Mhackers Embedded Systems Demo Night Presentation

Remote Controlled Car

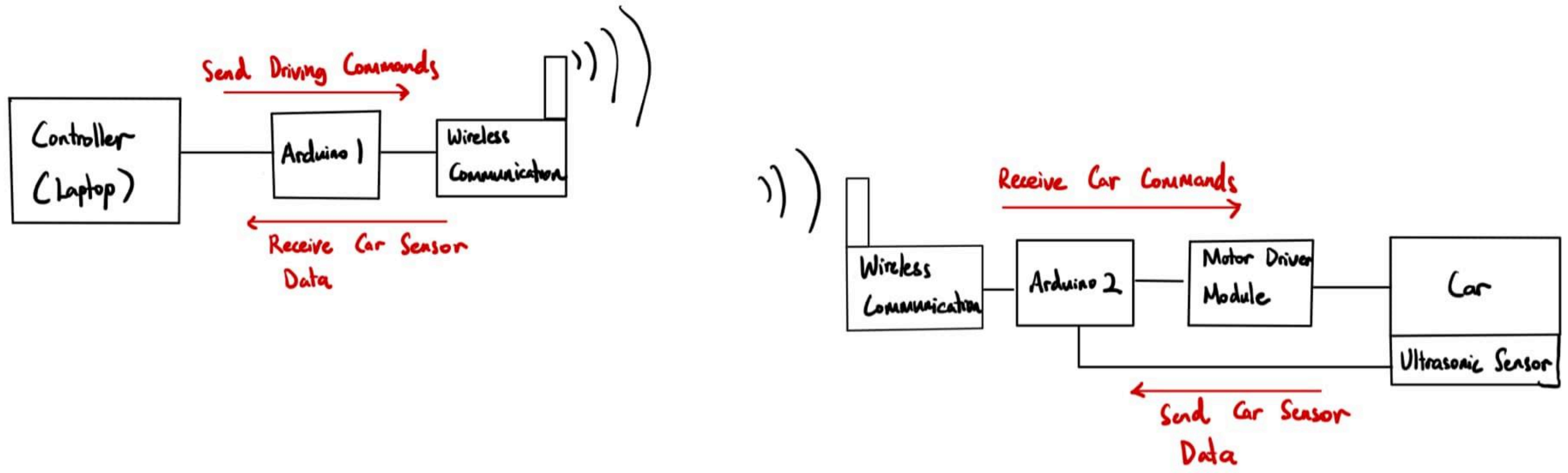
Purpose of Project:

1. To Learn About Wireless Communications and IoT-based Embedded Systems
2. To Learn Arduino-Controlled Motored Car Designs
3. To Design a Remotely Controlled Car for Various Applications (Vacuum Cleaner like Roomba)

Idea of the Project:

The idea of the project is to build two Transceiver modules (one on the remote controller, one on the car itself) to allow communication between two Arduino boards. The Arduino board attached to the car receives driving commands (right wheel forward/stop, left wheel forward/stop) from the controller Arduino and controls the motor driver module that moves the left and right wheel. An ultrasonic sensor is connected to the car to measure the distance between the car and the first obstacle. The sensor data is then sent back to the controller Arduino to provide users with real-time data. A further extension of this project is to connect the controller Arduino with a computing server that analyzes the sensor data and automates the transmitting of driving commands to the car Arduino.

High-Level Diagram



Wireless Communication (nRF24L01 Transceiver Module)

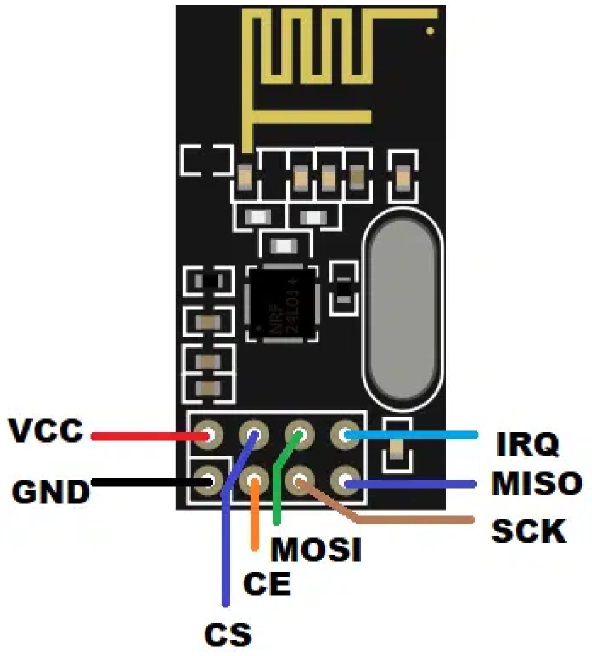
The nRF24L01+ is a 2.4 GHz wireless transceiver that operates through SPI (Serial Peripheral Interface). The module can transmit and receive data wirelessly through its built-in antenna port. You can only transmit or receive (but not both) at any given moment. It has a special piping capability wherein you can have six transmitters talking to 1 receiver through the Multiceiver approach. Each transmitter should have its own unique address, and this should be matched with the receiver.

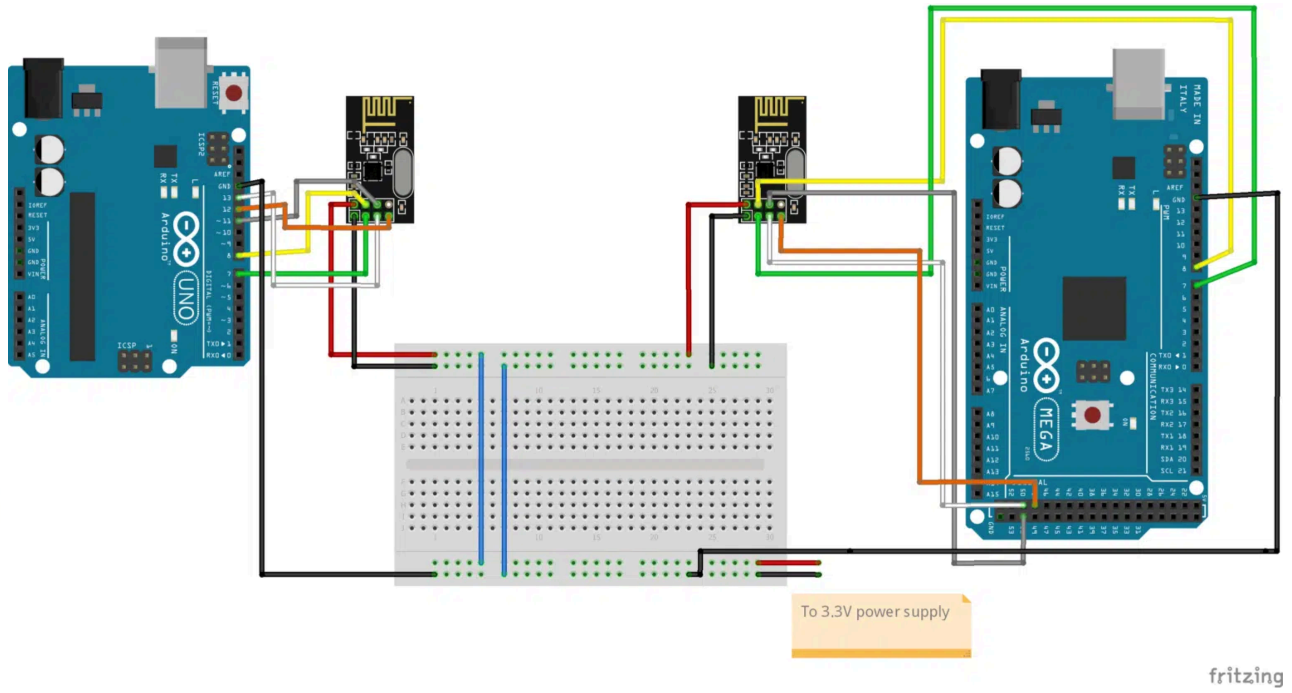
We plan to use this Transceiver Module to transmit payloads encoded with driving commands from the controller Arduino to the car Arduino. We also plan to using the transceiver module to transmit sensor data back to the controller via the Transceiver module.

The hardest part about using the transceivers is trying to establish connections between two transceiver modules using existing libraries. We are currently experimenting with different libraries and currently use the RF24 library from Arduino.

Code to establish a Connection Pipe

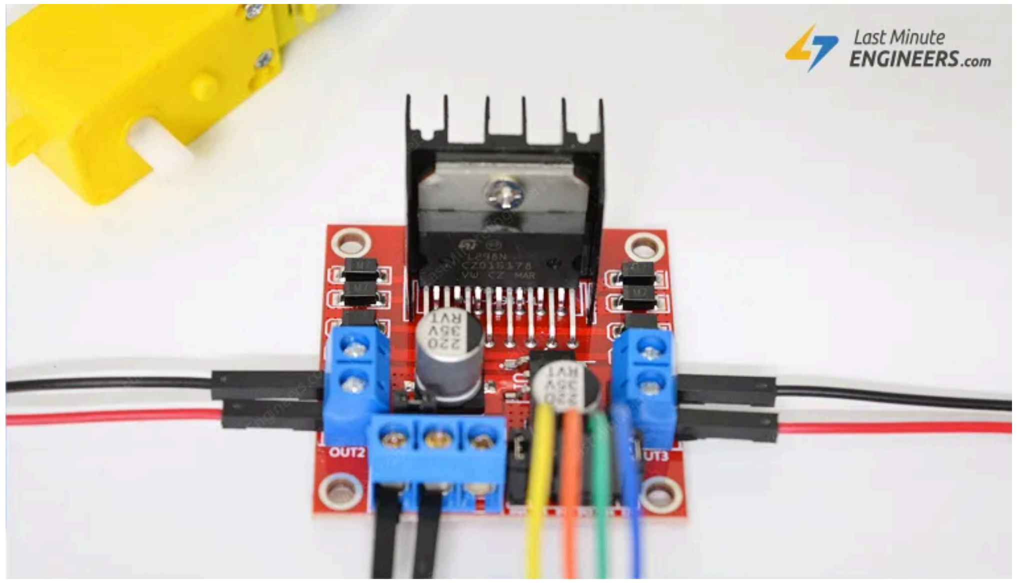
```
#include <SPI.h>
#include "printf.h"
#include "RF24.h"

#define CE_PIN 7
#define CSN_PIN 8
// instantiate an object for the nRF24L01 transceiver
RF24 radio(CE_PIN, CSN_PIN);

// Let these addresses be used for the pair
uint8_t address[][6] = { "1Node", "2Node" };

// to use different addresses on a pair of radios, we need a variable to
// uniquely identify which address this radio will use to transmit
bool radioNumber = 1;  // 0 uses address[0] to transmit, 1 uses address[1] to transmit

// Used to control whether this node is sending or receiving
bool role = false;  // true = TX role, false = RX role

// For this example, we'll be using a payload containing
// a single float number that will be incremented
// on every successful transmission
float payload = 0.0;

void setup() {

  Serial.begin(115200);
  while (!Serial) {
    // some boards need to wait to ensure access to serial over USB
  }

  // initialize the transceiver on the SPI bus
  if (!radio.begin()) {
    Serial.println(F("radio hardware is not responding!!"));
    while (1) {}  // hold in infinite loop
  }
  // print example's introductory prompt
  Serial.println(F("RF24/examples/GettingStarted"));

  // To set the radioNumber via the Serial monitor on startup
  Serial.println(F("Which radio is this? Enter '0' or '1'. Defaults to '0'"));
  while (!Serial.available()) {
    // wait for user input
  }
  char input = Serial.parseInt();
  radioNumber = input == 1;
  Serial.print(F("radioNumber = "));
  Serial.println((int)radioNumber);

  // role variable is hardcoded to RX behavior, inform the user of this
  Serial.println(F("*** PRESS 'T' to begin transmitting to the other node"));

  // Set the PA Level low to try preventing power supply related problems
  // because these examples are likely run with nodes in close proximity to
  // each other.
  radio.setPALevel(RF24_PA_LOW);  // RF24_PA_MAX is default.

  // save on transmission time by setting the radio to only transmit the
  // number of bytes we need to transmit a float
  radio.setPayloadSize(sizeof(payload));  // float datatype occupies 4 bytes

  // set the TX address of the RX node into the TX pipe
  radio.openWritingPipe(address[radioNumber]);  // always uses pipe 0

  // set the RX address of the TX node into a RX pipe
  radio.openReadingPipe(1, address[!radioNumber]);  // using pipe 1

  // additional setup specific to the node's role
  if (role) {
    radio.stopListening();  // put radio in TX mode
  } else {
    radio.startListening();  // put radio in RX mode
  }
}  // setup
```

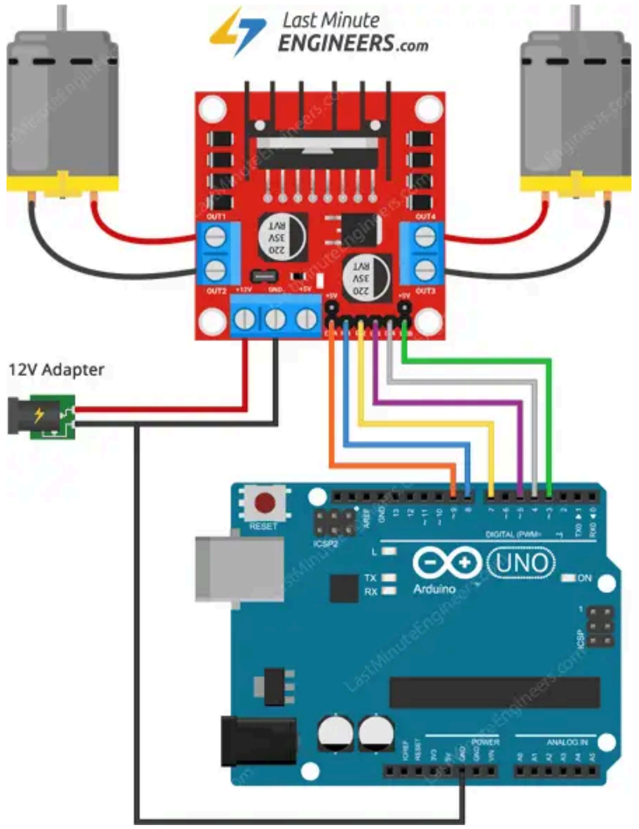

VCC
GND
IRQ
MISO
SCK
MOSI
CE
CS


To 3.3V power supply
fritzing

L298N Motor Driver Module

The Motor Driver Module is the main component that allows us to use Arduino to interface with the wheels of the car. The Motor Driver Module interfaces with the DC motors which powers the wheels of the car to stop, reverse, or move forward. It can control the speed and spinning direction of two DC motors.



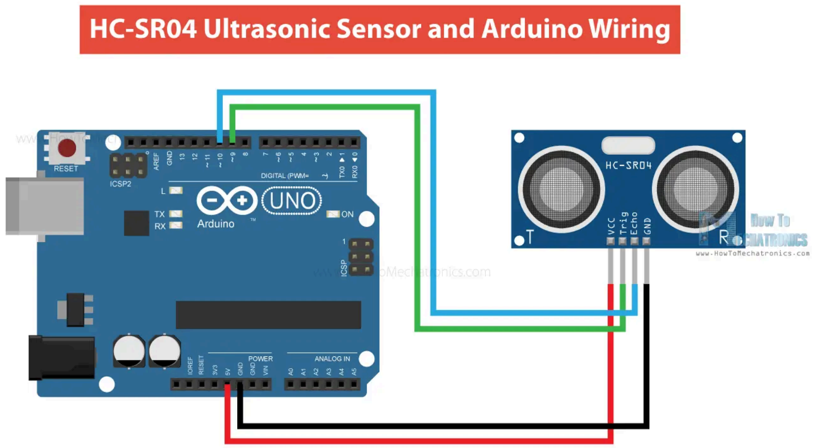


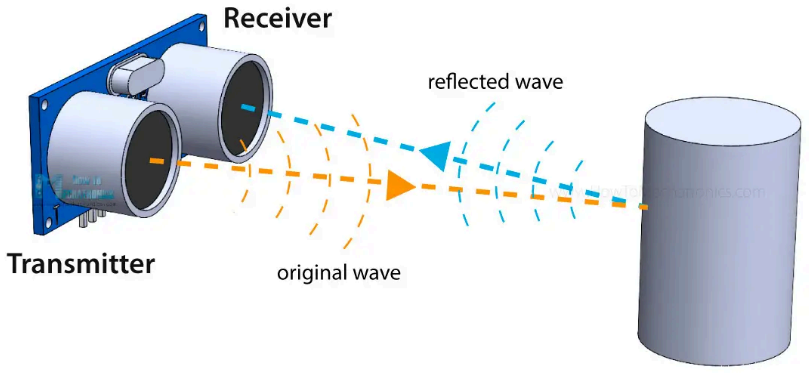
Code for Motor Driver Module

```
#define Trig 8
#define Echo 9


const int in21 = 4;    // L298N-2 pin 4
const int in22 = 5;    // L298N-2 pin 5
const int in23 = 6;    // L298N-2 pin 6
const int in24 = 7;    // L298N-2 pin 7
const int enA = 10;    // L298N-2 pin 10
const int enB = 11;    // L298N-2 pin 11

void setup()
{
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);

  pinMode(in21, OUTPUT);
  pinMode(in22, OUTPUT);
  pinMode(in23, OUTPUT);
  pinMode(in24, OUTPUT);

}
unsigned int impulseTime=0;
unsigned int distance_sm=0;

void loop()
{
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10); // 10 микросекунд
  digitalWrite(Trig, LOW);
  impulseTime=pulseIn(Echo, HIGH);
  distance_sm=impulseTime/58;

  if (distance_sm>25)
  {

      digitalWrite(in21, LOW);
      digitalWrite(in22, HIGH);
      analogWrite(enA, 60);

      analogWrite(enB, 60);
      digitalWrite(in23, HIGH);
      digitalWrite(in24, LOW);
  }
  else
  {

      digitalWrite(in21, HIGH);
      digitalWrite(in22, LOW);

      analogWrite(enA, 100);

      analogWrite(enB, 100);

      digitalWrite(in23, HIGH);
      digitalWrite(in24, LOW);
    delay(1100);

  }

  delay(50);

}
```

Ultrasonic Sensor

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves. It is a device that uses a transducer to send and receive ultrasonic pulses that relay back information about an object’s proximity. With some calculations from the ultrasonic sensor pulse data, we are able to calculate the distance of the obstacles in closest proximity.

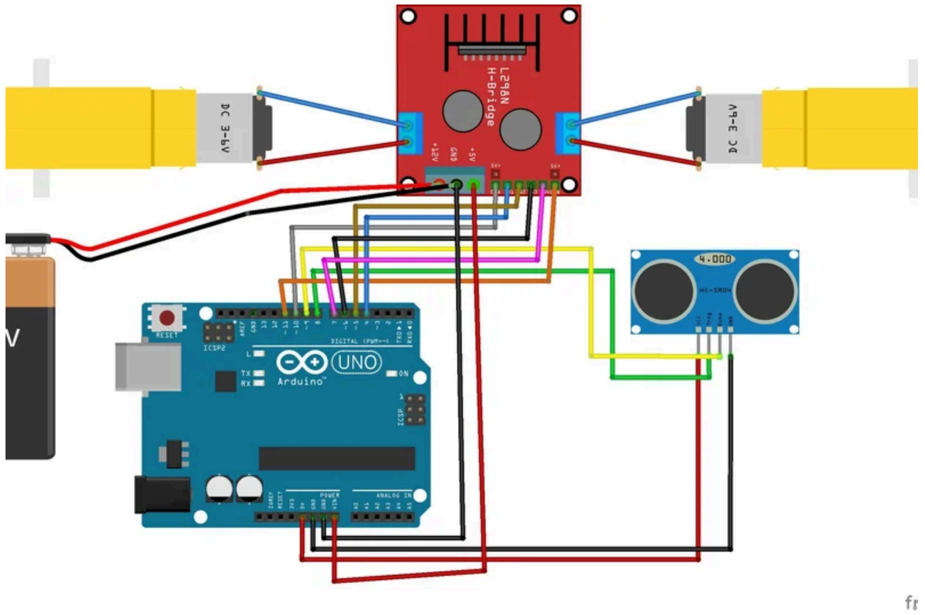


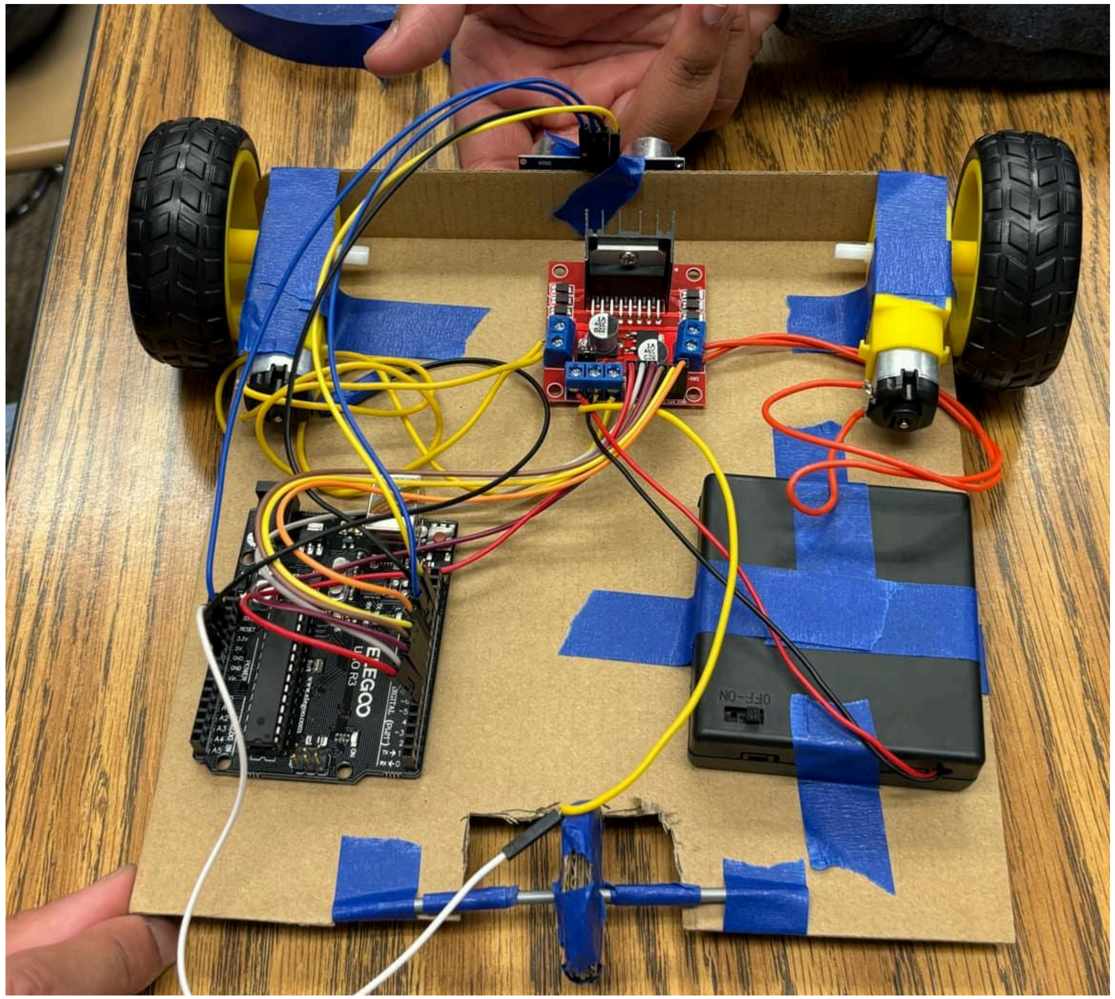


Code to Interact with Ultrasonic Sensor

```
const int trigPin = 9;
const int echoPin = 10;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.println(distance);
}
```

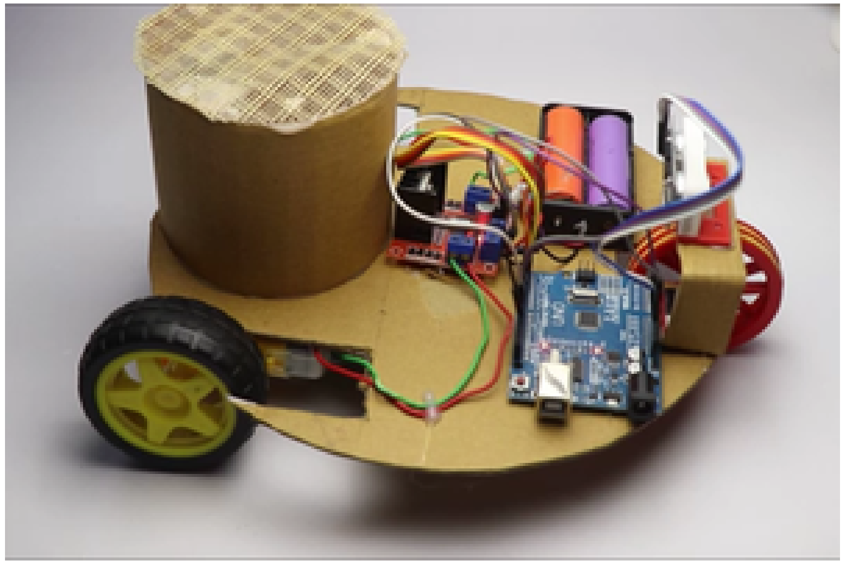
Full Car Prototype





Potential Applications of a Remote-Controlled Car

- Remote Controlled Vacuum Cleaner



Action Plans for Next Semester

1. Debug and Get Wireless Communications Working
2. Fully Assemble the Car, Install Fan for Vacuum Cleaning Purposes
3. Integrate Wireless Communications to the Car
4. Integrate Computer Vision Models that automate driving commands based on sensor data