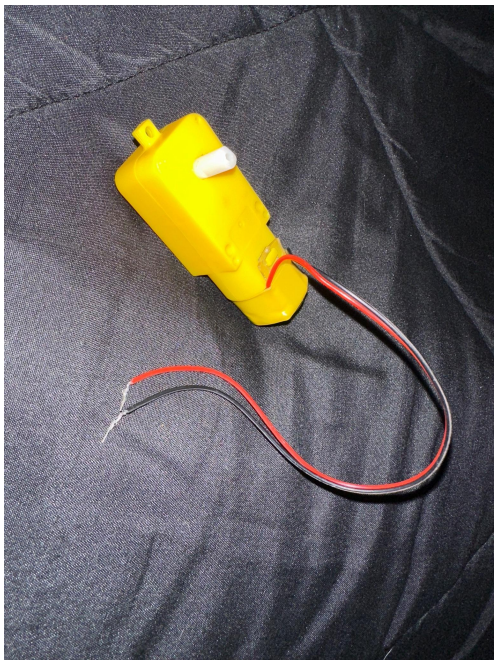
- 3x Line Sensor with TCRT5000 Reflective Optical Sensor



- 3x IR Infrared Obstacle Avoidance Sensor



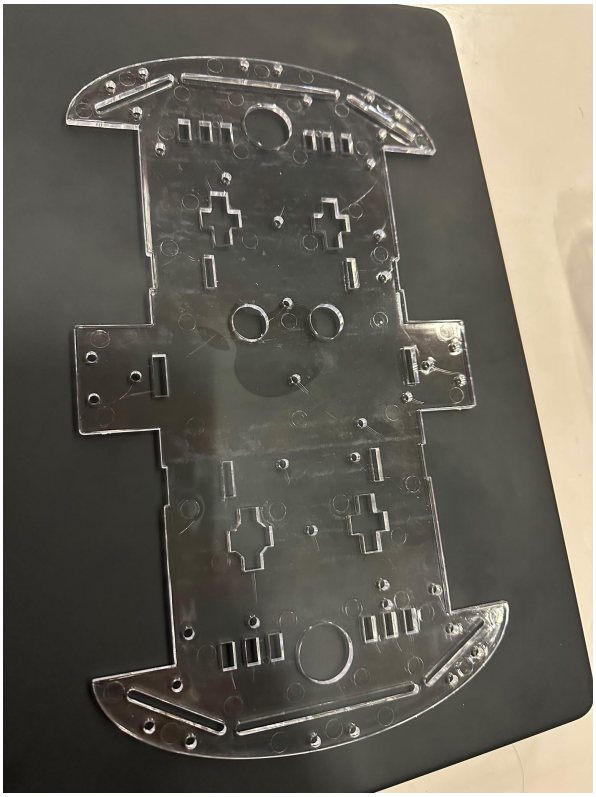
- 2x Motors



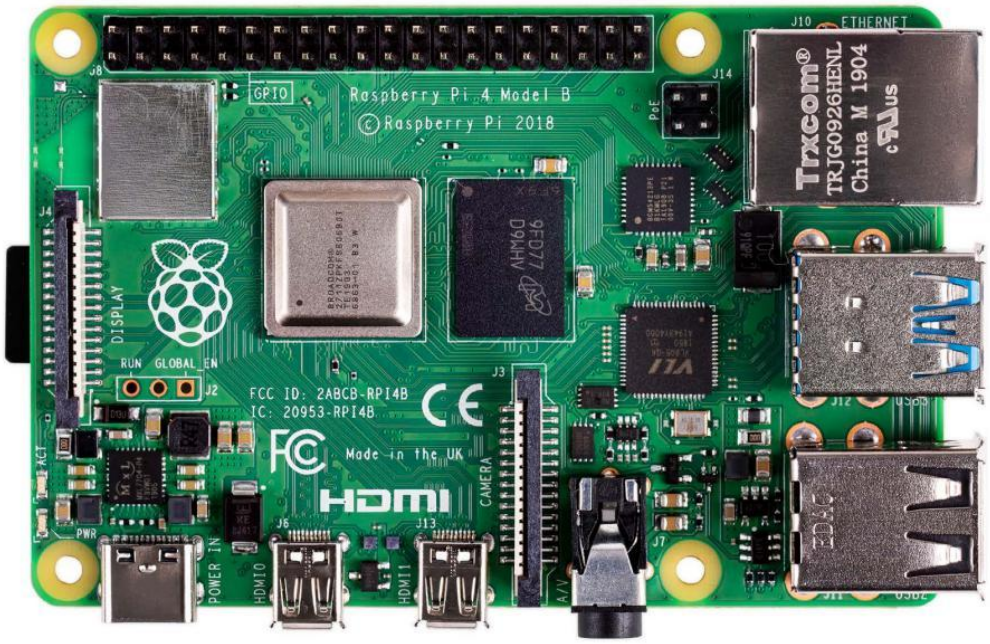
- 2x Wheels

- 1x Rollerball Bearing

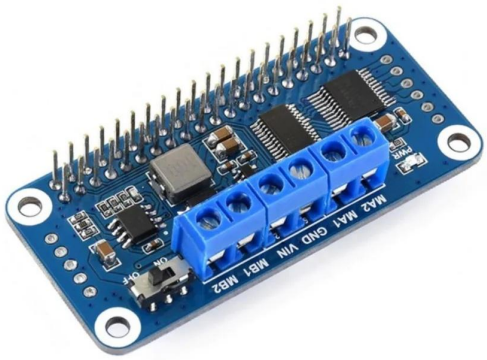
- 2x Structure plates



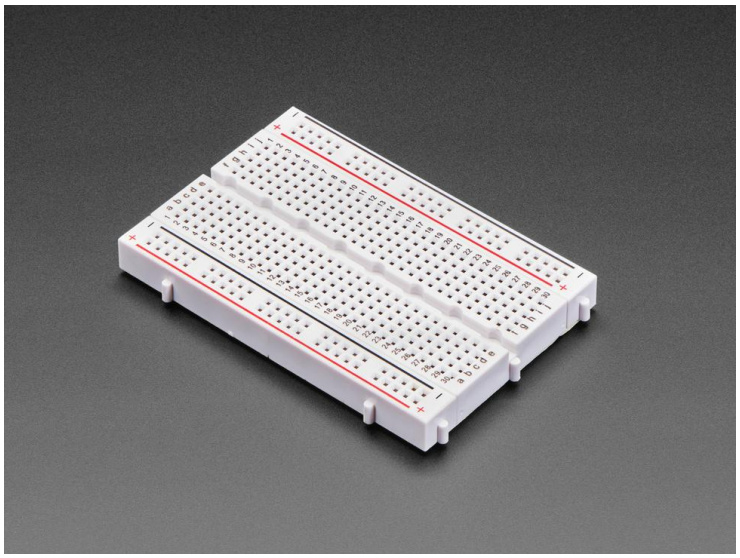
- 1x RaspberryPi4



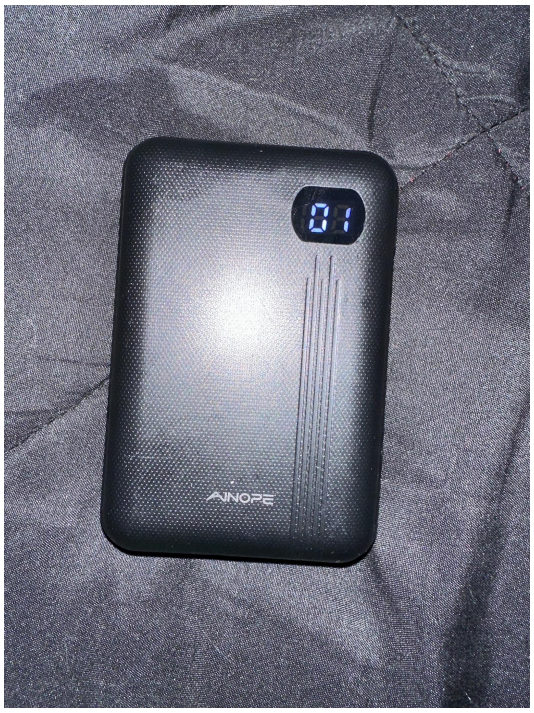
- 1x WaveShare Motor Drive HAT



- 1x Breadboard



- 1x Portable Battery



- 20+ Jumper cables

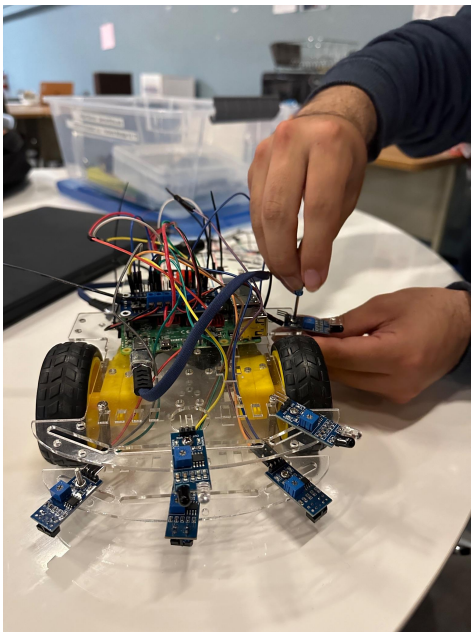
- 20+ Screws

- 12+ Nuts

- 3x Zip ties

How was bot built?

The bot was built by first viewing all of our options to use. We inspected everything given such as the Line sensors, the IR sensors, the Sonar sensors, the wheels, the motors, and the plates. Following viewing these items, we decided what we were going to use. We decided on the Line sensors, a Sonar sensor that would rotate, 4 motors, four wheels, and two structure plates. We first put two wheels on and instead of going with 4 wheels and 4 motors, we decided we would try to use two wheels, two motors, and a rollerball bearing. We asked Bierman for this part as it was not included in the original set. Once we got this we put the ball bearing on and started to put the Line sensors in the front of the car, one in the middle and the other two on the left and right side. Following this we put 4 main Shexton nuts, two in the back and two in the front with one on the left and one on the right. This would provide support for the structure plate to be on top and create a palace to put things in between the two structure plates. One we had both structure plates on and we positioned the Raspberry Pi4 towards the center of the top structure plate. Following this we put the Motorhat on top of the Raspberry pi4 and ensured that the connection was working by testing it on one of the Line sensors. Upon doing a few test runs, we decided that it would be a good idea to add a button to start the car so that we could run the program and not have to worry about the car flying off something right away after starting the program. So we added a breadboard behind the Raspberry Pi4 and the Motorhat. We then positioned the button and connected it to the Raspberry Pi4. Once this was done and we had the breadboard connected we decided to connect the rest of the Line sensors. Once it was connected we tested the logic for line following and got it working. Next we moved on to adding the Echo sensor which we had trouble with. We didn't know how to mount it as we wanted it to rotate via servo that we obtained from Bierman. We got an attachment from Bierman for this and tried testing that the Echo would work properly. After some testing we couldn't get it working properly so we ended up switching to using an IR sensor. We put this IR sensor in the front middle of the top structure plate and connected it to the breadboard and the Raspberry Pi4. After some test runs we decided to add two



more IR sensors to increase the amount of things it can see, essentially for more information. When we got this working it was just up to fine tuning.

Libraries/Software Used

#include <stdio.h>

Reference: Part of C standard library,

https://www.ibm.com/docs/en/zos/3.1.0?topic=files-stdioh-standard-input-output

#include <stdlib.h>

Reference: Part of C standard library,

https://www.ibm.com/docs/en/zos/3.1.0?topic=files-stdlibh-standard-library-functions

#include <pthread.h>

Reference: Part of IEEE POSIX standard,

https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/pthread.h.html

#include <unistd.h>

Reference: Part of IEEE POSIX standard

https://pubs.opengroup.org/onlinepubs/7908799/xsh/unistd.h.html

#include <pigpio.h>

Reference: pigpio C library,

https://abyz.me.uk/rpi/pigpio/

#include <signal.h>

Reference: Part of C standard library

https://pubs.opengroup.org/onlinepubs/9699919799/

#include "motor.h"

Reference: Custom made based on this design

https://www.waveshare.com/wiki/Motor_Driver_HAT

#include <stdbool.h>

Reference: Part of C standard library

https://en.wikibooks.org/wiki/C_Programming/stdbool.h

#include "PCA9685.h"

Reference: The PCA9685 is an I2C-bus controlled 16-channel LED controller optimized for LCD Red/Green/Blue/Amber (RGBA) color backlighting applications.

https://www.waveshare.com/wiki/Motor_Driver_HAT

#include "DEV_Config.h"

Reference: Hardware underlying interface for the motors

https://www.waveshare.com/wiki/Motor_Driver_HAT

#include <time.h>

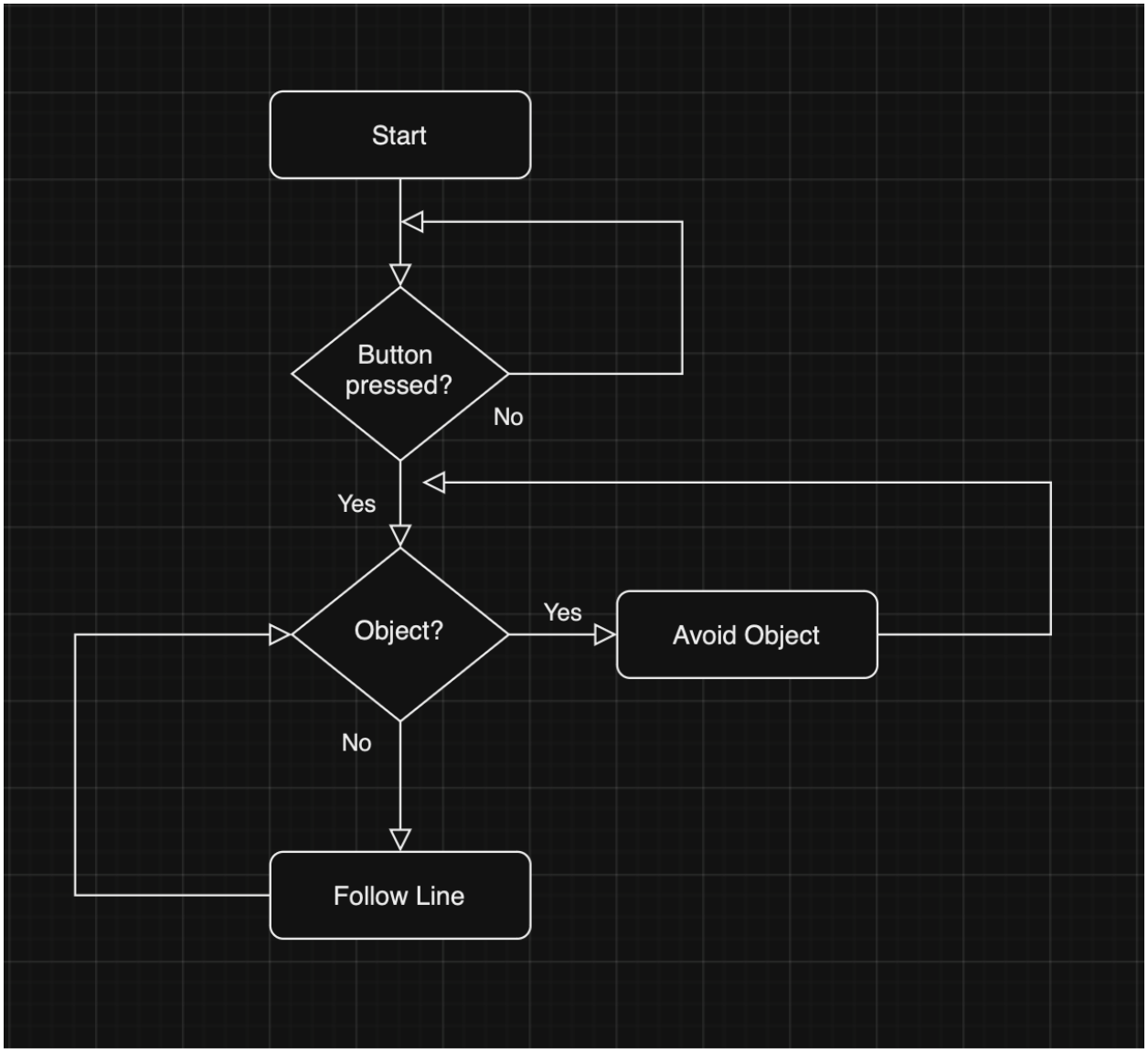
Reference: Part of C standard library,

https://pubs.opengroup.org/onlinepubs/7908799/xsh/time.h.html

#include "assignedPins.h"

Reference: Custom made just to separate pin assignments for some clarity

Flowchart of Code



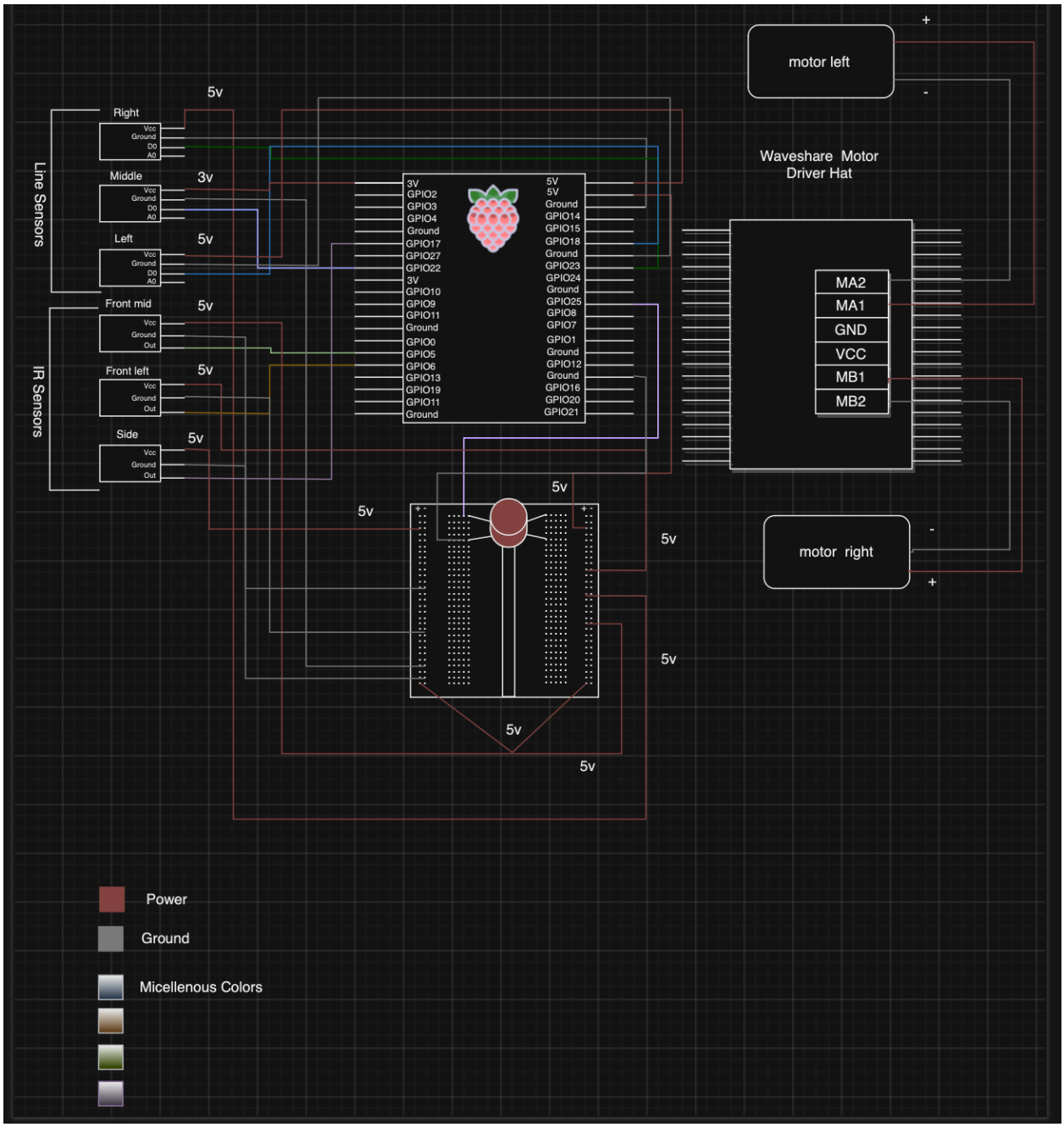
Pin Assignments

Line sensors(left, middle, right) : 18, 22, 23

IR sensors: (left-side, left-front, front): 17, 6, 5

Button pin: 25

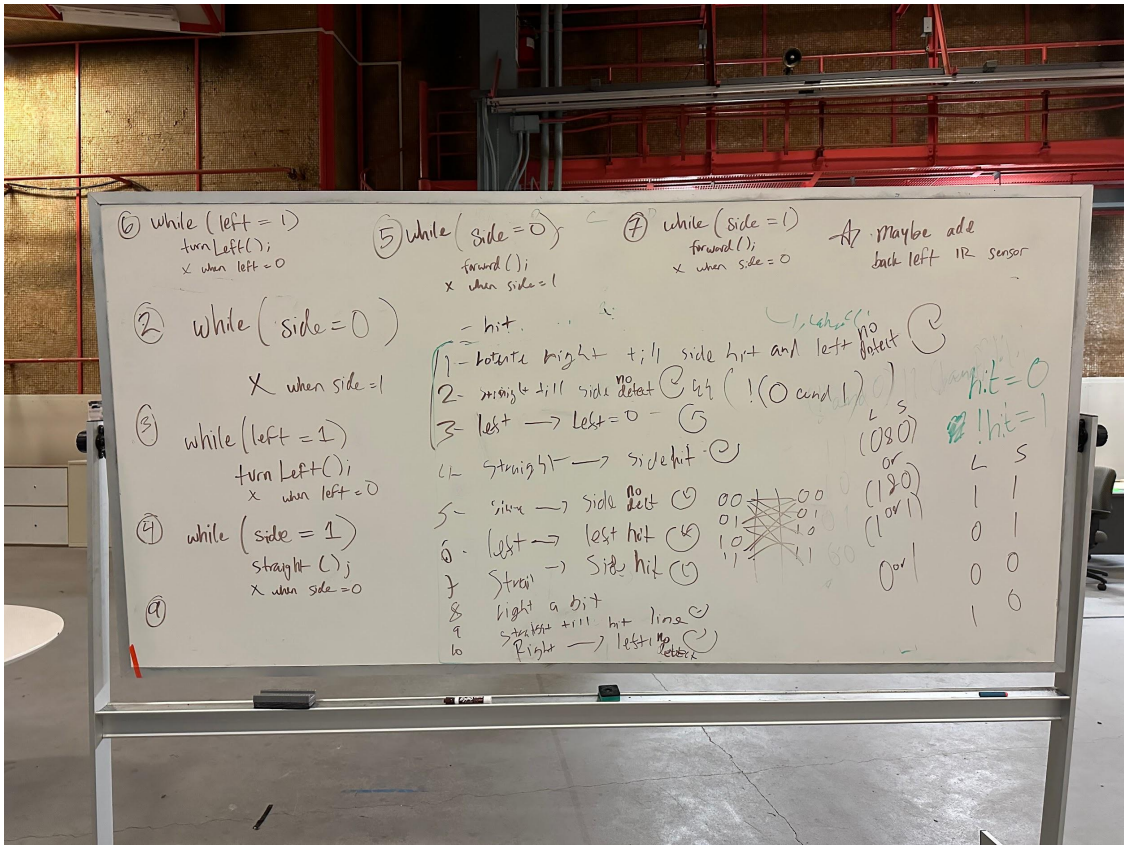
Hardware Diagram



What Worked Well?

1. A problem that seemed to be prevalent amongst other groups, was the issue of how to physically attach the sensors to the car. Some groups went to great lengths in order to accomplish the mounting, such as glue, tape, or even zipties. Our team found that the simplest and secure method of attaching the sensors was to simply screw them on. The IR and line sensors have holes where one can place a small screw and fasten it using a nut. This proved to be a great way to ensure that the sensors would not shift or bounce a great deal out of place while the car was in motion.
2. When it became time to think about what code should be written for the line following logic, and object avoidance logic, use of the whiteboard dominated as a major means for planmaking. As a team, through the rough terrain of our disputes, we collectively came up with a way to write our own form of pseudocode that translated our logical ideas into a written form that could then be translated into C. This was especially prevalent with our object avoidance logic. We broke down the problem of avoiding an object into individual steps. We also physically used our car and our hands to simulate manually what the path of avoiding an object would look like. We noted each movement and process of the car as a step. We then used the “story” of how the car would go around an object, to then write

our conditional statements.



3. Using IR sensors instead of the ultrasonic sensors was a complete game changer in terms of obstacle avoidance. Problems with the ultrasonic are mentioned in the issues section, so we'll focus on what exactly worked best with the IR. The IR sensors bring a much simpler, ON/OFF functionality when it comes to detecting things in front of it. The IR sensors also allowed us to not focus on the timing aspects of reading input with the ultrasonic sensors. IR sensors can also be adjusted in terms of the minimum distance required for object detection, which allows greater flexibility in terms of movement patterns and maintaining a locked point of reference for where the object is at all times, in relation to the car position.

ISSUES

1. When we first got our kit, we wanted to test out the working conditions of the components as well as the Raspberry Pi 4. Most of the components worked but when we tried to test the motors using a simple c program to see if the motors worked we couldn't get them to work at all. At first, we thought it was a problem with the motors but after using the Raspberry Pi from our individual kits to test the motors, they worked just fine. So we decided to reflash the image onto the SD card for the Pi 4 because when we first got the card, it was already flashed and we got it to boot on the Pi 4. After flashing the image onto the card and having it successfully booted up on the pi, we tried again but it still didn't work. The motors worked, the program worked, so what was the problem? After doing some research, it turned out that we forgot to enable the I2C protocol on the pi 4, which we needed for the HAT to power the motors.
2. After assembling the car, there were some problems with positioning the line sensors. The distance where the line sensors could pick up was too short and we couldn't attach the line sensors to places where they could detect the lines properly. We tried spreading them apart but that did not work either. We had to find a solution where two of them were close while the other was separated to work properly.
3. Another problem we had was with the obstacle sensor. Originally we were going to use the echo sensor on a servo to avoid the object. This was a great idea but not so much in practice as we had trouble figuring out how we were going to mount the sensor to the car. With the movement of the servo it was hard to find a spot. But when we tried to find a spot, we could not get it to work properly. By this I mean when we were testing it reading the distance it would only take the first reading and printing out once. This was due to it not being in a loop and only reading once. Put this in a loop to check and fix this issue. But after this we had an issue where it would read for a few moments but after words it would get stuck on a reading and only print out the reading which was very confusing to us. To fix this issue we switched to the IR sensor.
4. Yet another issue was with the IR sensor. We were attempting to only do it with one IR sensor and a rigid strict routine. While doing this we found issues with the motors. We already knew the motors had different power outputs even with the same output being sent through but it was made more apparent during the avoid sequence. It would turn two

far out causing it to go off track and miss the object completely or crash into the object. So we tried adjusting the power to the wheels to fix this. We tried only turning 1 wheel(rotating), or both wheels(turning) to fix this issue but were not able to fix it to be properly aligned and avoid the object correctly. So to fix this issue we ended up adding two more sensors, making it have 3 sensors in the front.

5. Another issue we had was getting on the same page with the logic. When we were going over the avoidance logic it got confusing as we had different ideas of how to do it. When we settled how we were going to do it, we still had to create the logic where it got even muddier. We were confusing eachother with what we meant, even though we were making perfect sense to ourselves. Communicating the logic to our group mates was a bit confusing but after doing it for a while and not giving up, we were able to successfully communicate the logic we had in our heads to our group mates and were able to create a avoidance logic that was able to work.
6. Yet another issue we had was how to work the code together. As we would primarily work directly on the pi itself, we did not want to push too much to the github so we would be able to merge it easier with the updated code in the pi. But we also tried working on the code without realizing that we forgot to pull code from pi to github which caused some confusion and a bit of extra work but we were able to figure it out.
7. One of the biggest issues we had was with the car actually going straight. Whenever we had it go straight perfectly it did not have enough speed to go into the turn correctly and vice versa. So we chose to prioritize turns as those are most difficult to deal with and could cause more issues.