
Instructable: How to Build RC Rar

Team Hidden Valley Ranch
Tim Gayed, Matthew Mazalewski, Kim Tran, & Wyatt Van Orden

Introduction

The objective of the project was to build an RC Car that can be controlled manually, but also be able to parallel park and avoid walls. To start, an Arduino was used to control the sensors. In order to achieve wireless connectivity, an ESP32 with built-in wifi was used. This would allow the car to be controlled via a phone app. The ESP32 controlled the motors based on the user input from the smartphone app.

Materials

Components

- 4x Brushless motors
- 1x Servo Motor
- 1x 11.1 V 1400 mAh Lipo Battery
- 1x ESP32 DEVKIT 1
- 1x Arduino Nano
- 2x Electronic Speed Controls (2 Additional Optional for more power)
- 3x VCNL 4000 Proximity Sensors
- 1x VL53L0X Laser Sensor

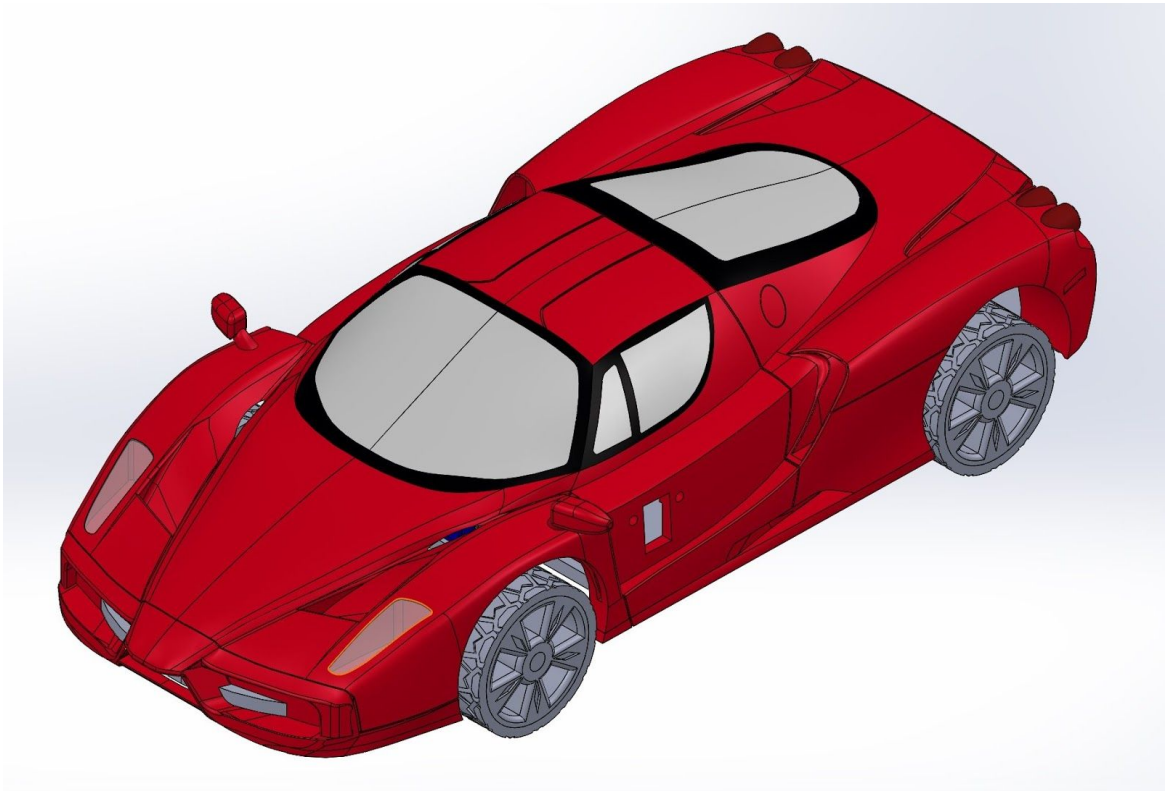
Assembly Parts

- 3D Printed Steering Assembly
- 3D Printed Rims
- 3D Printed Car Shell
- 20x Rubber O-Rings for Tires
- 2x M4 Bolts
- Paint (Optional)
- Tape (Optional
- Plaster (Optional)

Design

Step 1: Designing the Shell

The shell of the car is one of the most important aspects of the design as it provides the physical restraints for the rest of the component selection. For this project, we modeled our shell after a Ferrari Enzo. To do this we found a model on grabCAD. After downloading this model, we proceeded to hollow out the insides in Solidworks. This gave us space to house the sensors, motors, controllers, arduino, and battery.



After hollowing out the inside of the model, a base plate was designed. This will be where we will mount all of our components. Because of the size contrating of fitting all of the components inside of the car shell, familiar components such as sonic sensors cannot be used due to their size.

Step 2: Printing and painting the Shell

After hollowing out the Ferrari Enzo model, it was printed. The model was too large to be printed in a single piece, so it was split, printed separately, and then reattached. After assembling the

shell of the model, it was plastered. This made the print smoother and easier to work with. Next, the car was painted.

The picture below shows the painted shell of the car.



Step 3: Testing the motors

For this project, 4 Brushless motors were used to turn the wheels. 3D printed rims were fitted around the motors. 5 Rubber O-Rings were mounted to each rim to act as the tires for the wheel.

Because of the way that brushless motors work, their direction of rotation cannot be changed on demand. This is because they use 3 phase AC input to spin, compared to a traditional DC motor. Some ESCs have the ability to reverse directions, but because the ESCs that came with this motor were designed for drone flight, they were only intended to spin the motors in one direction very quickly.



Brushless Motor

For the car to be able to reverse, two of the leads on the brushless motor would have to be physically reversed. Initially, the two rear motors would be used to drive forward, while the front motors would be reversed to drive backwards. This opens a lot of possibilities for steering management. Putting the forward motors in the front, and the reverse motors in the back would allow the car to always be in ‘front-wheel drive’ and give a slight boost to steering. For this car, we decided to follow put the forward wheels in the back to more closely emulate a real life Ferarri Enzo. The motors can also be scaled down to where the car is driven off of a single motor. For example, the entire car can be driven off of a single brushless motor.

The following code was used to control motor speed with a potentiometer

```
int extra = 0;

void setup() {
  pinMode(6,OUTPUT);

  for(int x=0;x<100;x++){
  extra = map(extra,0,1023,0,1000);
  digitalWrite(6,LOW);
  delayMicroseconds(1000+extra);
  digitalWrite(6,HIGH);
  delayMicroseconds(1000+extra);
  x+=1;
  }

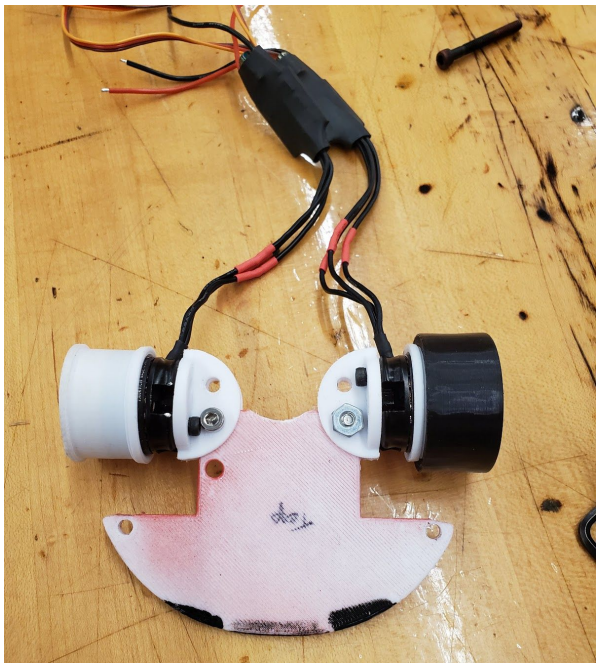
}
```

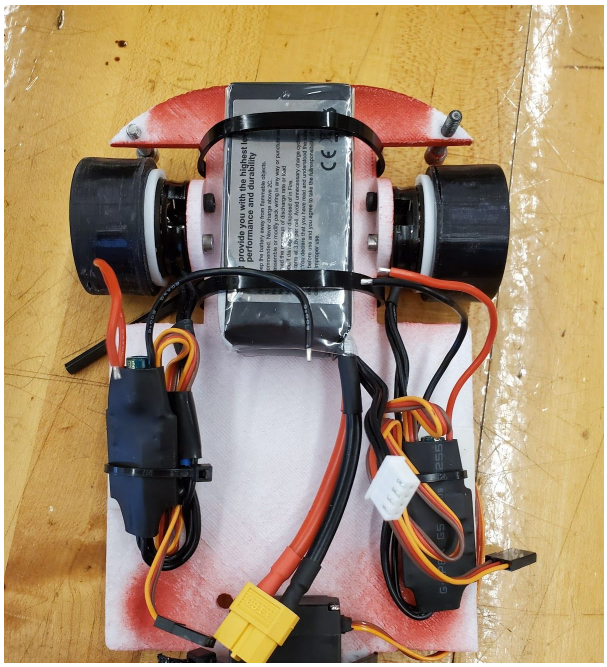
```
void loop() {
  extra = analogRead(0);
  extra = map(extra,0,1023,0,1000);
  digitalWrite(6,HIGH);
  delayMicroseconds(1000+extra);
  digitalWrite(6,LOW);
  delayMicroseconds(4000-(1000+extra));

}
```

Step 4: Attaching the Motors and Battery

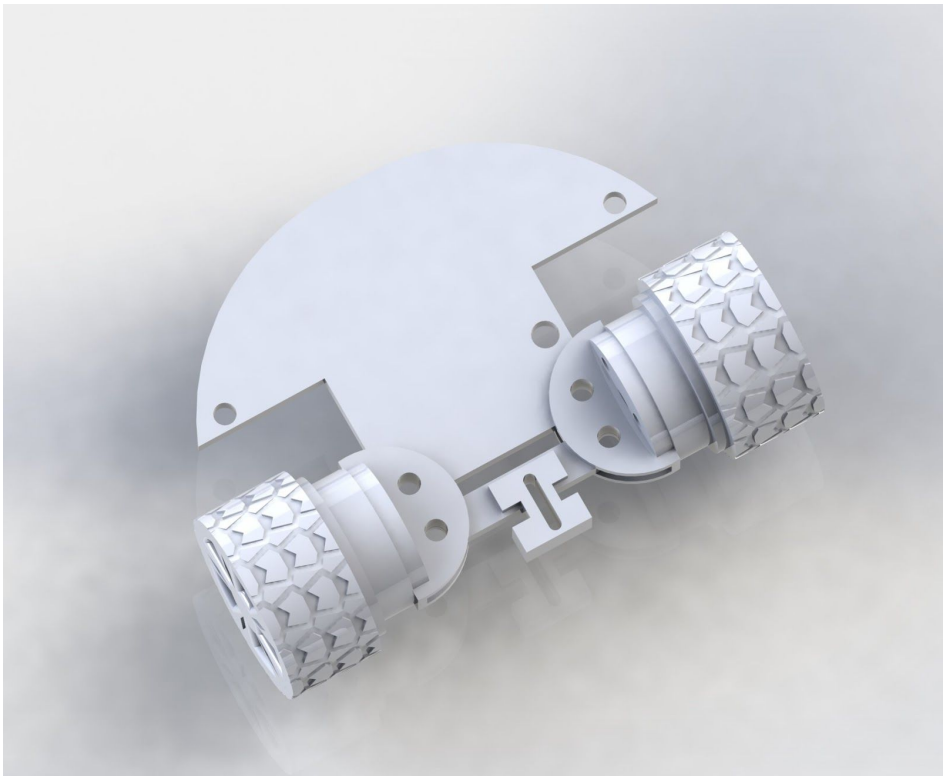
In the figure to the left, the two brushless motors are attached to the wheel holders by two M4 nuts and bolts. The wheel holders are then held to the front underbody of the car by two additional M4 nuts and bolts. Since the rear motors will not be assisting in the turning of the car, a steering mechanism is not needed. Therefore, only the wheel holders need to be attached to rear underbody of the car by four M4 nuts and bolts each. After mounting the motors, each electronic speed controller (ESC) must be connected to the proper pins on the ESP32 board. The ESC have three wires; brown, red, and orange, which correspond to ground, power, and signal, respectively.



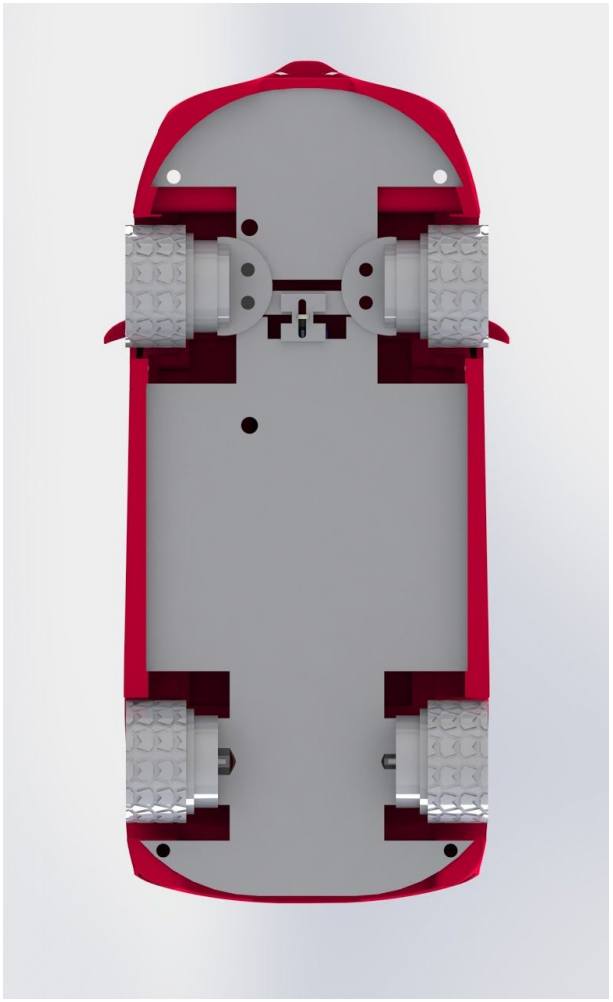


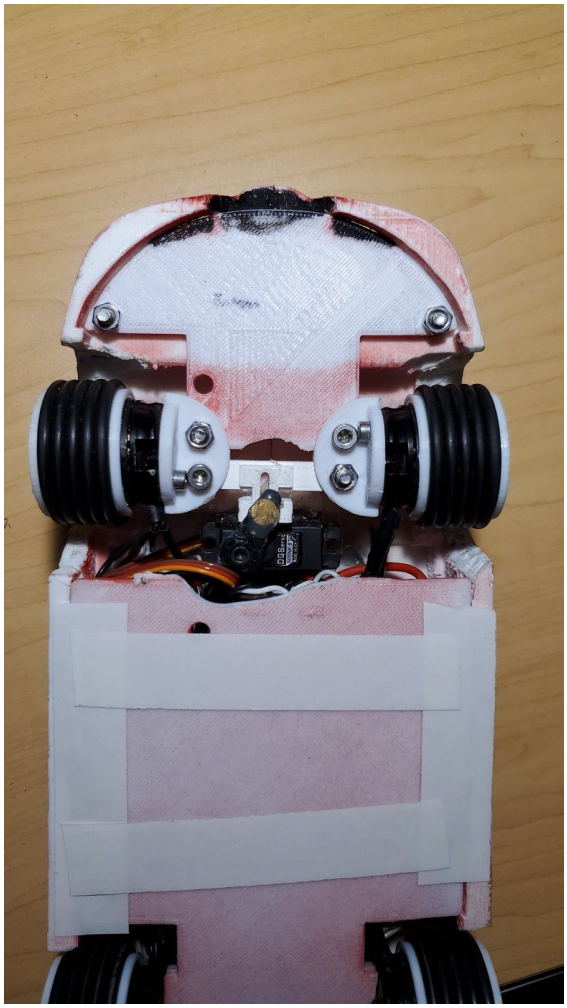
Step 5: Steering Design

Because of this projects traditional car design, a steering mechanism would need to be implemented. Below is the first iteration of the steering mechanism.



The two front wheels are connected with a 4-bar linkage. A servo is put into the cross-piece to move the wheels from side to side. The components for the steering mechanism where 3D printed.





The turning radius of the car was limited by the wheels hitting the chassis. To provide more space for the wheels to turn, the car was dremeled by the wheels.

Step 6: Setting up Bluetooth

Setting up wireless communication was the most challenging part of the project. Several devices were tried in an attempt to establish wireless communication to the car. A pair of HC-05 devices were tried in a master-slave configuration, however there seemed to be an issue with connecting the 2 devices even after binding the master device to the slave MAC address.

Next a pair of nrf24l01 transceivers were tried and again, no wireless communication was achieved. The goal was to try and get two devices connected together and build a seperate controller to drive the car. However, no communication was achieved using separate modules.

Finally an STM32, a development board with integrated wifi and bluetooth was used. Connection was established with this device by pairingthe STM32 with a smartphone. Using a joystick app and the serial monitor, the commands issued from the joystick app were recorded and assigned to commands for the car. The STM32 was responsible for controlling the motors based on inputs from the user smartphone app. Below is the code to control the car using the smartphone app.

```
#include "BluetoothSerial.h"
#include <ESP32Servo.h>

BluetoothSerial SerialBT;
Servo stearingServo;

int freq = 250;
int channel1 = 4;
int channel2 = 5;
int channel3 = 2;
int channel4 = 3;
int resolution = 10;
int durtyCycle = 256;

char controllerInChar = 'S';
int speedState = 0;

int servoPin = 18;
int servoCenter = 40;
int servoRight = 50;
int servoLeft = 26;
int servoCrnt = 40;

void setup() {

  Serial.begin(115200);
  if(!SerialBT.begin("Ferrari")){
    Serial.println("An error occurred initializing Bluetooth");
  }
  ledcSetup(channel1, freq, resolution);
  ledcSetup(channel2, freq, resolution);
  //ledcSetup(channel3, freq, resolution);
  //ledcSetup(channel4, freq, resolution );
```

```
  ledcAttachPin(12, channel1);
  ledcAttachPin(27, channel2);
  //ledcAttachPin(26, channel3);
  //ledcAttachPin(25, channel4);
  ledcWrite(channel1,125);
  ledcWrite(channel2,125);
  //ledcWrite(channel3,125);
  //ledcWrite(channel4,125);

  stearingServo.attach(servoPin, 500, 2400);

}

void loop() {
  while(SerialBT.available()){
    controllerInChar = SerialBT.read();
  }
  if(controllerInChar == 'S'){
    servoCrnt = servoCenter;
  }
  else if(controllerInChar == 'R'){
    servoCrnt = servoRight;
  }
  else if(controllerInChar == 'L'){
    servoCrnt = servoLeft;
  }
  else if(controllerInChar == 'F'){
    speedState += 1;
    controllerInChar = 'S';
  }
  else if (controllerInChar == 'B'){
    speedState -=1;
    controllerInChar = 'S';
  }

  switch (speedState){
    case -1://reverse
    ledcWrite(channel1, 256);//forward
    ledcWrite(channel2, 290);//Back
    break;
    case 0://stop
    ledcWrite(channel1, 256);//forward
    ledcWrite(channel2, 256);//Back
    break;
    case 1://slow
    ledcWrite(channel1, 280);//forward
    ledcWrite(channel2, 256);//Back
```

```
    break;
    case 2:// medium
    ledcWrite(channel1, 320);//forward
    ledcWrite(channel2, 256);//Back
    break;
    case 3:
    ledcWrite(channel1, 350);//forward
    ledcWrite(channel2, 256);//Back
    break;
  }

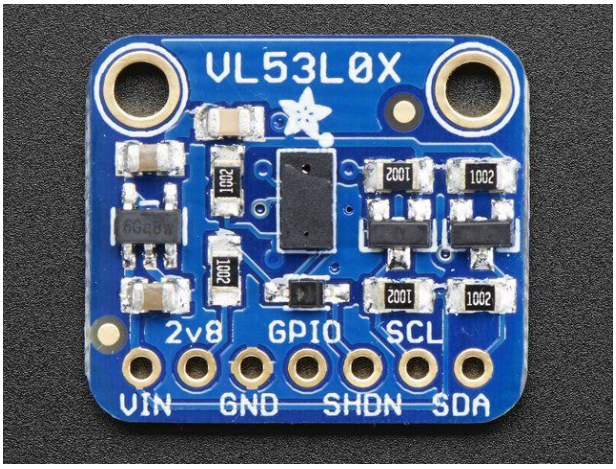
  stearingServo.write(servoCrnt);

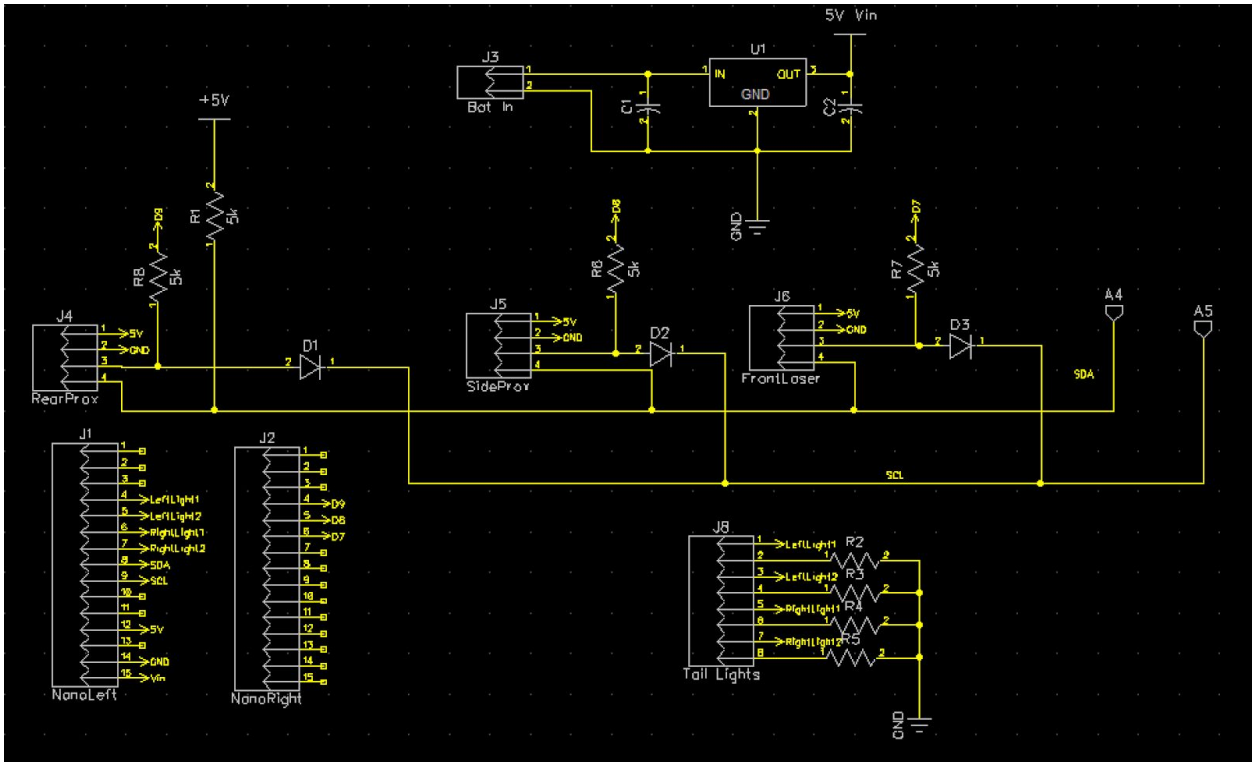
  Serial.print("In: ");
  Serial.print(controllerInChar);
  Serial.print("    Servo Pos: ");
  Serial.print(servoCrnt);
  Serial.print("    Speed: ");
  Serial.println(speedState);
}
```

Step 7: Connecting Sensors and Wiring Car

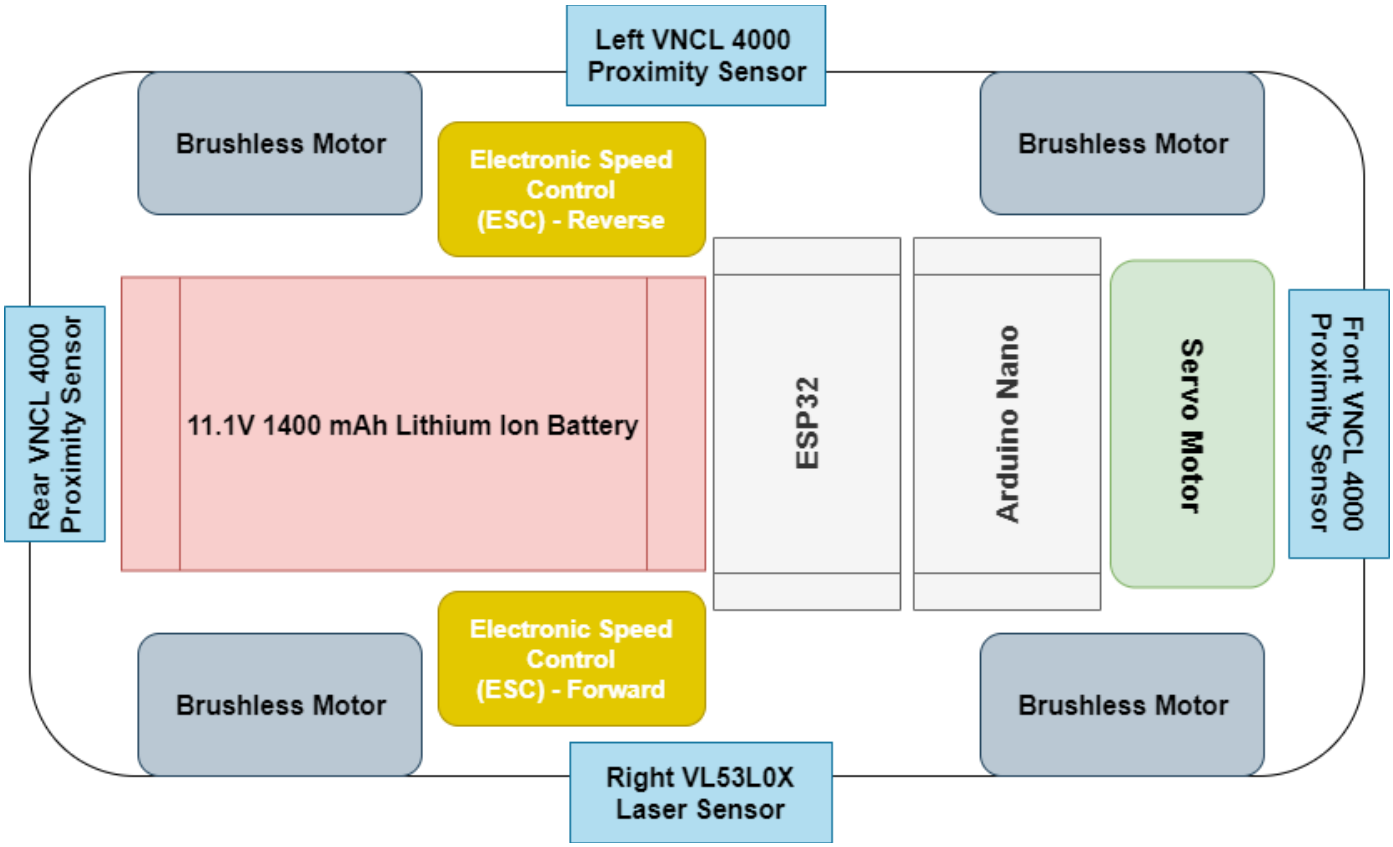
To be able to detect distance, sensors were placed on the front, back, and sides of the car. These sensors were chosen because they were small enough to fit inside of the car. The VCNL4000 is a proximity sensor. It is capable of detecting changes in ambient light and detecting when it approaches a surface. This allows the car know when it approaches a wall, but only when the car gets very close to the wall, and it does not accurately measure the distance to the wall. This device was used on the back and sides of the car.

The VL53L0X is a laser sensor, which can accurately measure distance up to 200mm. This device was used on the front of the car and allowed the car to measure how far it was away from a wall. Both devices are I2C devices and are run off of the Arudino Nano's I2C lines as shown in the schematic below. Because multiple VCNL4000 sensors were used, they were automatically assigned the same i2C address as shown in the schematic below.





The schematic shows a breakout for the arduino nano. Due to the size constraints of the car, using a breadboard to connect all of the sensors to the arduino would have been impractical. Instead, a small breakout that mounts directly on top of the arduino was made. The resistors at J9 were for tail lights for future work on the car. A 5V voltage regulator was used to regulate the 11.1V from the battery to usable 5V to power both the Arduino and the ESP32. Below is a diagram of the physical layout of all the components in the car.



Step 8: Setting up Autonomous driving

Using the parameters specified in the project document, the car to parallel park in the given space. The autonomous driving was carried out in stages. The car would use its sensors to detect when it got too close to the wall and trigger the next stage of parallel parking and obstacle avoidance.

The limiting factor of this challenge was the turning radius of the car, and controlling the car when stopping. The brushless motors used in this car can spin very quickly, but do not have a lot of torque. Also, the weight of the car is pretty substantial, especially when being driven by a single motor. As a result, when changing direction, the car would roll further than intended, because the motor acting in the opposite direction would not be able to stop the car immediately as intended.

Below is the code for autonomous driving.

```
#include "BluetoothSerial.h"
#include <ESP32Servo.h>
```

```
BluetoothSerial SerialBT;
Servo stearingServo;

int freq = 250;
int channel1 = 4;
int channel2 = 5;
int resolution = 10;
int durtyCycle = 256;

char controllerInChar = 'S';
int speedState = 0;

int servoPin = 18;
int servoCenter = 40;
int servoRight = 50;
int servoLeft = 26;
int servoCrnt = 40;

void setup() {

  Serial.begin(115200);
  if(!SerialBT.begin("Ferrari")){
    Serial.println("An error occurred initializing Bluetooth");
  }
  ledcSetup(channel1, freq, resolution);
  ledcSetup(channel2, freq, resolution);
  ledcAttachPin(12, channel1);
  ledcAttachPin(27, channel2);
  ledcWrite(channel1,125);
  ledcWrite(channel2,125);

  pinMode(34,INPUT);
  pinMode(35,INPUT);
  pinMode(33,INPUT);
  pinMode(32,INPUT);

  stearingServo.attach(servoPin, 500, 2400);

}

void loop() {
  //Bultooth stuff
  /*
  while(SerialBT.available()){
    controllerInChar = SerialBT.read();
  }
```

```
if(controllerInChar == 'S'){
  servoCrnt = servoCenter;
}
else if(controllerInChar == 'R'){
  servoCrnt = servoRight;
}
else if(controllerInChar == 'L'){
  servoCrnt = servoLeft;
}
else if(controllerInChar == 'F'){
  speedState += 1;
  controllerInChar = 'S';
}
else if (controllerInChar == 'B'){
  speedState -=1;
  controllerInChar = 'S';
}
*/

switch (speedState){
  case -1://reverse
  ledcWrite(channel1, 256);//forward
  ledcWrite(channel2, 280);//Back
  break;
  case 0://stop
  ledcWrite(channel1, 256);//forward
  ledcWrite(channel2, 256);//Back
  break;
  case 1://slow
  ledcWrite(channel1, 280);//forward
  ledcWrite(channel2, 256);//Back
  break;
  case 2:// medium
  ledcWrite(channel1, 350);//forward
  ledcWrite(channel2, 256);//Back
  break;
  case 3:
  ledcWrite(channel1, 512);//forward
  ledcWrite(channel2, 256);//Back
  break;
}


//Stearign
if(digitalRead(34)==HIGH and digitalRead(35)== LOW){//Left
  servoCrnt = servoLeft;
}
```

```
  else if(digitalRead(34)==LOW and digitalRead(35)== HIGH){//Right
    servoCrnt = servoRight;
  }
  else if(digitalRead(34)==LOW and digitalRead(35)== LOW){//Naw
    servoCrnt = servoCenter;
  }
  else{//Something is going wrond just don't turn
    servoCrnt = servoCenter;
  }

  //Stearign
  if(digitalRead(32)==HIGH and digitalRead(33)== LOW){//Forward
    speedState = 1;
  }
  else if(digitalRead(32)==LOW and digitalRead(33)== HIGH){//Reverse
    speedState = -1;
  }
  else if(digitalRead(32)==LOW and digitalRead(33)== LOW){//Naw
    speedState = 0;
  }
  else{//Something is going wrond just don't do anything
    speedState = 0;
  }

  stearingServo.write(servoCrnt);

  Serial.print("In: ");
  Serial.print(controllerInChar);
  Serial.print("    Servo Pos: ");
  Serial.print(servoCrnt);
  Serial.print("    Speed: ");
  Serial.println(speedState);
}
```