Final Project - Car

San Francisco State University

CSC 615 | Professor Robert Beirman

Group | **Robert RedBull Racing**

Zach Howe (923229694)

Yu-Ming Chen (*923313947*)

Aditya Sharma (917586584)

James Nguyen (922182661)

Github | *ZHowe1*

Table Of Contents

Description

The goal is to build a car utilizing a stack of sensors, make it follow the line (black tape), achieve specific actions depending on the line color, and avoid and go around the obstacles.

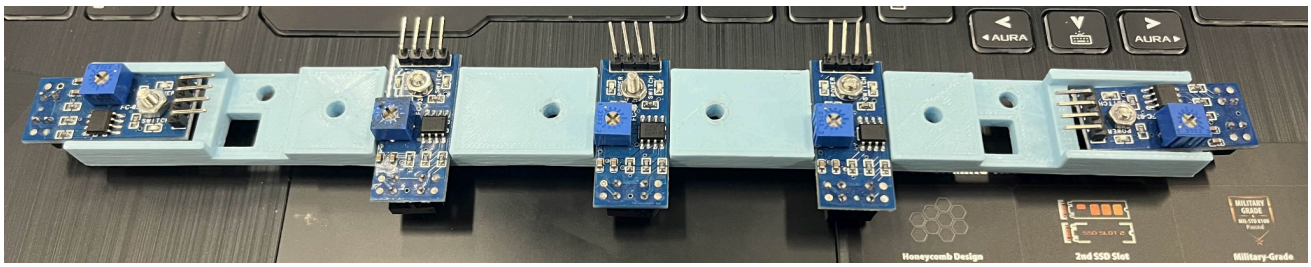
A self-defined library made it possible to work with a cleaner function pool and case-specific actions.

Parts / Sensors Used

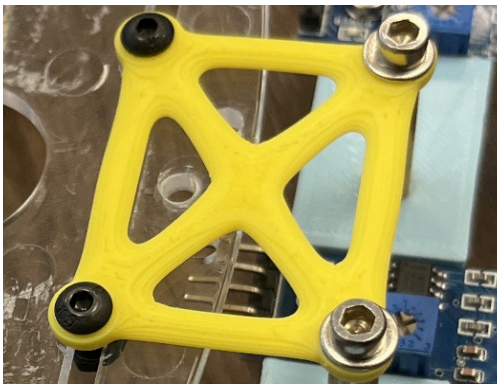
- 3x **HC-SR04** Sonic echo Sensors
- 5x **TCRT5000** Reflective Optical Sensors
- 1x Raspberry Pi Zero 2 W
- 4x Metal motors

The Building of the Bot

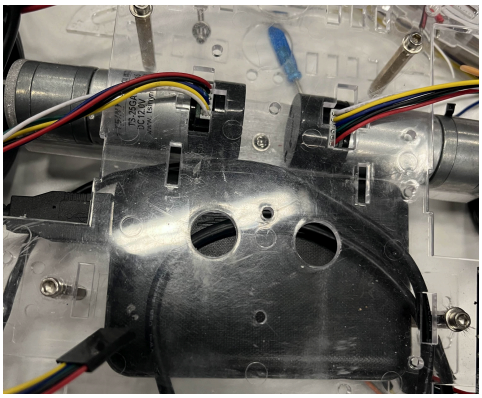
1. The first step/iteration for building the car was to build the basic car that came in the kit, using the provided chassis and motors to build a test frame we could test code on. Through the iterative design process, we were able to make decisions on what parts of the car needed to be removed, appended, or modified.
2. The second iteration was moving from the default hard mount holes on the car chassis for the **TCRT5000 Reflective Optical Sensors** *(will be referred to as TCRT5000)* to a custom part. We designed a sensor array bracket in Fusion 360 CAD based on the **TCRT5000** schematics/dimensions online, and 3D printed it in PLA. The choice for this decision was to be able to spread out our sensors more. We did this to be able to read how far off we are. For example, if the far left sensor was triggered, we need to correct a lot more to the left than if the closest left sensor was triggered.



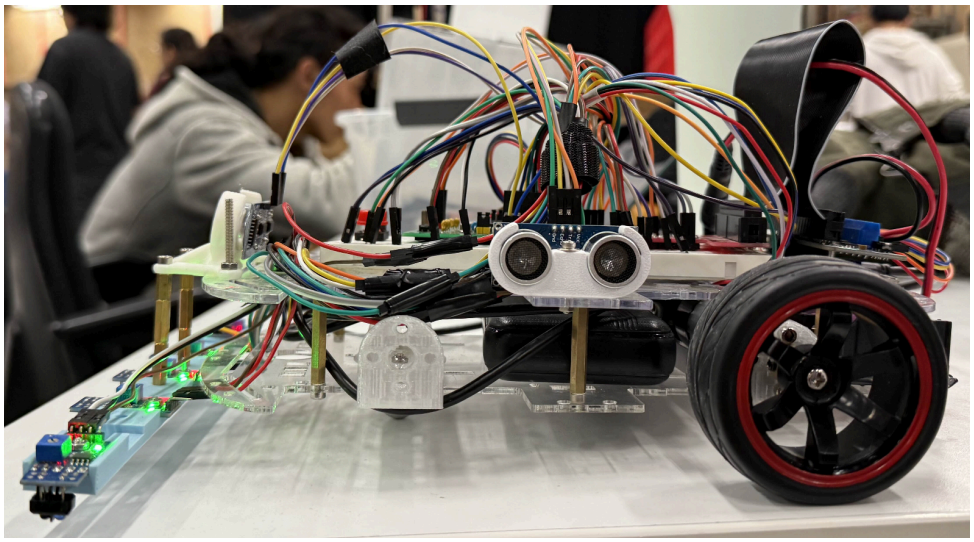
3. The third small iteration that needed to be made was to shift the **TCRT5000** sensor array bracket that we designed forward, as it interfered with the front wheels. The mounted empty sensor array was able to clear the front wheels. Though when the **TCRT5000** sensors were mounted on the array, the pins on the boards did not have sufficient clearance to attach the wires due to an oversight that is also discussed in our issues section. So we designed a small brace that would move the mounting holes of the bracket forward, pushing our sensors forward too. This was done to make enough clearance to attach wires, and also by pushing the sensors more towards the front, we move the sensing further from our pivot point. By moving the sensors forward, we can detect deviations from the line earlier and execute smoother turns.



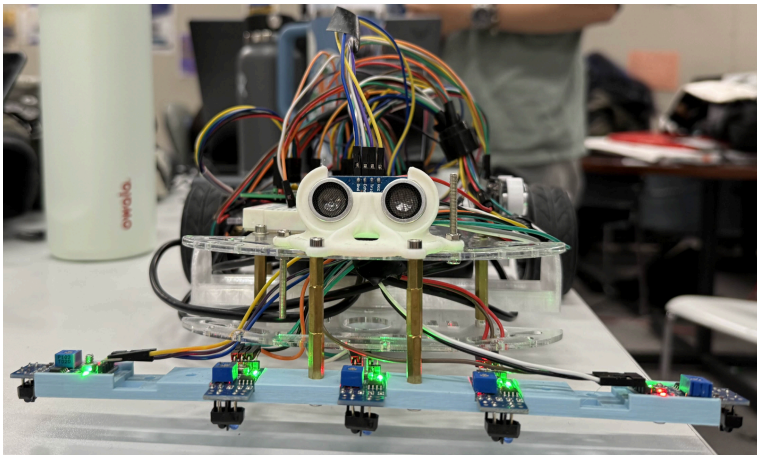
4. The fourth iteration was that we changed from the stock yellow motors to the metal motors. The main reason for this change was not for speed or torque but to move the center of gravity down and to widen our wheelbase. We printed the provided motor mounts STLs in PETG, as PETG is stronger and more durable than PLA. Due to how the mounts are designed, their footprints on the chassis are smaller, so we were able to move our battery pack from the top to inside the car.



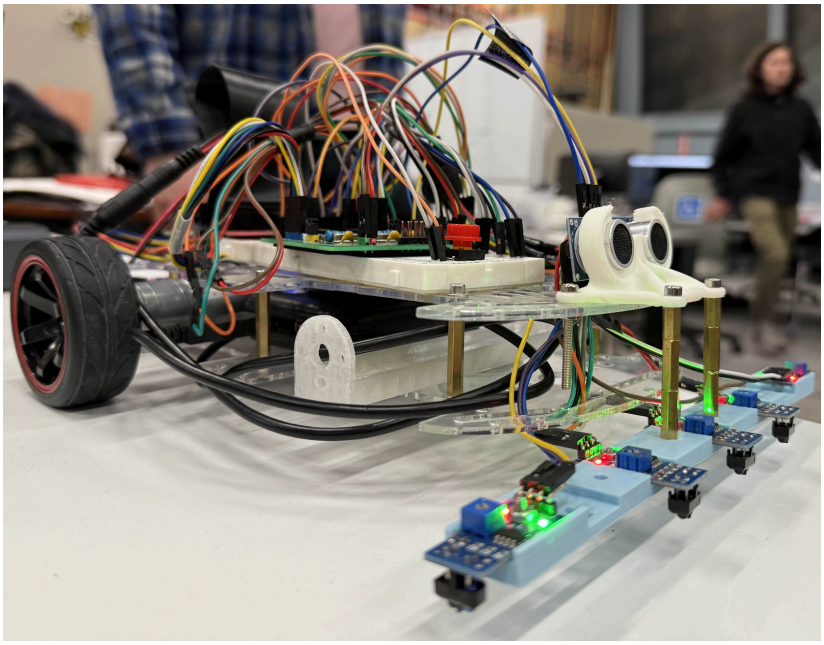
5. The fifth iteration was just the addition of the **HC-SR04 Sonic Echo Sensors** *(will be referred to as HC-SR04)*. We researched to find an STL online that mounts **HC-SR04s** onto the car. We printed 3 custom brackets in PETG, but only used two for the left and right sensors, as the front sensor mount needed to be changed to add a 10-degree tilt up.



6. The sixth iteration was when we started to mount the front **HC-SR04** for obstacle detection. Rather than using the same 3D printed mounts on the left and right **HC-SR04's** we redesigned the mount along with the front brace that held the **TCRT5000** sensor array. We made this decision because the front brace needed to be reinforced and because the mounting holes for the **HC-SR04** mount were covered by our custom brace. In Fusion 360 CAD, we sketched out the hard point holes needed for mounting the brace and also imported the original **HC-SR04** mount. Using fusion, we were able to connect the bodies using generative design, which optimized strength and weight. This new design was then printed in PLA and mounted to the car, allowing for the front **HC-SR04** to look forward up by 10 degrees while also holding the **TCRT5000** sensor array.



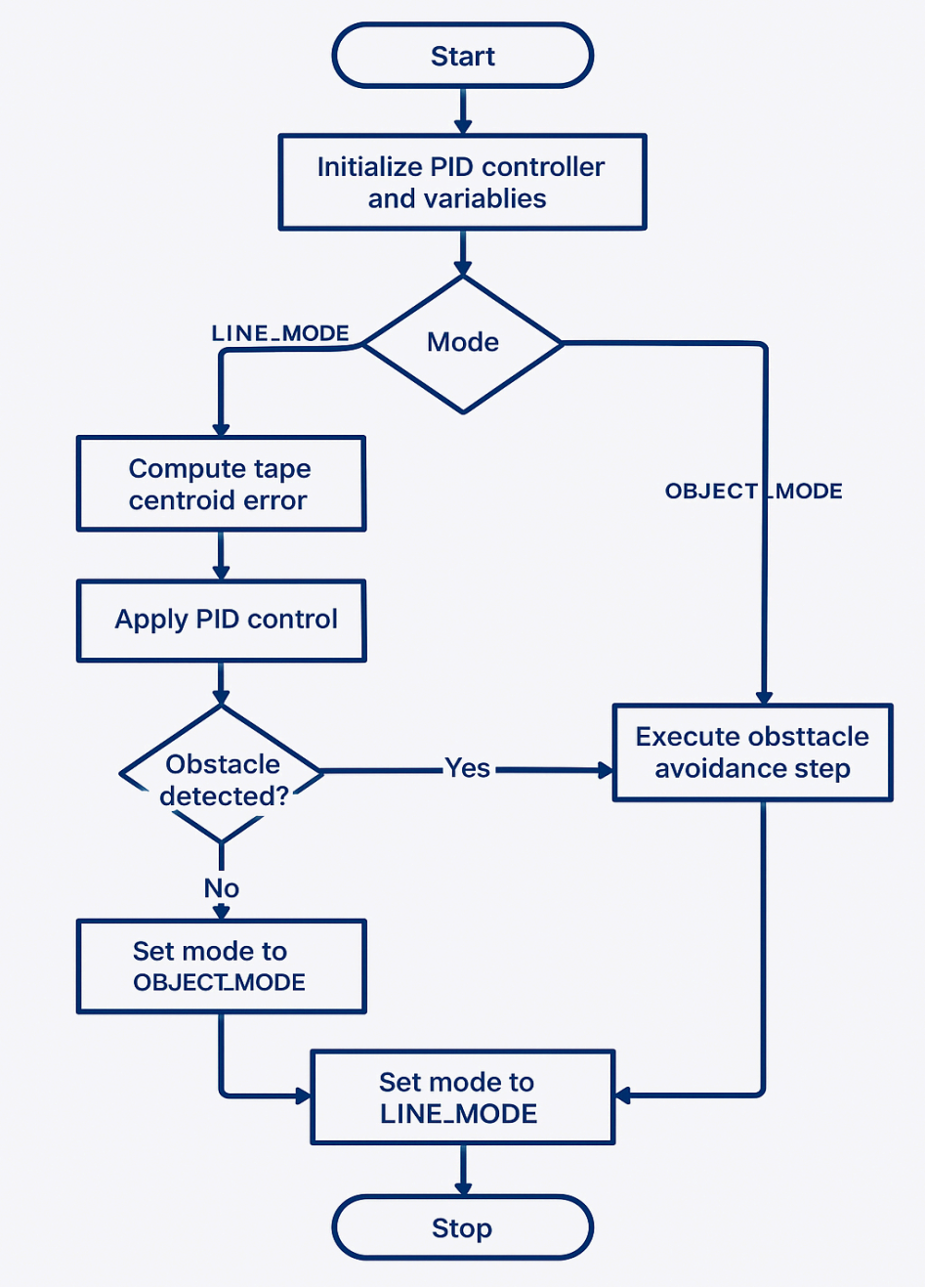
7. The seventh iteration was when we decided to remove the front two wheels and change the design to have two driven wheels in the back with a ball caster in the front. Our reason for this decision was that we already had a working solution for line following with two-wheel drive. So we decided to scrap the plan with four wheels. By scraping the two front wheels, we save weight and time as we can spend less time troubleshooting 4 four-wheel drive and can continue working on the rest of the car.



Libraries/Software Used

- Waveshare Motor Driver HAT driver code → I2C

Flowchart of the Program Execution



Pin Assignments used

Component	Purpose	BCM Pin	Physical Pin	Direction
Tape Sensor 1	Left-most tape sensor	26	Pin 37	Input
Tape Sensor 2		19	Pin 35	Input
Tape Sensor 3	Center tape sensor	13	Pin 33	Input
Tape Sensor 4		6	Pin 31	Input
Tape Sensor 5	Right-most tape sensor	5	Pin 29	Input
Ultrasonic Front TRIG	Trigger pin for front sensor	23	Pin 16	Output
Ultrasonic Front ECHO	Echo pin for front sensor	24	Pin 18	Input
Ultrasonic Side TRIG	Trigger pin for side sensor	20	Pin 38	Output
Ultrasonic Side ECHO	Echo pin for side sensor	21	Pin 40	Input
Start Button	Start program on press	15	Pin 10	Input
Button LED	Output pin to signal button press	17	Pin 11	Output
Motor Driver HAT	I2C control	SDA/SCL	Pins 3/5	Output/Input
	PWM outputs to motors via PCA9685	(I2C Addr)	-	-

Approach / Issue & Resolutions

Here is our section of approaches in order to complete this assignment and build a library for future projects:

Building the Self-Defined Library

We took inspiration from the previous assignments and utilized the WaveShare code to build our own GPIO library. For this project, we decided to go wild and use a memory-mapped I/O addressing method to control our Raspberry Pi.

```c
typedef struct        ZHowe1, 4 weeks ago * Added older library that works with motors,
{
    int protection;
    int flags;
    char path[100];
    int location;
    int perms;
} MMap_Config;


// Start the mmap`
volatile unsigned int* init_io(MMap_Config config, void** gpio_map);

// Set pins to input or output
int set_output(volatile unsigned int* gpio, int pin);
int set_input(volatile unsigned int* gpio, int pin);


/*
*  Set the pin(s) from on to off
*  or vice versa. (write)
*/

int write_pins(volatile unsigned int* gpio, int* pins, int num_pins, int state);
int write_pin(volatile unsigned int* gpio, int pin, int state);

/*
*  Get the pin(s) data. (read)
*/

int* read_pins(volatile unsigned int* gpio, int* pins_val ,int* pins, int num_pins);
int read_pin(volatile unsigned int* gpio, int pin);

/*
*  clean and unmmap
*/

int clean(void* gpio_mmap);

/*
* Set up default signals for cleanup
*/

int set_signals();
```

(Our pins.h that handles all the GPIO actions needed)

```c
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))

#define GPIO_SET  *(gpio + 10 - (state * 3))
#define GET_GPIO(g) (*(gpio+13)&(1<<g))
```

(Our defines in pins.c that allow us to initialize pin in/out by doing bitwise action on memory maps.)

This allows us to create the same functionality that the PiGPIO library would do.

Thread Pool

During the early stages of developing our code design, we considered using a thread pool to save scarce CPU resources. The idea of a thread pool is to have a working queue on the line, and the worker threads will try to get jobs that are listed. Once the worker thread finishes its current job, it will go and grab more work from the queue.

Since it would be difficult to build a complete thread pool library from scratch, I

However, after we finished with the thread pool library, we decided not to use it. There is too much uncertainty for the car to run with only two threads when multiple sensors need data processing. The accounting for job/task priority is also required to be taken into consideration. Although it is a pity that we set it aside, it is truly a possibility that we can explore in the future.

State Machine

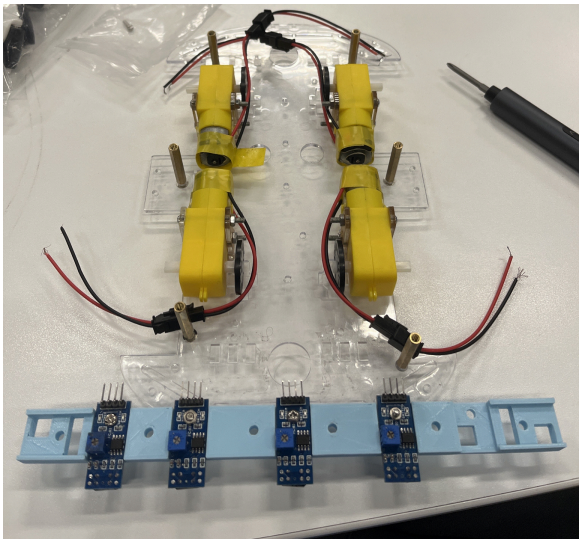
We built a state machine because we wanted to flip between parts.

- Following the line
- Detecting the line
- Going around the object
- Object Mode but losing the object
- Object Mode, but on the line first

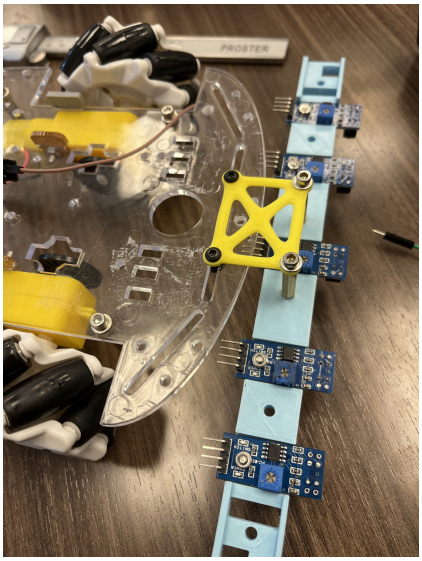
We decided to do this because it felt like it made it easier to organize the code. The core premise is to allow us to have different actions for different moments in the code. We decided to use a switch statement for it because we can quickly switch between each one within each one.

Issues when Building the Car & Physical Tuning

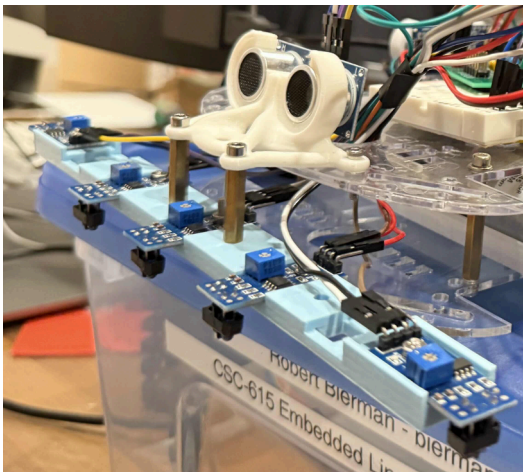
At first, we had the sensor bar directly mounted to the car's chassis like in the image below, but we soon realized that there wouldn't be enough clearance for the pins coming off the IR sensors, as the pins and the wires would hit the front wheels.



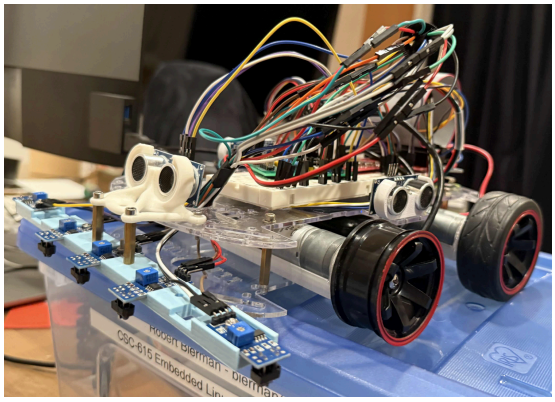
We then designed and 3D printed a new bracket so we could move the sensor array forward to make clearance for the wheels and sensor pins, as seen below.



Then for the third iteration of this bracket, we redesigned and 3D printed the bracket to be dual purpose, to also accommodate the sonar sensor as seen below. We modeled the third iteration in Fusion 360 CAD. We modeled the hardpoint requirements and then utilized generative design to connect the hard points together to achieve the desired organic design.



Lastly, we changed from our 4 wheeled design to a 2 wheel design with a ball caster in the front. We made this decision as we had tested and confirmed the car following the line with only 2 motors driving. When trying to implement 4-wheel drive, we ran into issues of tire skidding and grip issues, so we decided to scrap the 4-wheel drive design as we had 2-wheel drive working. Immediately below is the 4-wheel design, and below that is the current 2-wheel design.



An issue we ran into for the RGB sensor is that the Raspberry PI didn't detect the I2C when using the ribbon extender for the GPIO pins. For some unknown reason, the extension ribbon would not relay the SCL and SDA pins properly, so the PI couldn't detect the RGB sensor. Due to that problem, we decided to omit the RGB sensor from the final build of the car.

Obstacle Avoidance

For this, we are using PID (Proportional–integral–derivative controller) for our method to avoid obstacles. PID, in simple terms, is a feedback-based control mechanism that allows us to have continuous control during the avoidance of the obstacle.

There are all logical errors where our PID function did not make enough of a correction at first and after changing the KP KI values as well as changing the weighted edges of each sensor to be (-8, -2, 0, 2, 8) we were able to successfully stay on the line with our PID calculations providing enough of a corrections that it changes each of the motors power enough to turn.

In addition to this, if the sensors all read a line or no line, it will set the last error and the current error and add the integral to it in order to spin and find the line again. The following screenshots show the initially dull correction, and after the much finer adjustments we were able to achieve, including the power differences of the motors, which eventually led us to accurate line following.

```
DEBUG: L= 52% R= 48%
```

```
SENS:00100  ERR=+0.00  S=+0.00  L=80 R=80  DIST=1192.9
SENS:00100  ERR=+0.00  S=+0.00  L=80 R=80  DIST=1192.9
SENS:00100  ERR=+0.00  S=+0.00  L=80 R=80  DIST=1192.9
SENS:00100  ERR=+0.00  S=+0.00  L=80 R=80  DIST=1192.9
SENS:00110  ERR=+1.00  S=+35.09  L=45 R=100  DIST=1192.9
SENS:00110  ERR=+1.00  S=+7.00  L=73 R=87  DIST=1192.9
SENS:00110  ERR=+1.00  S=+7.00  L=73 R=87  DIST=1192.9
SENS:00110  ERR=+1.00  S=+7.00  L=73 R=87  DIST=1192.8
SENS:00110  ERR=+1.00  S=+7.00  L=73 R=87  DIST=1192.8
SENS:00110  ERR=+1.00  S=+7.00  L=73 R=87  DIST=1192.8
SENS:00110  ERR=+1.00  S=+7.00  L=73 R=87  DIST=1192.8
SENS:00010  ERR=+2.00  S=+42.05  L=38 R=100  DIST=1192.8
SENS:00010  ERR=+2.00  S=+14.00  L=66 R=94  DIST=1192.8
SENS:00010  ERR=+2.00  S=+14.00  L=66 R=94  DIST=1192.8
SENS:00010  ERR=+2.00  S=+14.01  L=66 R=94  DIST=1192.9
SENS:00010  ERR=+2.00  S=+14.01  L=66 R=94  DIST=1192.9
SENS:00010  ERR=+2.00  S=+14.01  L=66 R=94  DIST=1192.9
SENS:00010  ERR=+2.00  S=+14.01  L=66 R=94  DIST=1192.9
SENS:00010  ERR=+2.00  S=+14.01  L=66 R=94  DIST=258.7
SENS:00000  ERR=+2.41  S=+28.38  L=52 R=100  DIST=258.7
SENS:00000  ERR=+2.86  S=+32.78  L=47 R=100  DIST=258.7
SENS:00000  ERR=+3.37  S=+37.67  L=42 R=100  DIST=258.1
SENS:00001  ERR=+8.00  S=+184.40  L=0 R=100  DIST=258.1
SENS:00001  ERR=+8.00  S=+56.02  L=24 R=100  DIST=258.1
SENS:00001  ERR=+8.00  S=+56.02  L=24 R=100  DIST=70.5
SENS:00000  ERR=+8.99  S=+90.69  L=0 R=100  DIST=70.5
SENS:00000  ERR=+10.15  S=+99.91  L=0 R=100  DIST=70.5
SENS:00000  ERR=+11.51  S=+118.04  L=0 R=100  DIST=70.5
SENS:00000  ERR=+13.08  S=+134.68  L=0 R=100  DIST=69.7
SENS:00010  ERR=+2.00  S=-297.38  L=100 R=0  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.04  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.05  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.05  L=66 R=94  DIST=69.7
SENS:00010  ERR=+2.00  S=+14.05  L=66 R=94  DIST=69.7
```

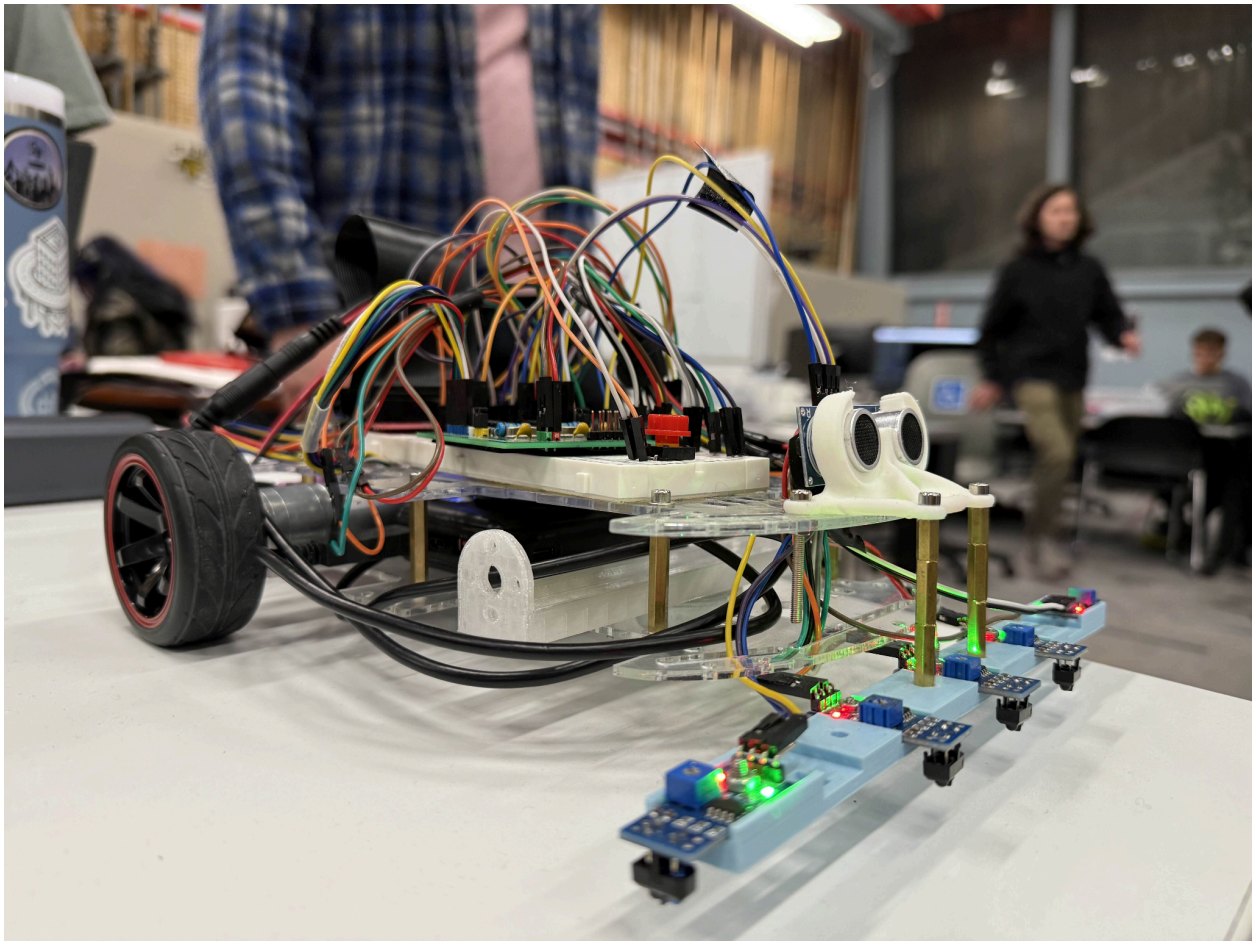
(The unstable output, before changing KP&KI values)

```
DEBUG: L= 50% R= 50%
```

```
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.25 steer=0.71 L=50.71 R=49.29
DEBUG: error=0.33 steer=2.68 L=52.68 R=47.32
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.81 L=50.81 R=49.19
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
DEBUG: error=0.33 steer=0.82 L=50.82 R=49.18
```

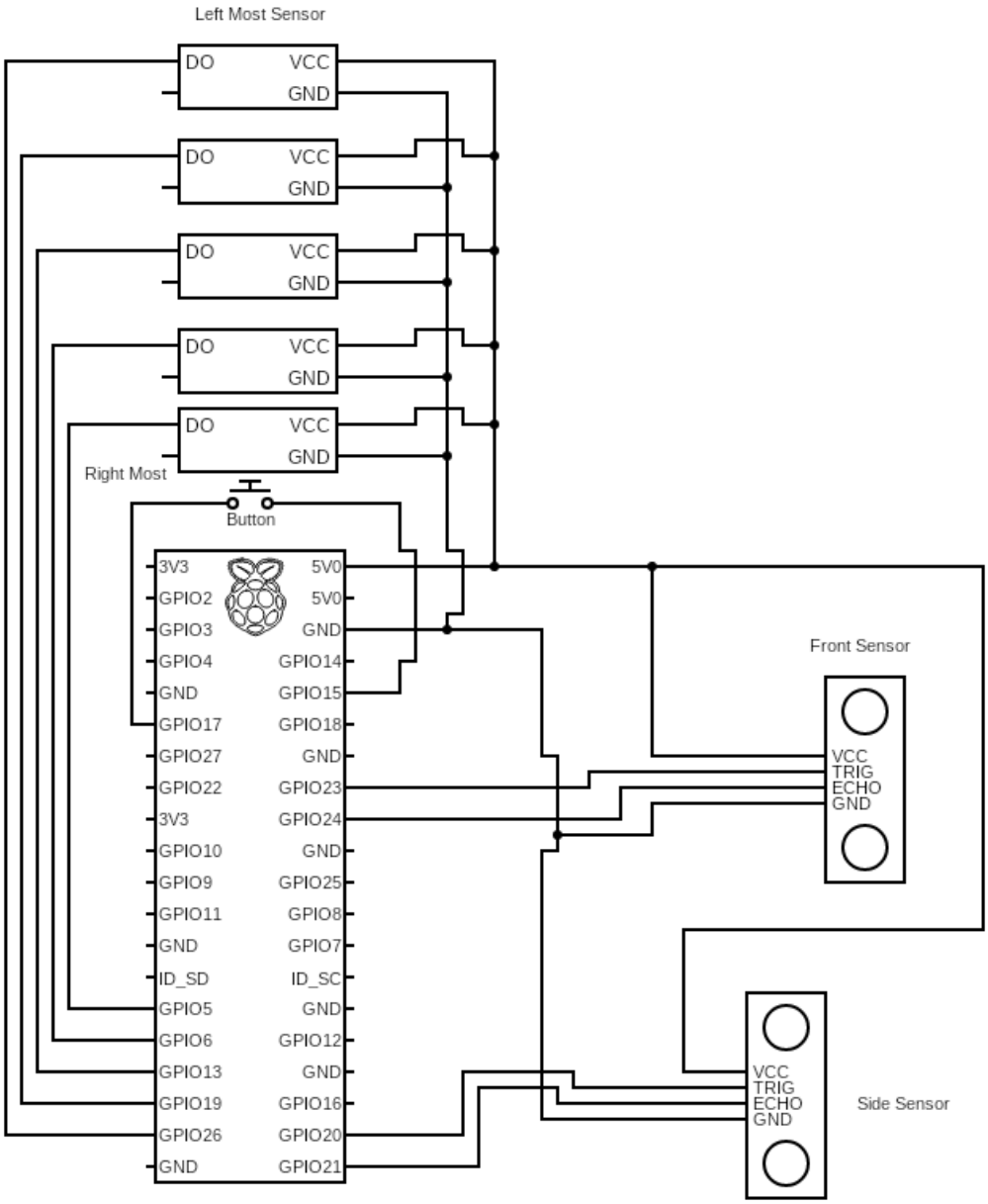
(The more stable/acceptable results after changing KP&KI values)

Photo of the completed car



Hardware Diagram

Tool used: Circuit Diagram www.circuit-diagram.org/



Work Assignments

Name	Work Done
Zach Howe	Building the library, object detection, and debugging
Aditya Sharma	Car movement implementation, object detection, and debugging
Yu-Ming Chen	Coding style review & formatting, writeup, and debugging
James Nguyen	Program loop implementation, building the car, and write-up

Team Photo