CAR PROJECT
CSC 615

Team: ControlAltDelete
Github:

https://github.com/CSC615-2024-Spring/csc615-term-project-anthonySilva23
Video: https://youtu.be/r7j7xK1Ynzo

Anthony Silva: 922907645 : anthonySilva23

Pedro Grande: 921149265 : PFGrande

Rafael Sant Ana Leitao: 922965105: Rafael-Leitao

Tyler Hsieh: 920216320: Tyler9648

Task Description

This project aims to design, build, and program an autonomous car that follows a pre-defined line on the ground and can detect and navigate obstacles. The project integrates mechanical assembly, electronic circuitry, sensor deployment, and software programming to create a fully autonomous robotic vehicle using a Raspberry Pi 4 as the main processing unit.

Objectives

1. Mechanical Assembly: Construct a robotic car using a provided chassis kit, ensuring all mechanical parts including motors, wheels, and the sensor mount are securely assembled.

2. Sensor Integration: Implement and calibrate multiple sensors to enable the car to detect and follow a line and to identify obstacles in its path.

3. Software Development: Develop software to process sensor inputs and control motor outputs, implementing a PID controller for precise movement control.

4. Testing and Optimization: Conduct comprehensive tests to refine the car's line-following and obstacle avoidance capabilities. Optimize the software and hardware setup based on test results.

Specific Tasks

Building the Robot

- Assemble the chassis, motors, and wheels.

- Install and securely mount the line tracking sensors and obstacle avoidance sensors on the front of the vehicle.

- Mount the ultrasonic sensor on a servo motor for dynamic scanning.

Sensor Setup

- Configure and calibrate five line-tracking sensors (TCRT5000) and an IR obstacle avoidance sensor.

- Integrate an HC-SR04 ultrasonic sensor for enhanced obstacle detection.

Software Development

- Utilize the PIGPIO library for precise GPIO pin management.

- Implement multithreaded sensor data handling to read sensor inputs in real-time.

- Develop a weight style algorithm in `steering.c` to adjust the steering based on the line alignment data from sensors.

- Created a motor, and sensor library that run independently, that can be easily utilized by other car functions

System Integration and Testing

- Ensure all components are properly communicating with each other, with real-time sensor data influencing motor control decisions.

- Adjust PID controller parameters to improve the responsiveness and accuracy of the car's steering mechanism.

- Perform field tests to validate the functionality of line following and obstacle detection, making iterative improvements based on test outcomes.

Documentation and Reporting

- Maintain detailed documentation of the building process, software development, and testing phases.

- Compile a final project report that outlines the design process, challenges encountered, solutions implemented, and testing results.

- Prepare and deliver a presentation summarizing the project goals, methodology, achievements, and learnings.

Deliverables

- A fully functional autonomous line-following car equipped with obstacle detection, and smooth turning.

- Comprehensive project documentation, including assembly instructions, software code, and testing protocols.

- A final project report and presentation detailing the entire project scope and outcomes.

Timeline

Set deadlines for each major phase of the project, including milestones for assembly completion, initial software functionality, integration testing, and final presentation.

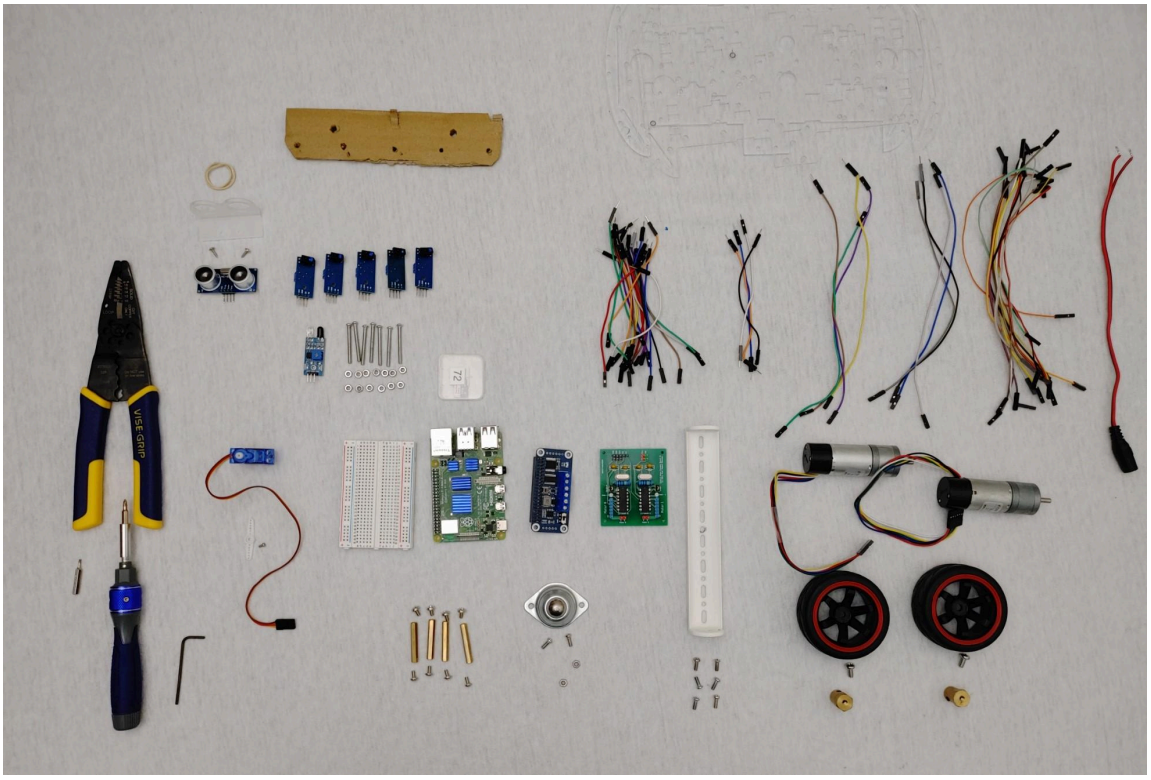
Resources Required

- Hardware: Raspberry Pi 4, motors, chassis kit, TCRT5000 sensors, IR sensor, HC-SR04 ultrasonic sensor, motor driver HAT.

- Software: PIGPIO library, custom software scripts for sensor and motor control.

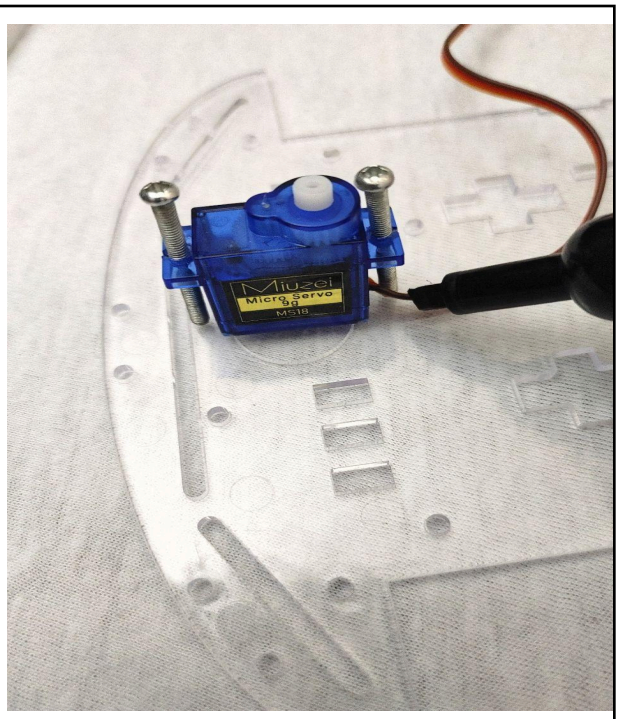
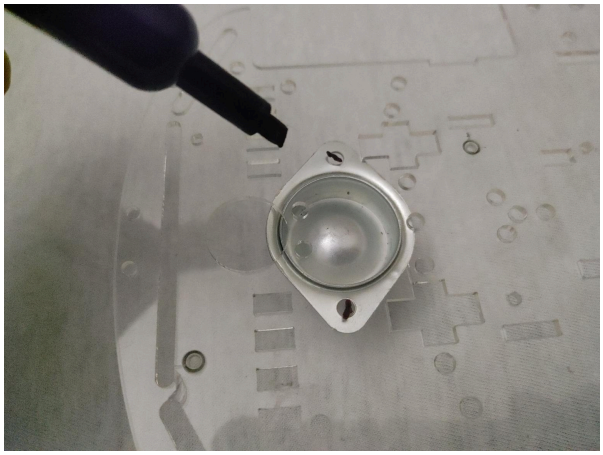
- Miscellaneous: Tools for assembly, testing spaces, and materials for creating a track for line-following tests.

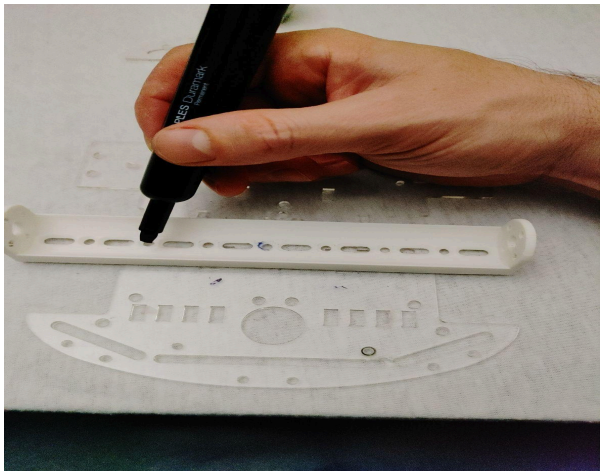
Building the Robot



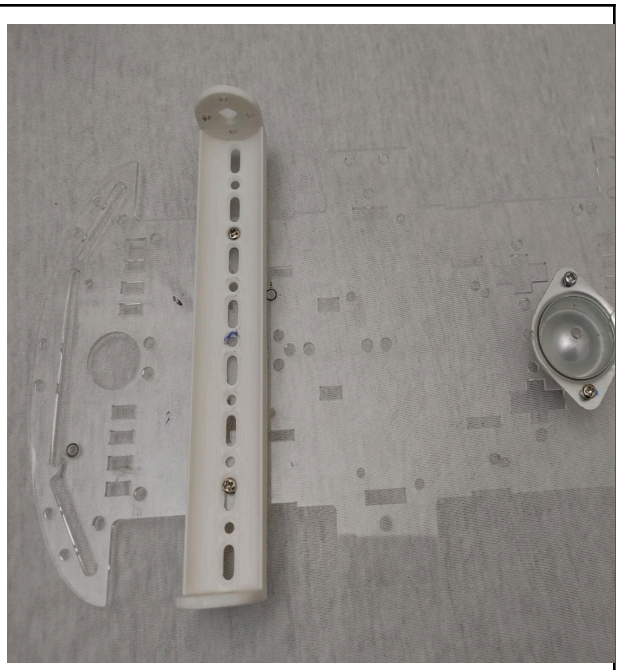
Using the list of parts, follow these directions to rebuild the bot. The picture above shows the bot broken down into individual parts and components. The image also includes the tools needed to assemble the bot, excluding a drill and marker. Note: the tutorial is divided into sections and doesn't necessarily need to be followed in order.

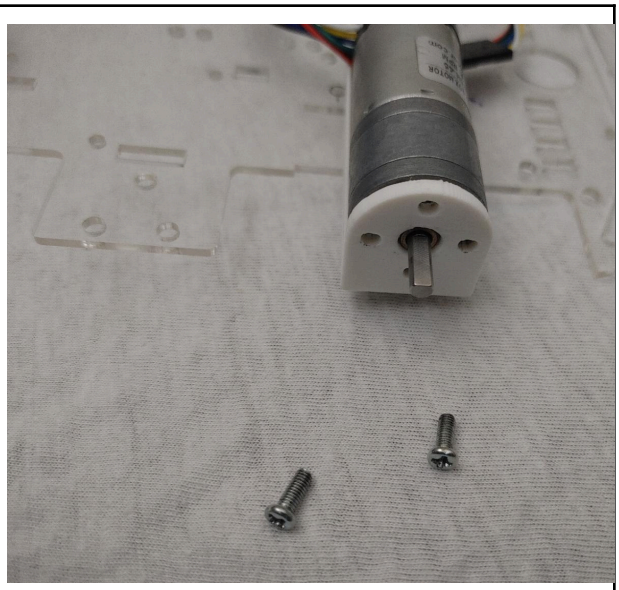
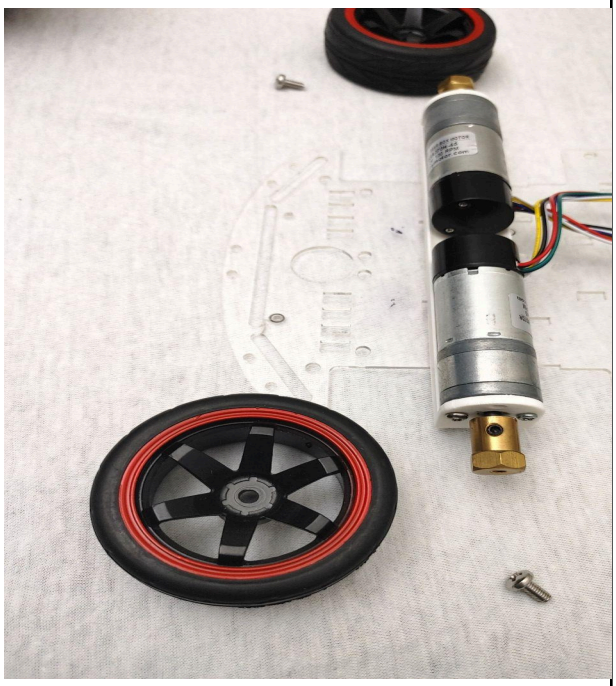
1. Drilling holes to mount the motor cradle, front wheel, and servo

Place bolts in the servo and position on the upper chassis where you want the servo to be mounted. With a marker, mark around the bolts and drill the holes.	
Position the Swivel Ball Caster under the low chassis and mark the holes in the frame. Then, drill holes in the marked positions.	

Position the motor harness evenly at the opposite end of the low chassis and mark two holes. Then, drill the holes for the harness.	

2.Installing the motor and wheels

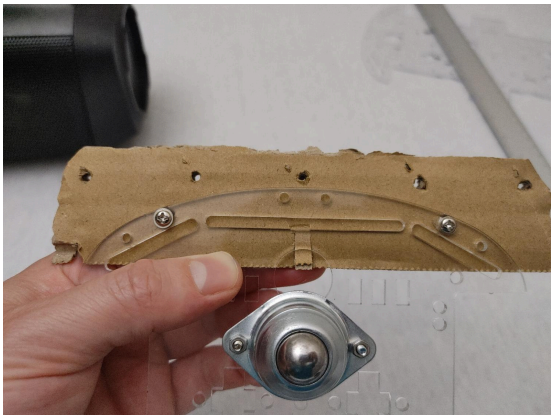
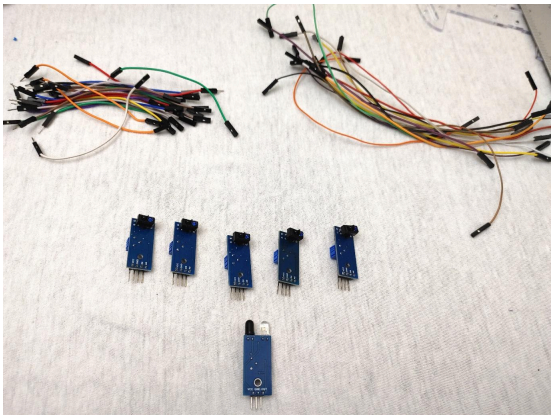
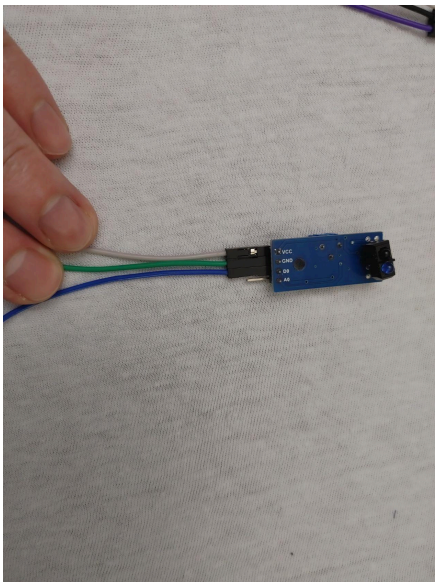
Using bolts and screws, mount the motor harness on top of the chassis and the front wheel below. The result should look like this. Note: Mounting the chassis underneath elevated the rear of the bot to a high level. Also, the servo can be mounted later as it can get in the way during the bot's construction.	

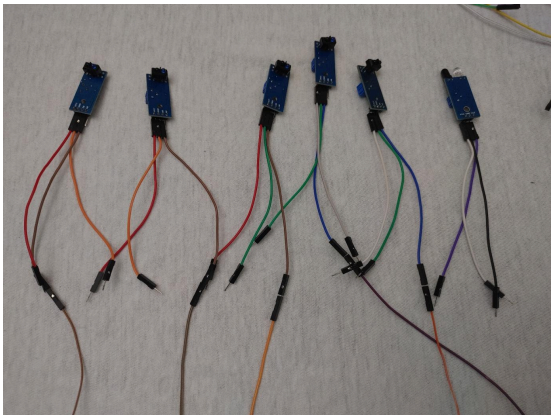
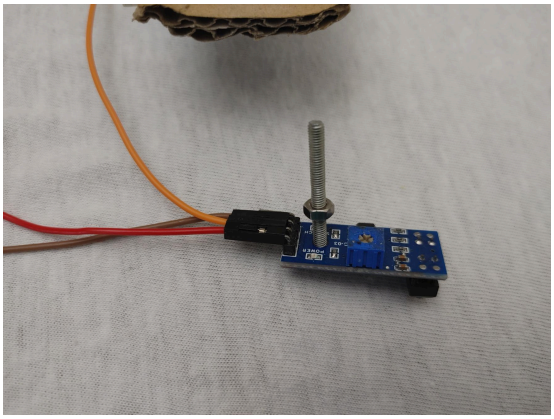
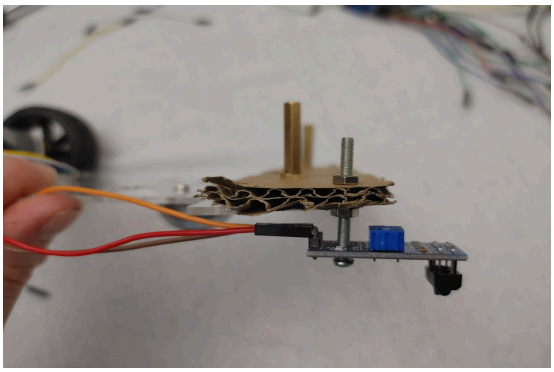
Place the motors in the harness and align the holes for the bolts. Then screw the bolts into the motors.	
Attach the wheel mounts and tighten them with a 2mm Hex Key. Make sure to leave an even gap on both sides between the motor harness and wheel mounts.	

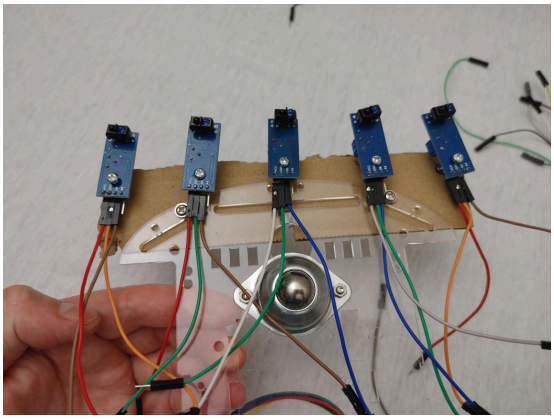
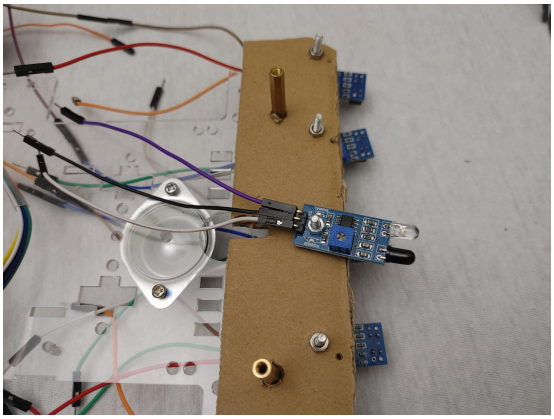
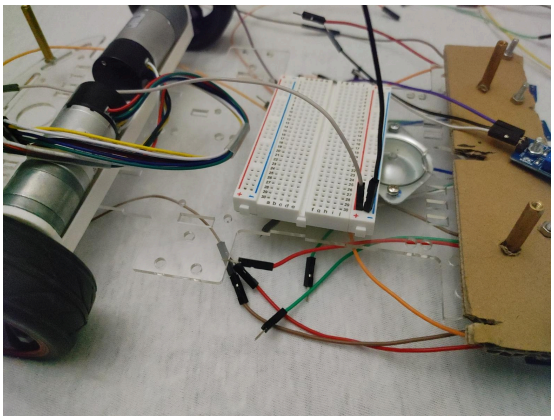
The result should look similar to this. Note: having the wheels and motor mounted makes the rest of the construction easier.	

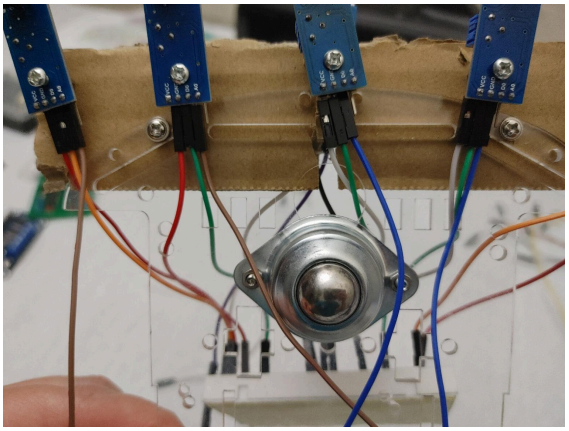
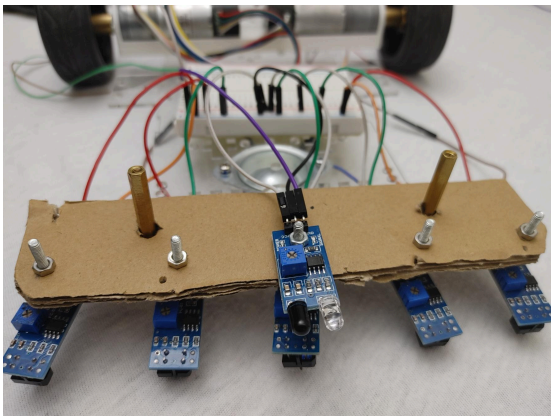
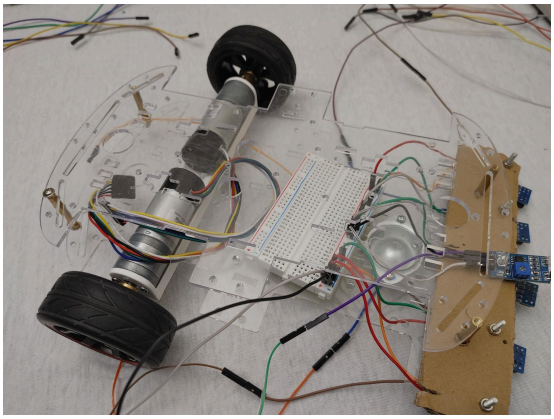
3.Installing the line and Obstacle Avoidance sensors

Cut 16cm x 3cm piece of cardboard, mark 5 point evenly spaced, and puncture holes for the line sensors.	

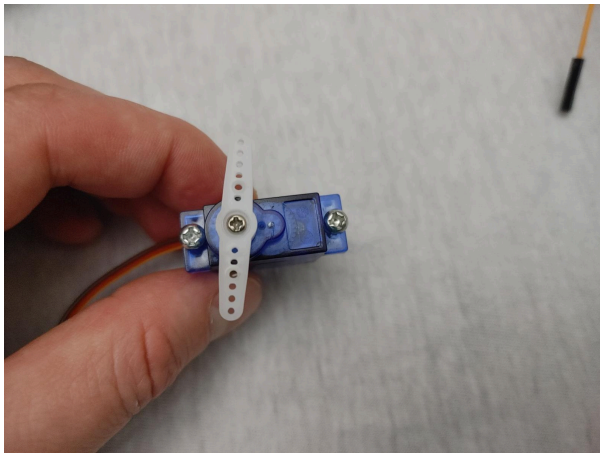
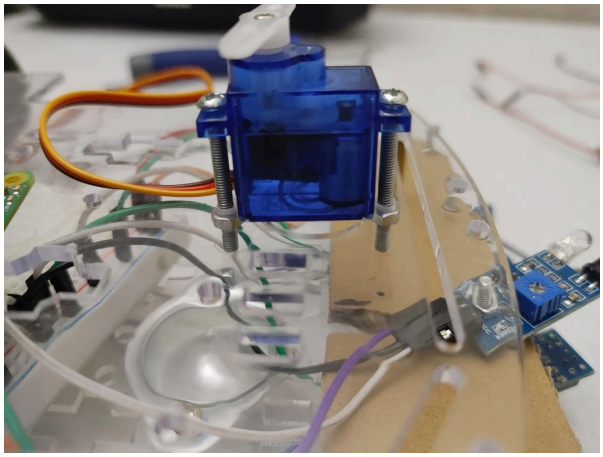
Position the line sensor mount towards the front of the vehicle and puncture two holes through the pre-drilled holes in the chassis. Then attach the L30 spacers with the M3*8mm screws on the front and back of the vehicle.	
Gather your wires, 5 line track sensors, and one IR Infrared Obstacle Avoidance Sensor.	
Then, wires are attached to the VCC, GND, and D0(out pin) pins. Make sure the out pin wires are double in length.	

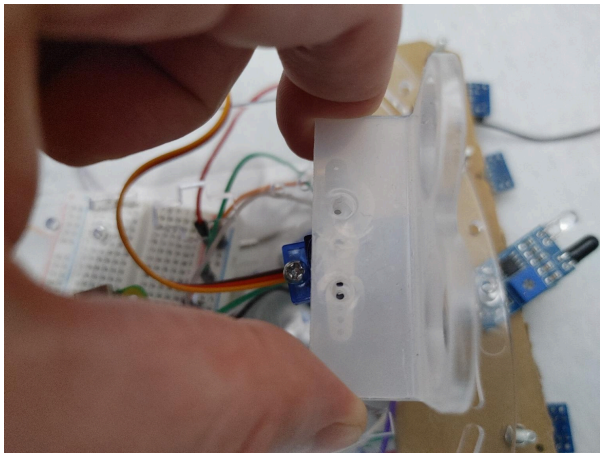
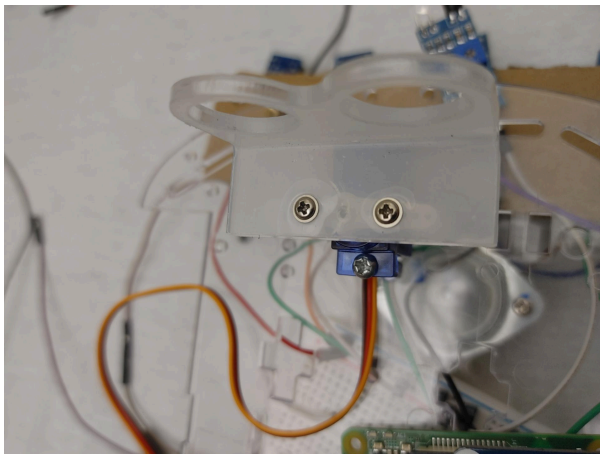
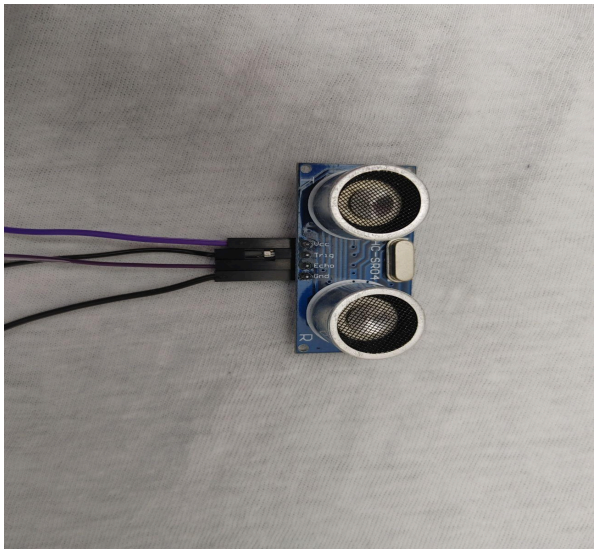
Should look like this.	
Then take 1 M3*30mm bolt and slide it through the sensor and screw an M3 nut half way down the bolt.	
Slide the bolt through the mount and tighten a fasten with another M3 nut on top inorder to keep the sensor in place and adjust the height.	

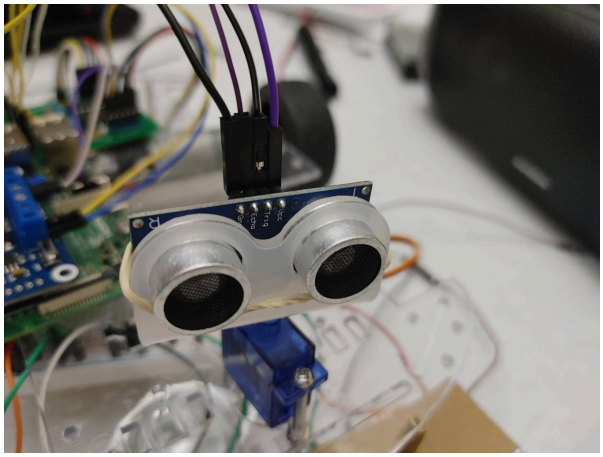
Repeat for all line sensors.	
Attach the Obstacle Avoidance Sensor to the center bolt and secure it with a M3 nut.	
Next, position the short breadboard on the chassis behind the line sensors.	

Connect all power and ground connections to the breadboard for the line sensors. Use the precut holes in the chassis and slide the wires through them to help keep the sensors facing straight ahead.	
Do the same for the IR Infrared Obstacle Avoidance Sensor.	
Install rear L30 spacers lower chassis and place the upper chassis on the vehicle.Install a bolt to one of the L30 spacers .Do not tighten so you can easily move the chassis around when connecting all the wires.	

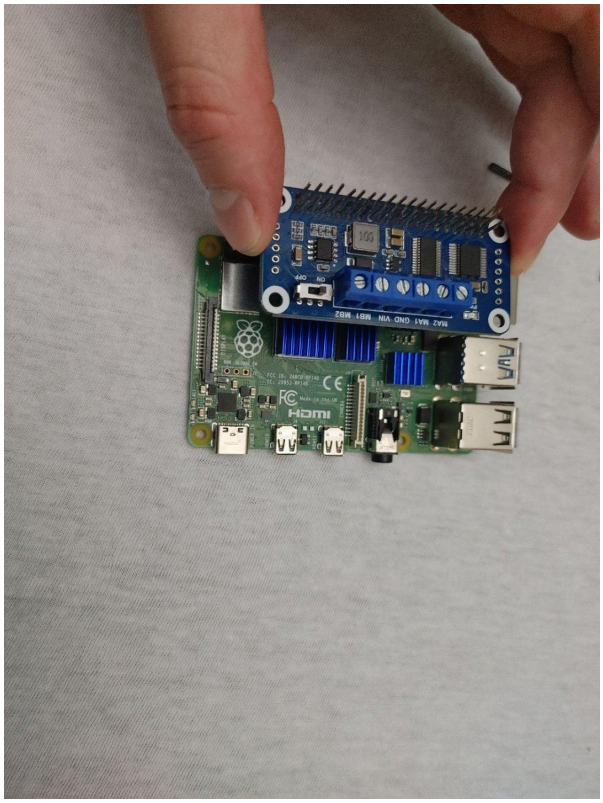
4. Installing the servo and Ultrasonic Module HC-SR04

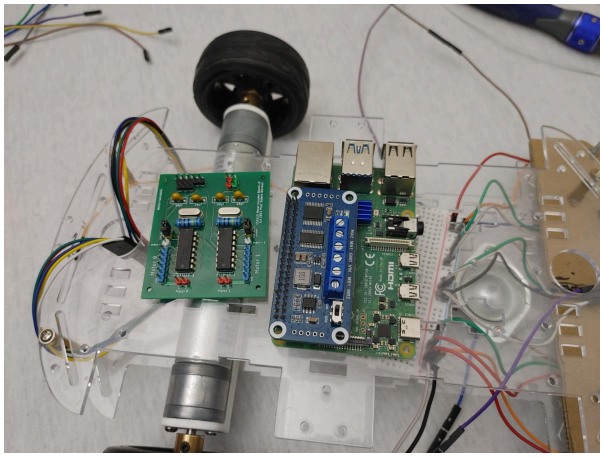
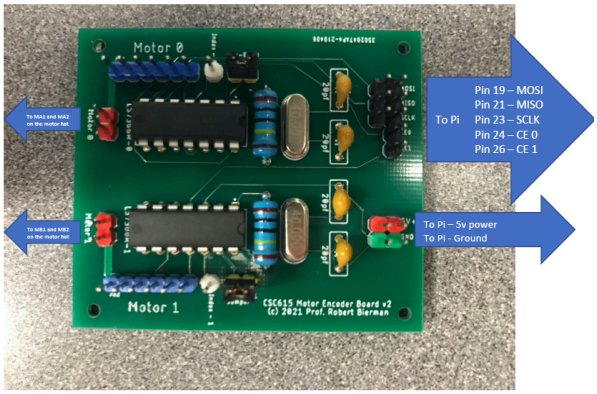
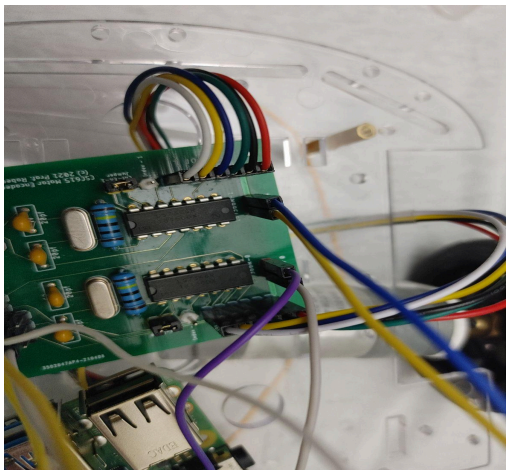
Attach the 32mm servo arm to the top mount using the accessories kit.	
Mount the servo the upper chassis with 2x M3 nuts using the cut holes from step 1.	

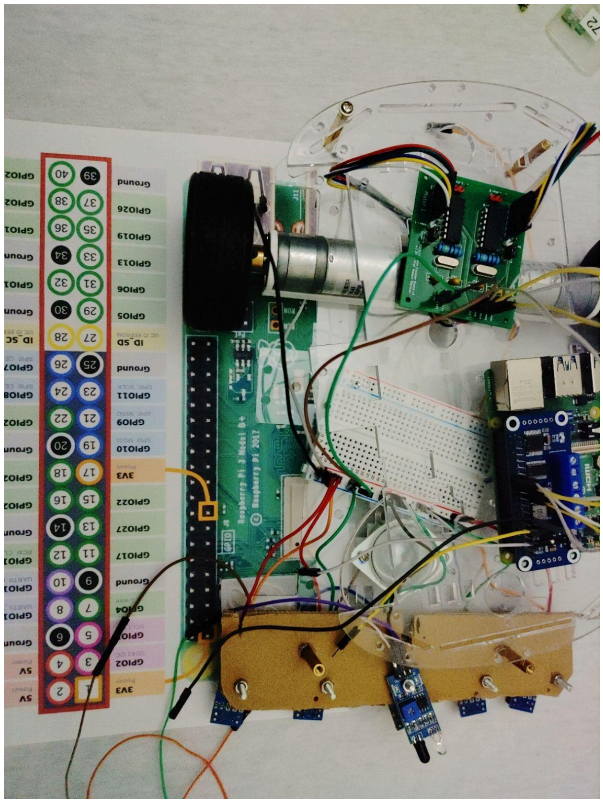
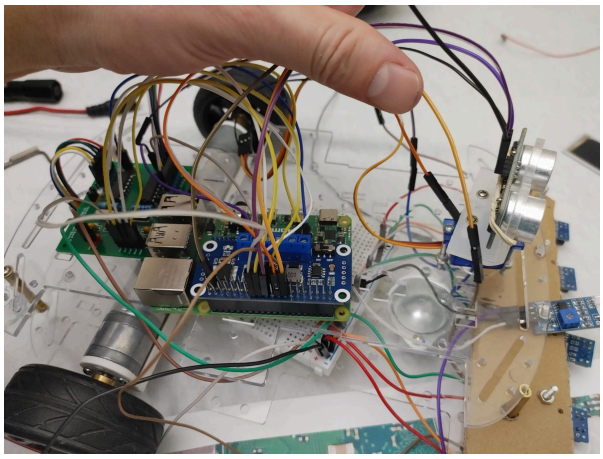
Align the Ultrasonic Module mounting bracket evenly with the servo arm.	
Use the remaining two screws from the kit to install the mounting bracket.	
Connect wires to the pins on the Ultrasonic Module.	

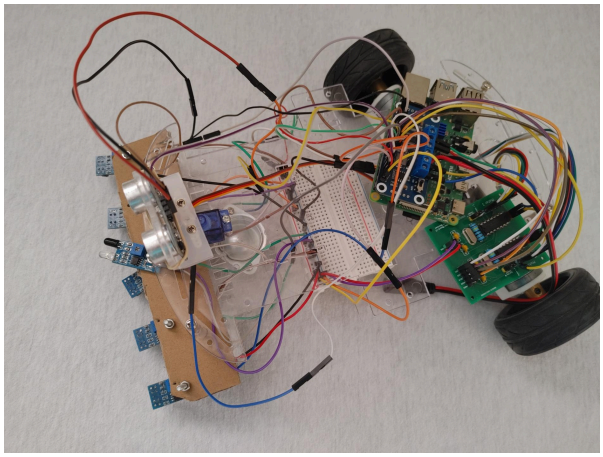
Fasten the Ultrasonic Module to the mounting bracket with a rubber band.	

5. Wiring everything together

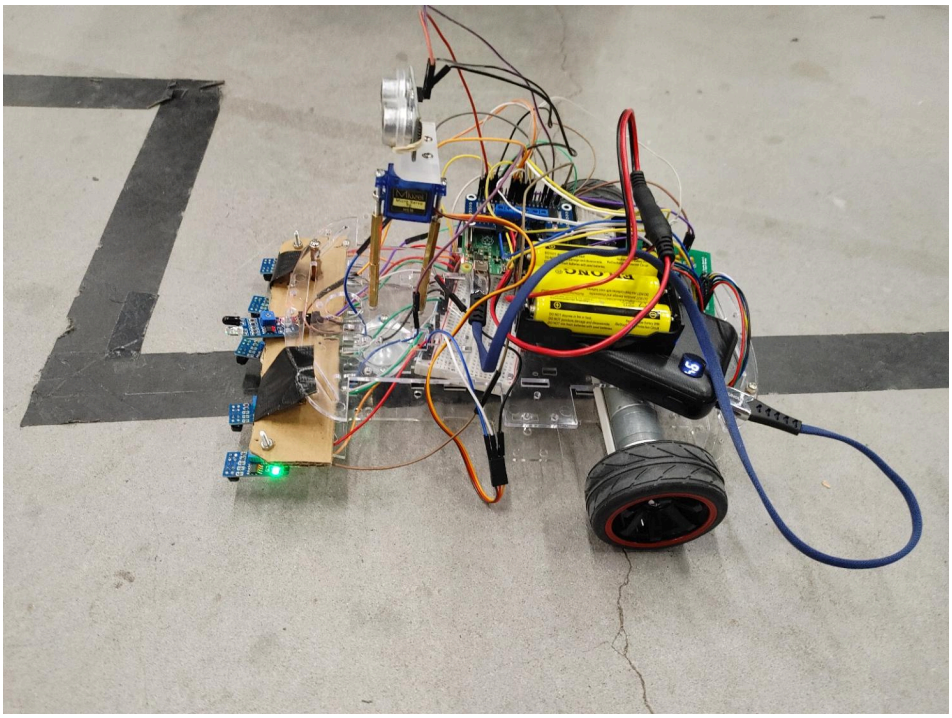
Attach the Motor Driver HAT to the Pi as shown in the image.	

Place the Pi(with attached Motor Driver HAT) and Motor encoder Board on top of the upper chassis.	
Follow the image to connect the motors to the Motor encoder Board and Motor encoder Board to the Motor Driver HAT.	
Should look like this.	

Now connect all the modules to their proper pins. Refer to the pin assignments section of the documentation for this step or follow the hardware diagram. To help this process go smoothly it's recommended to align a printout of the GPIO pins like in the image.	
After completion you should see a mess like this.	

Lastly install the remaining M3*8mm screws to the L30 spacers.	

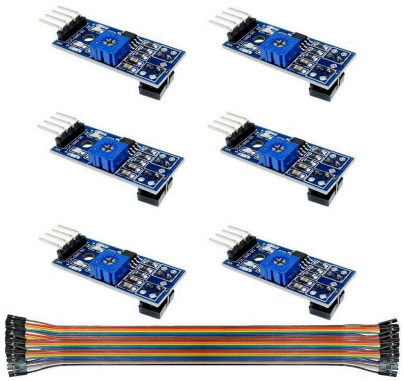
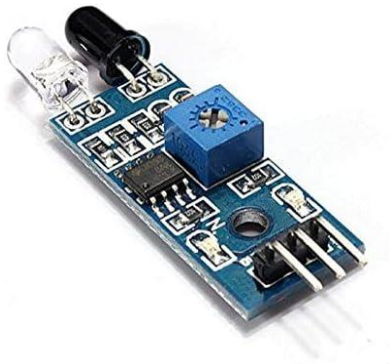
6. (Final step)Attach the batteries

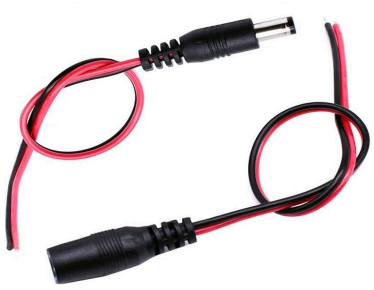
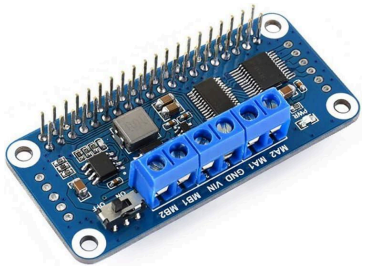
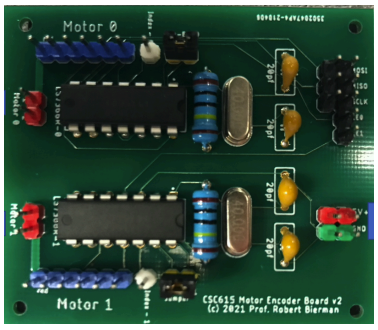


Assuming you have set up the Pi and cloned the repository you can now install the 2 batteries. One with the type C connection to the pi and one using the barrel connector to the driver hat. Then force them onto the upper chassis somehow(good idea to hold them down with electrical tape) and you're ready to race the track.

Parts / Sensors Used

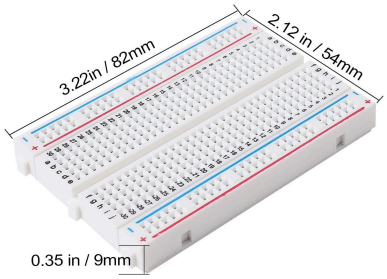
Parts	Image
Perseids DIY Robot Smart Car Chassis Kit Parts used in kit • Chassis (2x) • M3 nuts (8x) • M3*8mm screws (6x) • M3*30mm screws (8x) • L30+6 spacers (4x)	
Raspberry Pi 4 Model B/2GB(1x)	
Raspberry Pi 4b Heatsink(1x)	
SanDisk 16GB Micro SD Class 10(1x)	

Miuzei MS18 micro servo motor kit(1x)	
TCRT5000 Infrared Reflective IR Photoelectric Switch Barrier Line Track Sensor Module w/Cable 3.3V-5V(5x)	
Ultrasonic Module HC-SR04 Distance Sensor(1x) with mounting bracket(1x)	 
OSOYOO IR Infrared Obstacle Avoidance Sensor(1x)	

12 volt Barrel Connector(1x)	
Metal Gear Motor 12V DC High Speed 130RPM Gear Motor with Hall Effect Encoder(2x)	
Waveshare Pi Motor Driver HAT - PCA9685 TB6612FNG Drive Two DC Motors I2C Interface(1x)	
CSC615 Motor encoder Board V2 © 2021 Prof. Robert Bierman(1x)	

4mm Motor Flexible Coupling Connector for Car Wheels Tires Shaft Motor(2x)	
12mm Hex Wheel Rims & Twill Rubber Tires(2x)	
M3x12mm SS Bolts (4x) Use: ● Securing the motor to the bracket. ● Attach Swivel Ball Caster to bottom chassis	
M3 nuts(12x) Use: ● Attach bracket to chassis. ● Attach servo to chassis ● Attach Line Track Sensors and Obstacle Avoidance Sensor to cardboard spacer. ● Attach Swivel Ball Caster to bottom chassis.	

3D printed dual Gearmotor Bracket(1x)	
3 x 18650 Battery Holder with DC2.1 Power Jack(1x)	
18650 Batteries(3pcs) and Charger(1x)	
5/8" Swivel Ball Caster(1x)	

16cm x 3cm piece of cardboard(1x) Later replaced by 16 cm x 3 cm, 3D printed clear plastic strip(1x)	
5 Volt Battery Pack(1x)	
USB C to USB A cable(1x)	
Short breadboard(1x)	

Multicolored Breadboard Jumper Wires with Assorted connection(40x)	
Rubber bands(1x)	
220 ohm Resistor(5x)	

How was bot built

Sensors and components

Motors

We chose two 12V DC Gear Motor with Hall Effect Encoder over the Perseids DIY Car kit motors because we thought we could use the Hall Effect Encoder to calculate our speed and position to assist in maneuvering around the track. Ultimately, through trial and error, we ended up not implementing this functionality and instead manually hardcoded motor speed changes and sleep times to make certain maneuvers around the track. We also opted out of using 4 motors because we thought this would add more unnecessary complexity to our bot.

Wheels
Like the motors we chose to avoid complexity and use the standard wheels instead of the omni directional wheels and attach a swivel ball caster at the front of the vehicle. With this setup we were confident that the bot could be just as effective in navigating obstacles with the use of the ultrasonic Module HC-SR04 Distance Sensor mounted on a servo.

Line Sensors
In our research we found a video(image shown below) of a person who greatly improved their bot performance on a line by using an array of 5 line sensors and implementing a PID (Proportional integral derivative controller) algorithm to keep the bot centered on the line. So we used 5 line sensors but were unable to implement the PID algorithm and wrote a simplified version.

Servo and ultrasonic module HC-SR04 distance sensor
While looking on amazon for additional components that might give our team an advantage we saw a picture of the bot below and believed that a similar arrangement would give us that advantage. We theorized that while navigating an obstacle we could track the bot's distance from the obstacle and make steering corrections to dynamically move around any shape and find the line. After trouble with our first design we elevated the unit with L30 spacers so it'll be less likely for its sound waves to be obstructed by the car.

Obstacle Avoidance Sensor
While testing different situations the bot may face on the track we found a vulnerability when presenting another obstacle while the bot is maneuvering around another one. Since the distance sensor changes direction we needed to add an obstacle avoidance sensor facing forward at the front of the vehicle to check for other obstacles.

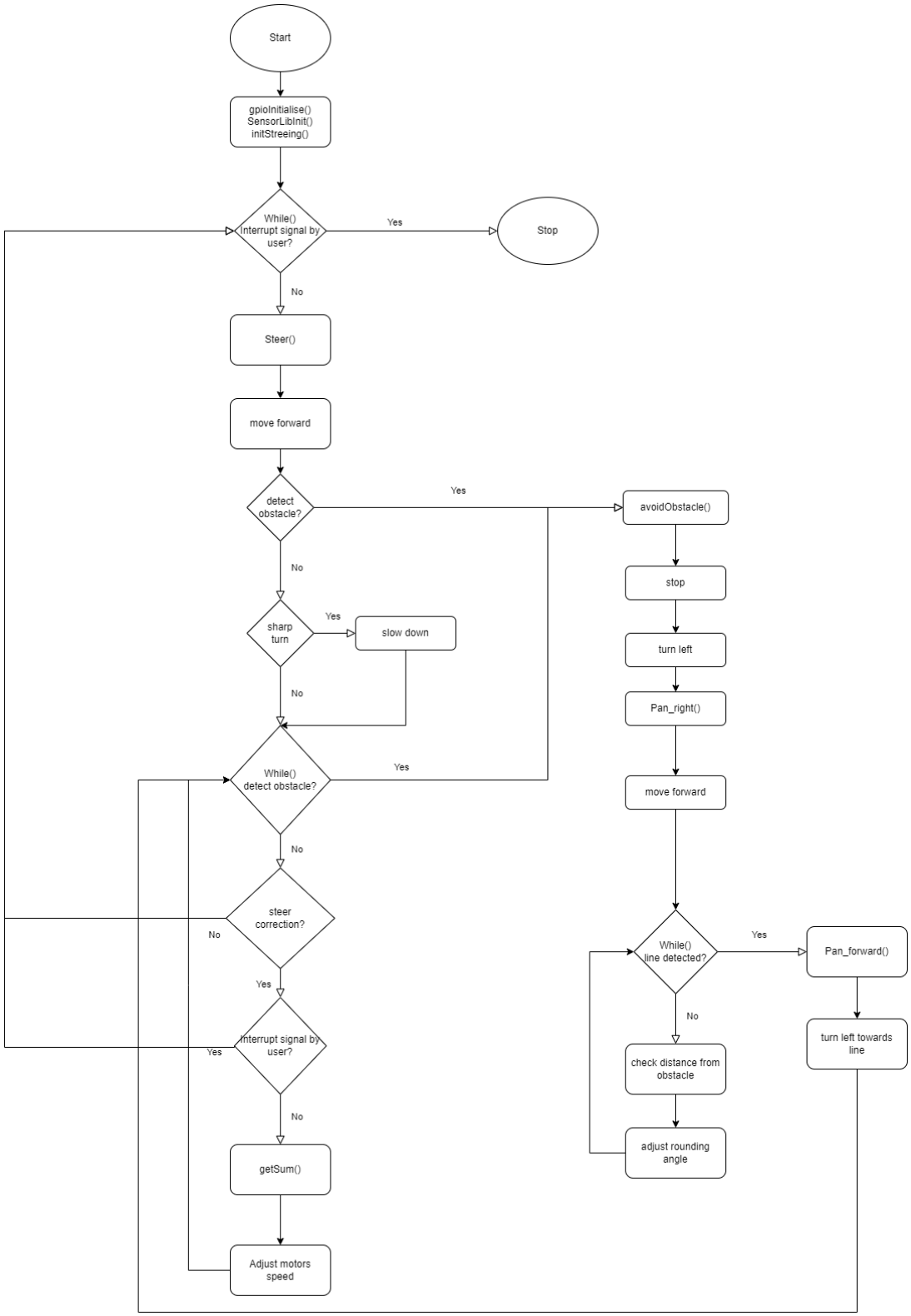
Libraries & software

Waveshare Motor Hat Library:https://www.waveshare.com/wiki/Motor_Driver_HAT

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

The PIGPIO library was used to reduce the amount of files used by Waveshare's motor hat library. PIGPIO includes functions that facilitated reading and writing from the hat's registers and allowed us to reduce the amount of files used by the library. When refactoring Waveshare's library, we referenced the manual and the Waveshare's code.

Flowchart


Start
gpioInitialise()
SensorLibInit()
initStreeing()
While()
Interrupt signal by user?
Yes
Stop
No
Steer()
move forward
detect obstacle?
Yes
avoidObstacle()
No
sharp turn
Yes
slow down
No
stop
turn left
Pan_right()
While()
detect obstacle?
Yes
move forward
No
steer correction?
No
Yes
While()
line detected?
Yes
Pan_forward()
No
turn left towards line
Interrupt signal by user?
Yes
check distance from obstacle
No
adjust rounding angle
getSum()
Adjust motors speed

Pin Assignments

I2C Communication

- GPIO 2 (I2C1 - SDA): Used for I2C communication with the Waveshare Motor Hat.

- GPIO 3 (I2C1 - SCL): Used for I2C communication with the Waveshare Motor Hat.

SPI Communication

- GPIO 7 (SPI0 - CE1): Used by the LS7366R encoder interface.

- GPIO 8 (SPI0 - CE0): Used by the LS7366R encoder interface.

- GPIO 9 (SPI0 - MISO): Used by the LS7366R encoder interface for Master In Slave Out communication.

- GPIO 10 (SPI0 - MOSI): Used by the LS7366R encoder interface for Master Out Slave In communication.

- GPIO 11 (SPI0 - SCLK): Used by the LS7366R encoder interface for Serial Clock.

Line Sensors

- GPIO 22: Connected to the first line sensor (from left).

- GPIO 23: Connected to the second line sensor.

- GPIO 24: Connected to the third (middle) line sensor.

- GPIO 25: Connected to the fourth line sensor.

- GPIO 26: Connected to the fifth (rightmost) line sensor.

Servo and Ultrasonic Sensor

- GPIO 12: Connected to the servo PWM output for controlling the angle of the ultrasonic sensor.

- GPIO 13: Connected to the ECHO pin of the ultrasonic sensor for receiving the echo signal.

- GPIO 14: Connected to the TRIG pin of the ultrasonic sensor for sending the trigger pulse.

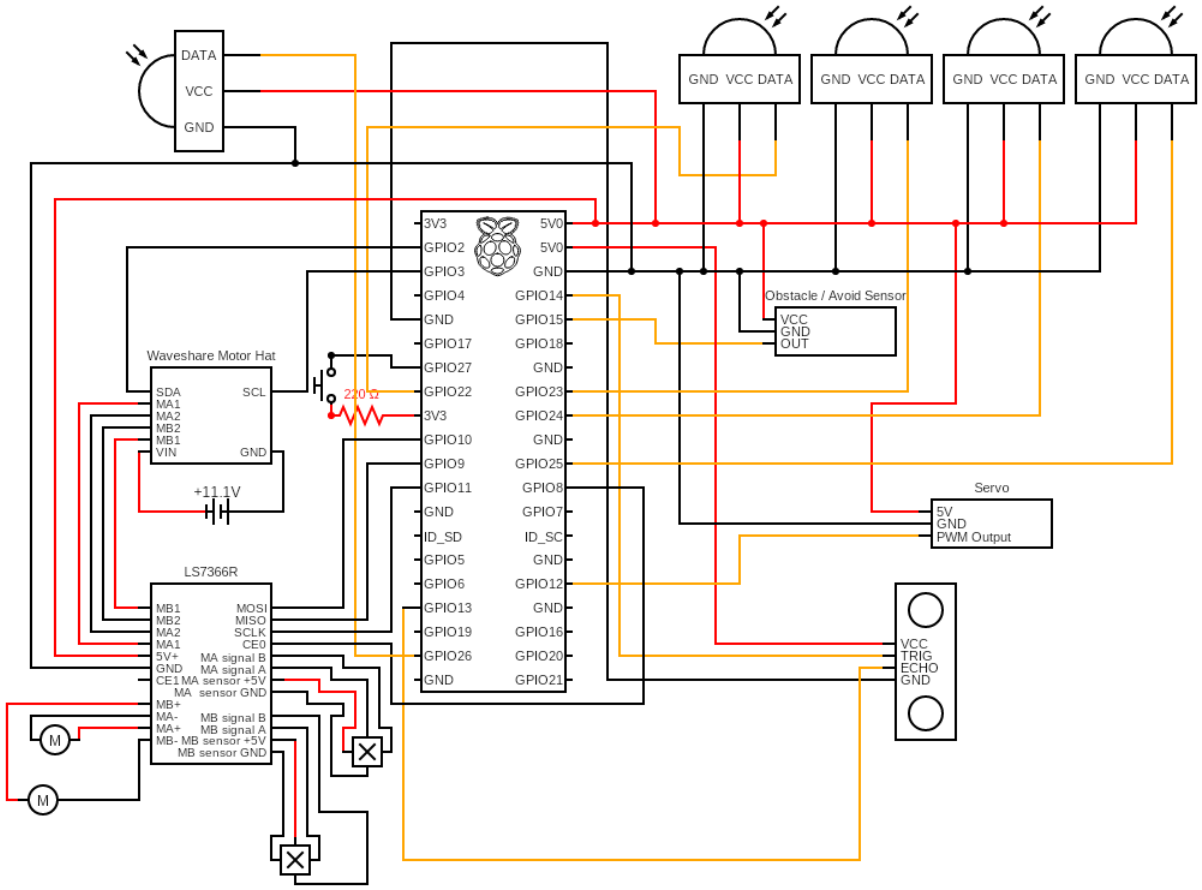
Obstacle Avoidance Sensor

- GPIO 15: Connected to the output of the IR Obstacle Avoidance Sensor.

Button:
- GPIO 27: Pressed in order to start the car

Free Pins
- GPIO 4, 5, 6: These GPIOs remain free and can be used for future expansions or additional functionalities.

Hardware Diagram



What worked well

Robust Sensor Integration

The integration of multiple sensors, including line sensors and the ultrasonic distance sensor, worked exceptionally well. The line sensors provided reliable and accurate detection of the track, allowing the autonomous car to follow the line with precision. The use of five line sensors helped in precise navigation, especially in differentiating between straight paths and curves, enhancing the car's ability to stay on the path consistently.

Steering Algorithm

The implementation of our algorithm for steering adjustment proved to be highly effective. The tuning of the our adjustment frequencies, and giving each line sensor a different weight resulted in smooth and responsive steering adjustments, which significantly improved the vehicle's ability to maintain its course along the line. This was particularly evident in areas where the track had sudden turns or deviations, where the car managed to adjust its path swiftly and accurately.

Stable Mechanical Setup

The mechanical setup of the car, including the chassis, wheels, and motor assembly, was robust and stable. This stability was crucial during operation, as it minimized vibrations and unnecessary movements, ensuring that the sensors could operate without interference and that the motion was smooth. The batteries and heavier components were placed towards the back of the car near the center of gravity, for proper weight distribution.

Reliable Communication Protocols

Utilizing I2C protocols for communication between the Raspberry Pi and peripheral devices such as sensors and motor drivers worked flawlessly. These communication protocols facilitated a stable and fast data transfer, which was critical for real-time sensor data processing and motor control. We also setup SPI, but wound up not needing to use it.

Software and Hardware Integration

The seamless integration between the hardware components and the software control systems was a major success. The software was able to effectively interpret sensor data, execute control algorithms, and send appropriate commands to the hardware (motors and sensors), which performed as expected without any significant issues.

Ultrasonic Sensor for Obstacle Avoidance

The ultrasonic sensor's ability to detect obstacles was notably effective. It provided the necessary data to implement dynamic obstacle avoidance strategies, allowing the car to navigate around obstacles without manual intervention and then return to the line-following mode promptly.

What were issues

Motor Control Challenges

One of the primary issues encountered in the project involved difficulties with motor control. Initially, the motors did not respond consistently to the commands issued by the Raspberry Pi through the Motor Driver HAT. Several factors contributed to this problem:

- Power Supply Insufficiency: The motors occasionally experienced power drops, particularly under load or during rapid direction changes, which led to uneven speed and torque performance.

- Software-Hardware Integration: Discrepancies between the motor control software and the hardware setup caused synchronization issues. The timing of signals sent to the motors was not always optimal for the required operations.

- Encoder Feedback: There were challenges in accurately reading back the motor encoder feedback, which is critical for precise speed and position control. Misreadings led to overcorrection or undercorrection in the PID control loop.

Sonar Sensor Inconsistencies

The sonar sensor, intended for dynamic obstacle detection and avoidance, also presented challenges:

- Erratic Readings: The sonar sensor occasionally gave erratic distance readings, which were likely caused by environmental factors such as angles of surfaces from which the ultrasound waves were reflected or interference from nearby objects not directly in the path of travel.

- Integration with Control Logic: Integrating these readings into the control logic was challenging because the sensor's output needed to be debounced and filtered to prevent the car from making unnecessary or abrupt changes in its path.

Additional Technical Issues

- Sensor Placement and Alignment: Proper alignment of the line sensors was initially a problem, affecting the car's ability to detect the correct path. Misalignment led to poor line tracking performance until the sensors were correctly positioned.

- Environmental Factors: Variations in lighting conditions affected the sensitivity and performance of the line sensors and the obstacle avoidance sensor. Calibration under different lighting conditions was necessary to ensure consistent performance.

- Software Bugs and Instabilities: Bugs in the initial versions of the software led to occasional crashes and unresponsive behavior, particularly when handling complex scenarios involving simultaneous line tracking and obstacle avoidance.

- Inconsistent Sonar Values due to Threading: We used CPU ticks (clock_t) to calculate the amount of time it took the sound wave to return to the sonar sensor. Because we had a thread per sensor, totalling to 7 threads, and our pi has four cores. Each of the pi's cores is single threaded. Having a thread for each sensor caused the sonar's values to become inconsistent because we were obtaining the time difference through ticks. The CPUs clock would keep going even when the sonar thread was waiting, causing inconsistent time readings.

Resolutions and Lessons Learned

Efforts to resolve these issues included enhancing the power management system to ensure a stable supply to the motors, and implementing more robust filtering and debouncing algorithms for the sonar sensor. The software was also iteratively debugged and optimized to improve stability and performance.

These challenges provided valuable learning experiences in dealing with real-world issues in robotics, emphasizing the importance of thorough testing and calibration, robust system design, and adaptive software development.

Regarding the sonar, because we had 7 threads in addition to the main process, and the Raspberry Pi 4's CPU is 4 single threaded cores, we moved all 5 of the line sensors into one thread. This was necessary to get accurate readings on the sonar, since it relied on the CPU's clock ticks to accurately track time. Having multiple threads caused the CPU to tick unevenly for the sonar's thread, as it would have to manage more than one thread.