Task PCDHD: Project
ROBOT ROVER PROJECT
Pipe Inspection

by

Paul Collins

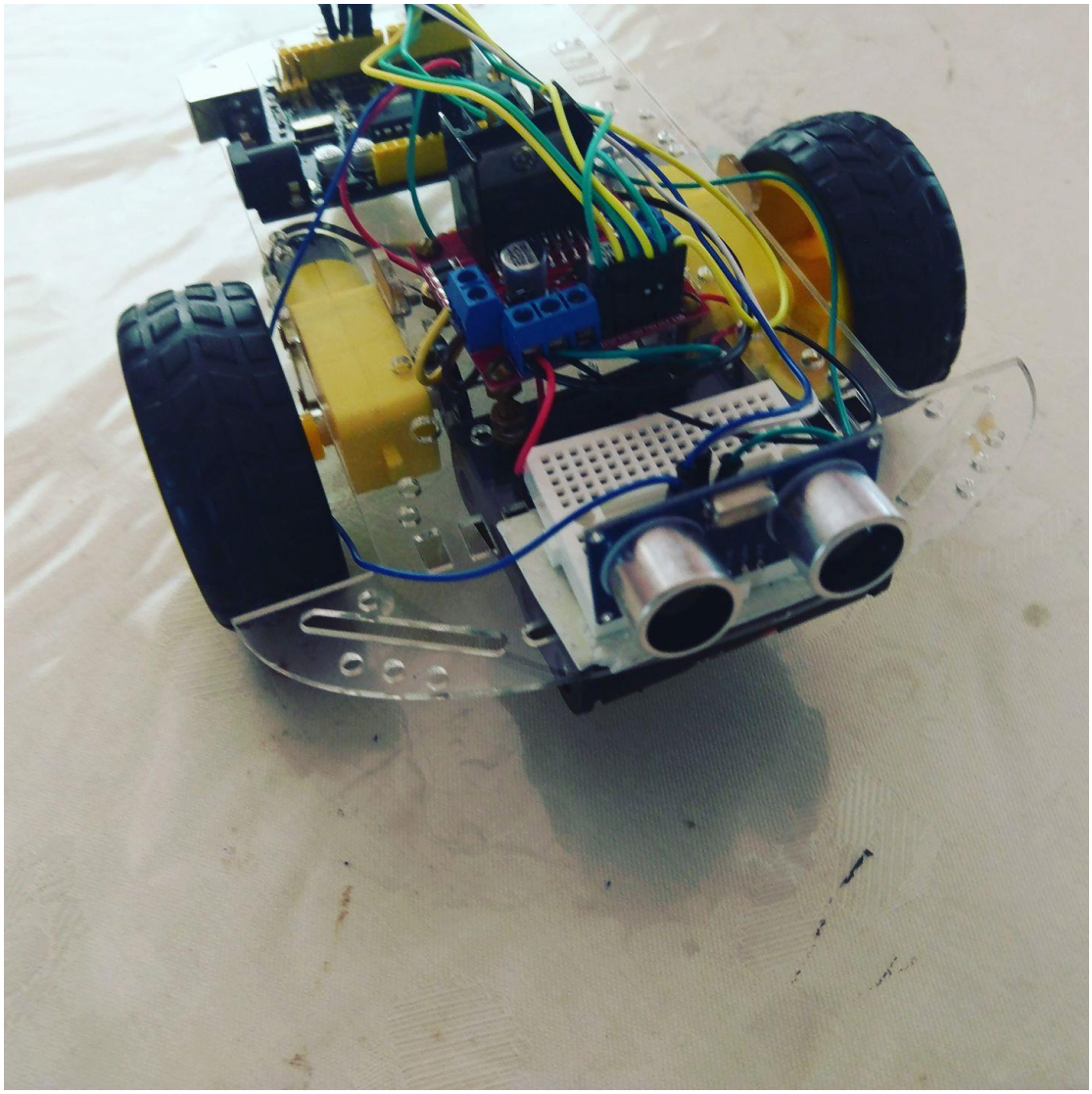
Problem Statement:

Pipes need to be checked for any blockages and cracks. The level of light and the temperature inside the pipes need to be obtained and sent to the cloud in real-time. Some of the smaller pipes are too small for workers to crawl through them to look for any possible pre-installation blockages or impairments/cracks to the interior surface. There are some artefacts on the bottom surface within some pipes that would obstruct waterflow. There are also cracks which let in light and there is excessive heat in some pipes.

Proposed Solution: Drainage Pipe Inspection Robot

A mobile rover robot equipped with ultrasonic, light and temperature sensors will be used to traverse through the pipes looking for any obstructions and send back the sensor readings dynamically - light level and temperature to cloud - webhook . If no obstruction is found, the robot will reach the opposite end of the pipe however, if any blockage or obstruction is found then the robot will not reach the end of the pipe and instead it will turn around and return to the entry of the pipe. The robot will report any light level and temperature level to the webhook cloud using photon particle.

Rover Chassis



The Robot Rover uses an Ultrasonic Sensor to detect any obstacles in the drainage pipe. It reads the output from the Ultrasonic transducer and drives the Robot Rover forward if there are no obstacles in front of it. Should an obstacle be sensed the robot will turn 90

degrees (right or left) and continue in the new direction if no other obstacles are detected. (Sensor model HC-SR04)

FUNCTIONAL DESCRIPTION

For my Robot Rover Project, the task is simple: roam through a water drainage pipe and if the pipe contains no blockages, the Rover will traverse its way successfully through the entire length of the pipe until it exits at the other end of the pipe. It just drives straight ahead until it detects an object blocking its path. At which point, it turns and comes back indicating that there is a blockage at a certain distance within the pipe.

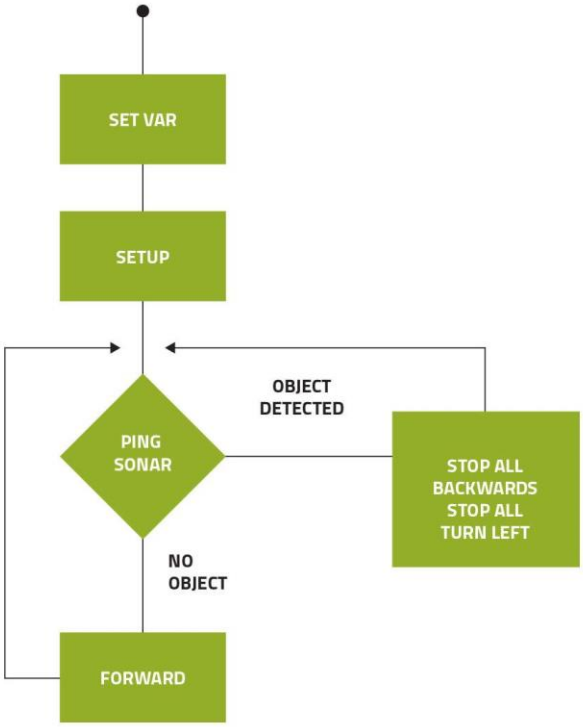
The Robot has two motors which are driven via a dual H-bridge motor controller and an ultrasonic distance sensor is used to detect obstacles. The Rover can be built on any rolling chassis; I selected a low cost one that is readily available from a variety of retailers around the world.

Fundamentally any robot exists as a system. The system is a set of inputs and outputs with a decision-making process in the middle. Arguably you could set your robot on a predefined pathway assuming nothing will go outside of your control.

Any system or robot will require various inputs to be evaluated before making decisions. Inputs can be simple or advanced, from pressure, air speed, temperature, angle, G-force or anything else you can measure. In this case, I am using distance as the input metric. I could achieve this through several means, but I will be using a cost-effective and easily obtainable ultrasonic transducer.

As you can see from the diagram below, the system follows a very simple strategy for obstacle avoidance. This is designed to provide very simple functionality to

demonstrate the principles in this project.



TECHNICAL SPECIFICATION

This robot is quite simple but still requires some pre-planning. Upon Power-up the robot will drive itself forward. It does this by switching both sets of drive motors in a forward direction. As the code loops, it uses the ultrasonic sensor to send out a pulse to which it measures the distance of any object in front. Once an object is detected, it stops the robot and enters a conditional section of code that sends it in the opposite direction, then manoeuvres to switch its direction. Once complete it reverts to the original routine and moves in a forward direction until it encounters another obstacle.

THE UNO BRAIN: In this instance, the Arduino UNO is my microprocessor of choice, as it is a simple, open source and a low-cost microprocessor. It also has the power required.

ULTRASONIC DISTANCE SENSOR:

To see where it's going the Rover uses a HC-SR04 ultrasonic distance sensor.

The HC-SR04 has 4 pins – VCC, Trig, Echo, and GND. You need to mount the sensor to the mini breadboard to steady it on the robot as well as to simplify all the necessary connections.

VCC on the sensor should be connected to 5v so use a male / male jumper to connect the distance sensor's VCC to the power rail on the breadboard.

This is one of the most simple and cost-effective methods for measuring distance; however, it comes with some caveats. The Ultrasonic works by sending out a timed soundwave well beyond what the human ear can hear. If there is an object in front of the sound wave, it is reflected off the object. This bounce back is received by the unit and the time taken for this to occur is then calculated using the speed of sound, to work out how far away the object is.

So how do we calculate the distance? The speed of sound is 0.034cm/µs or 340m/s so if the object is 10cm away it will take 294microseconds. However, what you get back from the echo pin will be double that number, due to the time it takes for the sound wave to return to the unit. To get the distance in centimetres, we need to multiply the received travel time value from the echo pin by 0.034 and then divide it by 2. By using this calculation against the known speed of sound, we effectively gain a distance measurement.

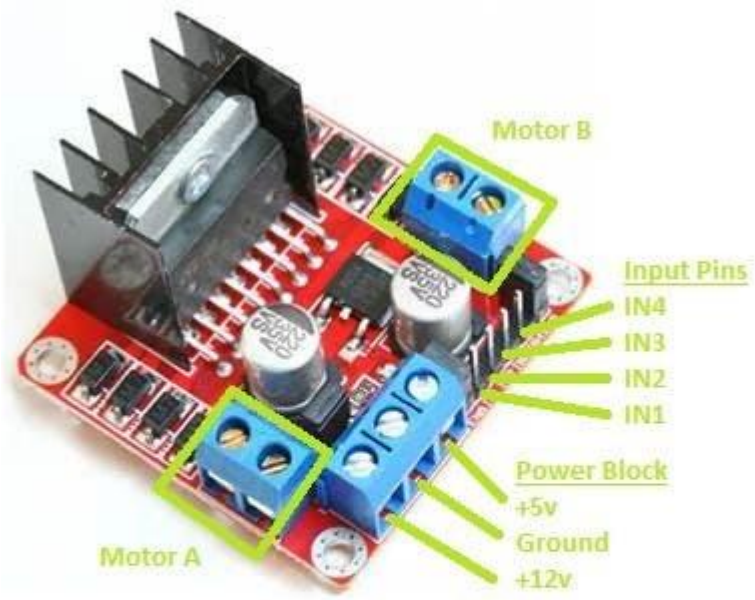
L298N MOTOR CONTROLLER

For this project, I will use the L298N Dual H Bridge package. In this instance, it is the duinotech, XC4492, which is an inexpensive module that makes utilisation incredibly simple. It operates as a two-channel device, which can let us operate the left and right pairs of motors. You can easily use two or four motors. I will use two channels and enable skid-steer or tank track-style direction changes.

The L298N works by sending a PWM signal (from 0-255) to the enable pin of the channel you are using. It then specifies the direction required by sending the two other pins high, low or a combination of both. This is repeated for the second channel, thus requiring six outputs from the UNO to control the motors.

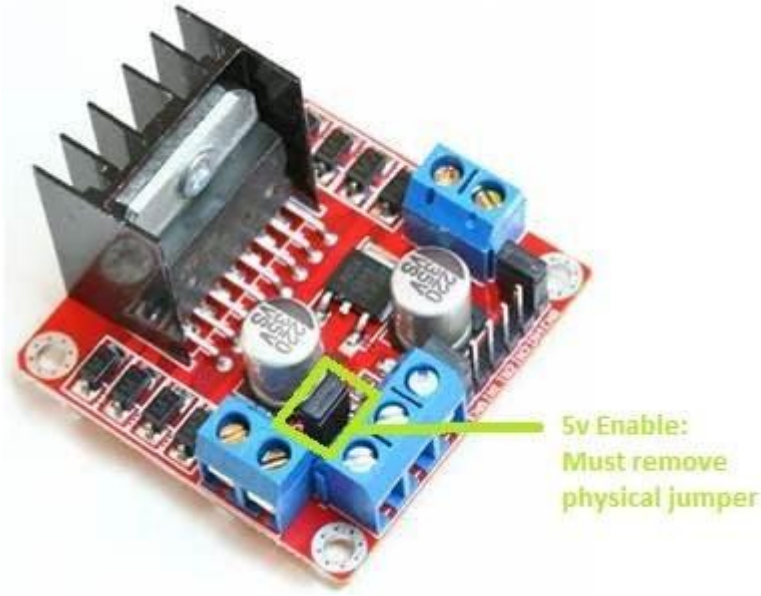
On top of the standard VCC and ground, the L298N has a second voltage input VMS, which is used to supply the power for the motors. Each channel requires three inputs: the ENA and the IN1 & IN2 in the case of motor A.

The L298N motor driver allows you to spin the motors forwards AND backwards using a handful of pins. First, connect the two wires you secured to each motor in the prior step to a pair of motor terminals – the red and black wire from one motor to ‘Motor A’ and the red and black wire from the other motor to ‘Motor B’. Polarity isn’t important and you can always switch the order of the wires later if your motor ends up spinning the wrong way when you deploy your code. Next, connect the wires from the 4 x AA battery holder to the power terminals – red to +12v input and black to Ground; the 4 AA batteries are the power source for the motors. Also be sure to run a wire from the Ground terminal on the L298N to a GND.



The L298N was designed to support a single power source for both the motors and the microcontroller / computer. The full voltage from the power source is routed to the motors. At the same time, the voltage from the power source is converted and regulated to 5v for the microcontroller / computer and is supplied via the +5v terminal on the power block.

First, remove the physical jumper – labelled ‘5v enable’ in the photo – from the L298N. This sets the motor controller logic to be powered by the UNO.



MOTOR OUTPUTS:

This project could easily be built with two or four motors driving the unit. The 4WD is not truly independent; it could be achieved by using a four-channel motor driver, then providing a sensor on each wheel for detecting slippage. Power could be removed or decreased to that wheel and apply more power to other wheels.

For this build, I will be using basic DC hobby motors. Each motor is driven by the control module with simple DC. They are connected to a small plastic gear box, which is connected to each wheel.

Hardware components:

THE PARTS LIST REQUIRED

	JAYCAR	ALTRONICS
1 x Arduino UNO	**XC4410**	**Z6240**
1 x HC-SR04 Ultrasonic Distance Sensor Module **HC-SR04 ultrasonic distance sensor**	**XC4442**	**Z6322**
1 x L298n Motor Control Module **L298N motor controller**	**XC4492**	**Z6343**

1 x Chassis & Motor Kit KR3162 K1092

Robot car chassis kit which includes a base, motors, and wheels

Jumper wires (generic)

Jumper wires – both male/male and male/female

1 x Mini breadboard

Tools:

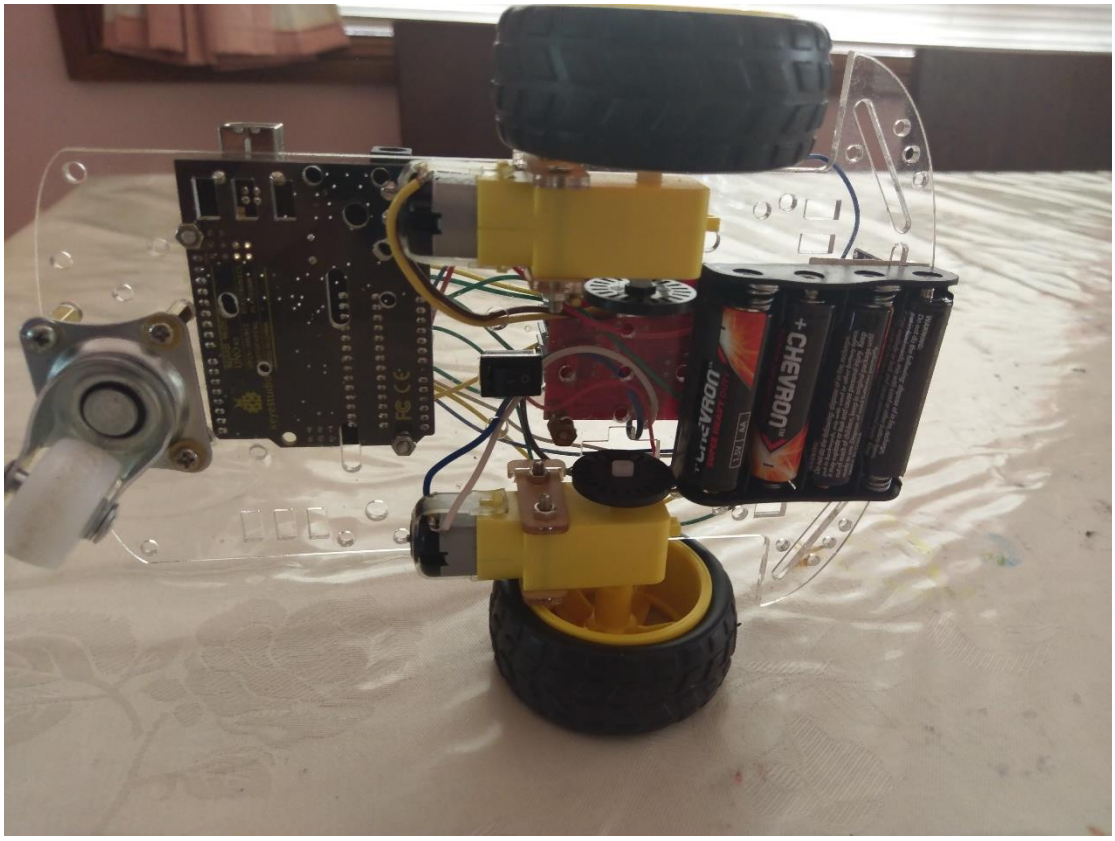
1. Multi meter
2. Phillips head screwdriver
3. Small needle nose pliers
4. Wire stripper
5. Soldering iron

Parts:

Robot chassis kit; optional 4 x AA battery holder with on/off switch

There are several robot kits on the market that will work with this project. You just need a kit with two driven wheels and a third for balance.

You need to solder wires to your motors.



THE CONNECTIONS AND WIRING DIAGRAM

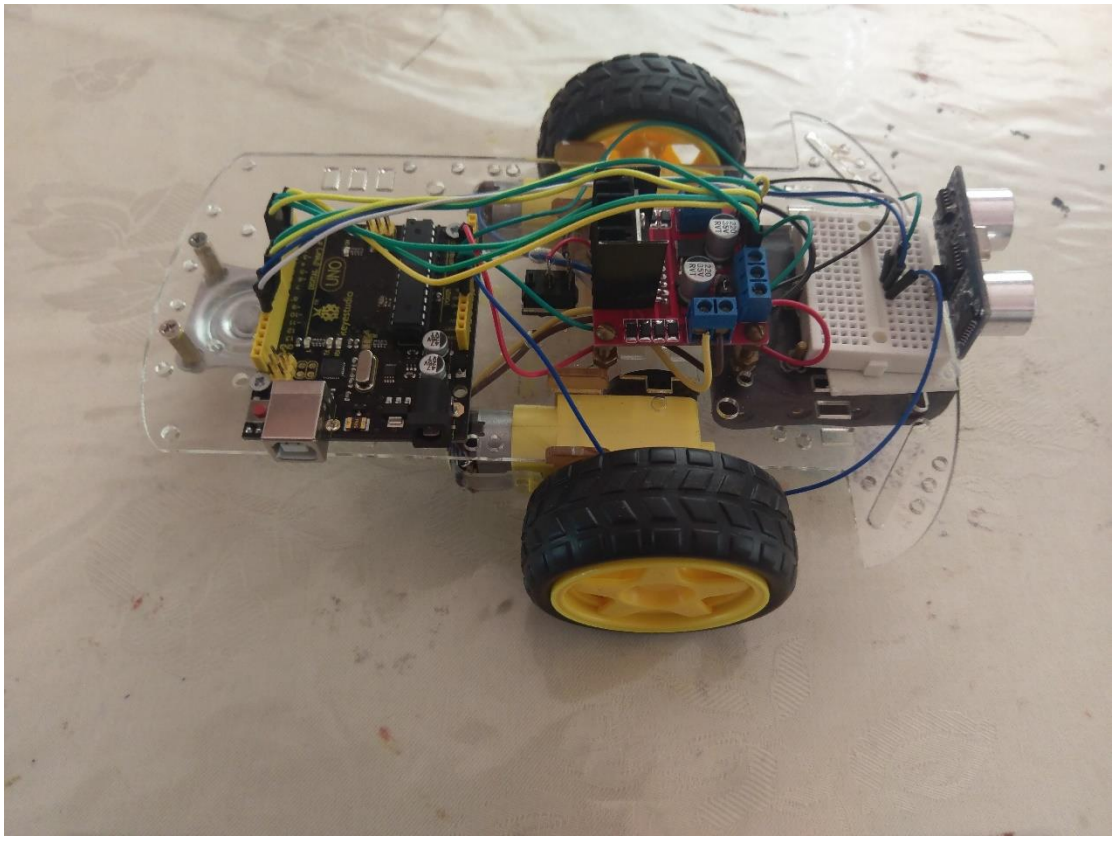
There are no external discrete components required. Connect the trig pin from the US to pin 11, and echo to pin 3. VCC to 5V and GND to GND on the UNO.

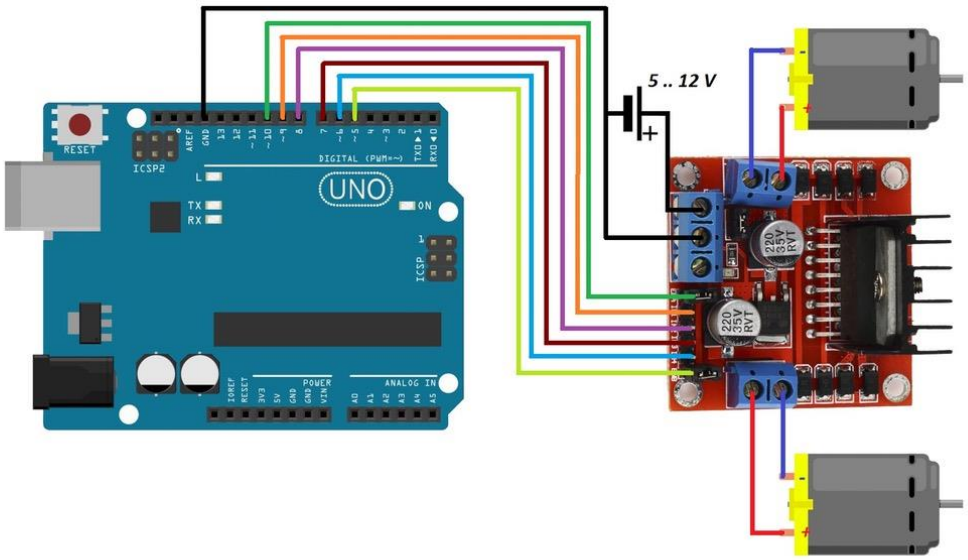
DUAL MOTOR DRIVER	**UNO**
ENA	10
IN1	9
IN2	8

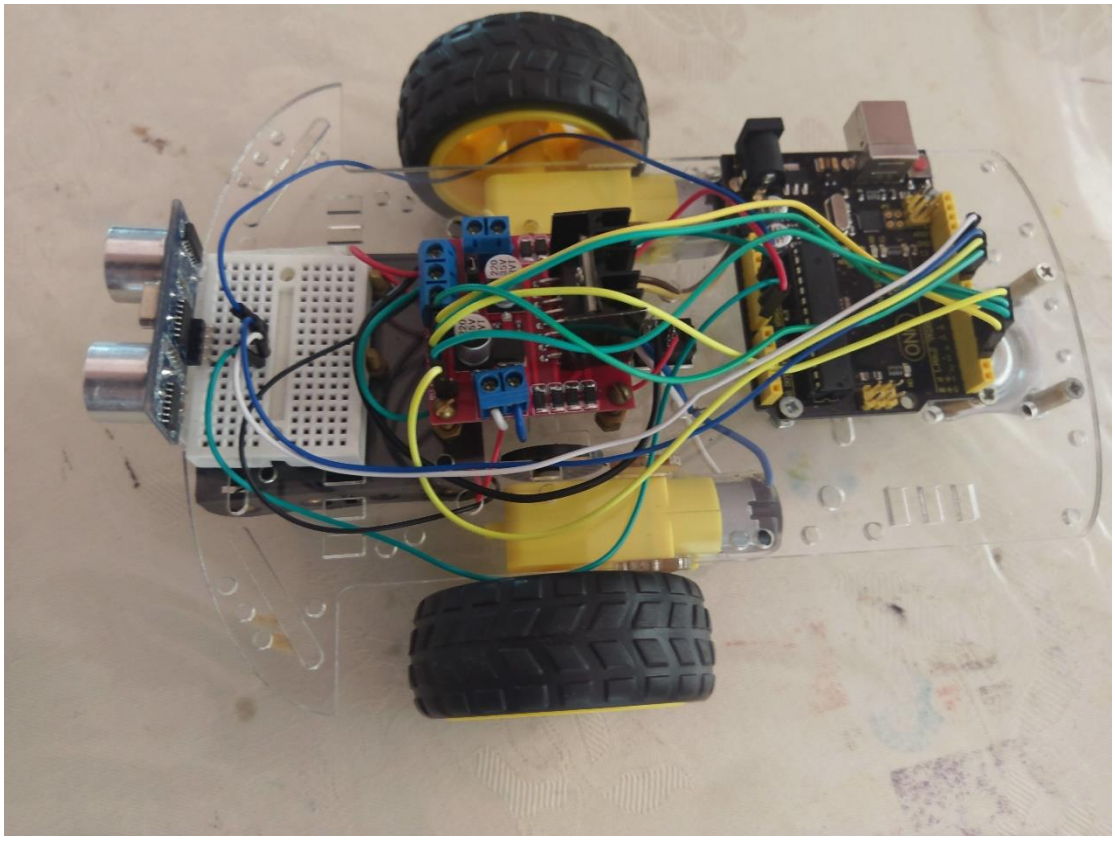
ENB	5
IN3	7
IN4	6
trig	11
echo	3
US GND	UNO
	GND
L298N	UNO
GND	GND

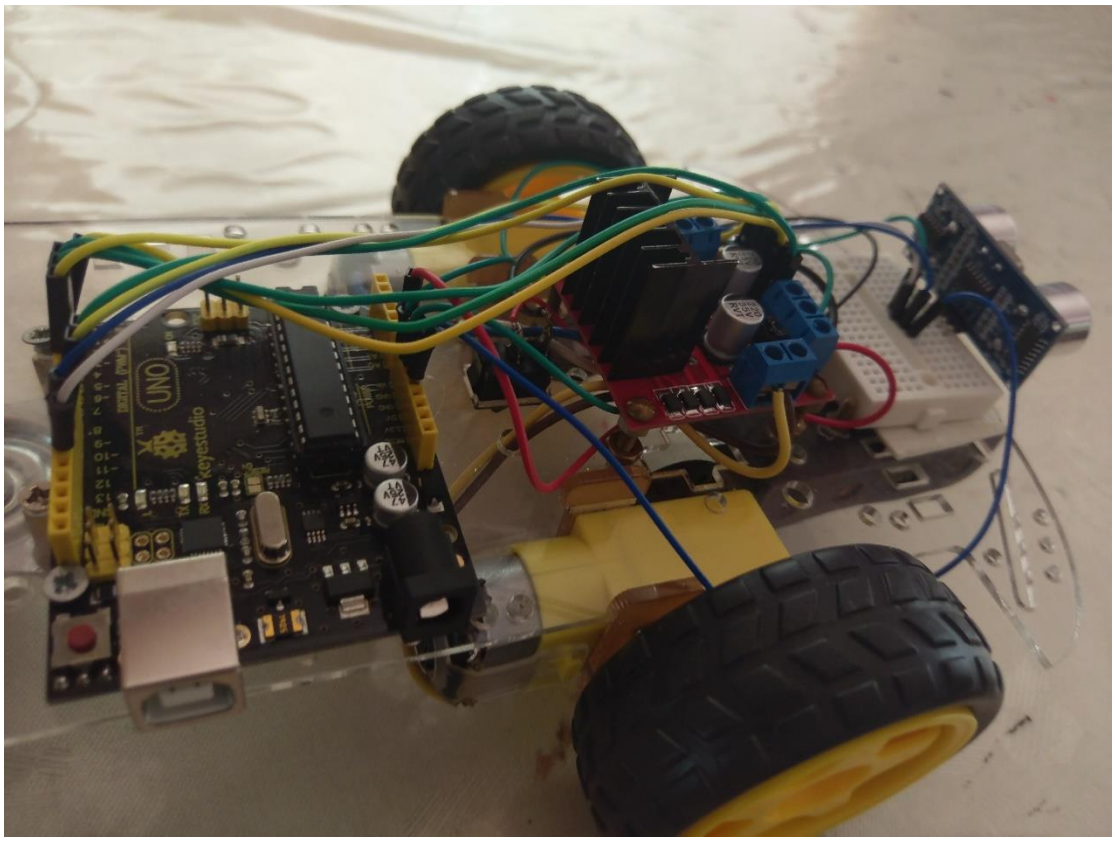
Using jumpers connect the following from the UNO to the L298N. Connect the other GND to the ground on the L298N board.

The motor’s rotation will be dependent on the polarity of the power applied. Best to colour-code each channel when soldering the wires on. If using two motors then connect each one to the motor to A and B screw terminals.




5 .. 12 V
RESET
ICSP2
UNO
DIGITAL (PWM=~)
TX
RX
ON
ICSP
POWER
ANALOG IN
220
35V
RVT




UNO
keyestudio

ISSUES:

The Ultrasonic modules work well when you have a large surface to reflect off, and the unit is perpendicular to the object. If the object is smaller, it will need to be closer before the sound wave will reflect and can be calculated correctly. If the unit is at an angle, then the reflection does not bounce back at all, and it will often require the robot to be incredibly close before it receives the information it requires to calculate the distance.

POSSIBLE ENHANCEMENTS TO MAKE TIME PROVIDING:

- A revolving turret for the ultrasonic sensor.
- Code to mimic a PWM signal on the Analog pins to adjust the Rover's speed.

THE CODE:

Understanding the code will help with debugging should the robot not work perfectly.

The only variable you need to change is max distance, which is the distance at which the robot will stop and turn.

https://github.com/TheCollo67/Project

```
// ---------------------------------------------------------------------------
// NewPing library sketch that does a ping about 20 times per second.
// ---------------------------------------------------------------------------
// Code to run the ROVER Robot via Ultrasonic detection
//
// By Paul Collins. Revised version.
//
// NOTE NewPing library has been modified to allow for timer interrupt
//   clash between NewPing and IRremote library files.

#include <NewPing.h>


#define TRIGGER_PIN  11  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     12  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.
#define MotorA_1     2
```

```
#define MotorA_2     3
#define MotorB_1     5
#define MotorB_2     4
#define EnableA      9
#define EnableB      10


NewPing sonar1(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing
setup of pins and maximum distance.



long dist1;
int wait = 200;


void setup() {
  Serial.begin(9600);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
}









void loop() {


 dist1=sonar1.ping_cm();

 if(dist1==0){
  dist1 = MAX_DISTANCE;
  }


 if(dist1<=40) {
  digitalWrite(EnableA, HIGH);

  digitalWrite(EnableB, HIGH);

  digitalWrite(MotorA_1, LOW);   //Has to be LOW,HIGH
  digitalWrite(MotorA_2, HIGH);
```

```
  digitalWrite(MotorB_1, HIGH);  //HIGH, LOW
  digitalWrite(MotorB_2, LOW);
  delay(400);
  stopMotors();
 }
 else {
  randNumber = random(1000, 2000);
  if randNumber = 1500 stopMotors();
  delay(randNumber);

  digitalWrite(MotorA_1, HIGH);
  digitalWrite(MotorA_2, LOW);
  digitalWrite(MotorB_1, HIGH);
  digitalWrite(MotorB_2, LOW);


  digitalWrite(EnableA, HIGH);
  digitalWrite(EnableB, HIGH);

 }

}

void stopMotors(){
  digitalWrite(MotorA_1, LOW);
  digitalWrite(MotorA_2, LOW);
  digitalWrite(MotorB_1, LOW);
  digitalWrite(MotorB_2, LOW);
  digitalWrite(EnableA, LOW);
  digitalWrite(EnableB, LOW);
}
```

Particle Photon Webhook:

```
// This #include statement was automatically added by the Particle IDE.
#include <Adafruit_DHT.h>

// ----------------------------------------
// Temperature and Light Monitor
// Monitor light, temperature and Humidity
// Particle Photon + Particle.io Webhook + ThingSpeak
//
// ----------------------------------------
#define DHTPIN 2
#define DHTTYPE DHT11 // DHT11 Module
#define publish_cycle 60000 // Only publish every 60 seconds
```

```
const String key = "525ASS1WXKN11V4Y"; // Thingspeak api write key
int led = D7; // Sensor Read Light
int photoCell = A0;
int power = A5;
int light;
double tempF; // Temperature F
double tempC; // Temperature C
double humidity; // Humidity
unsigned int lastPublish = 0;
DHT dht(DHTPIN, DHTTYPE);
void setup() {
    // Set Pin Modes
    pinMode(led,OUTPUT);
    pinMode(photoCell,INPUT);
    pinMode(power,OUTPUT);
    digitalWrite(power,HIGH);
    digitalWrite(led,LOW);
    // Connect variables to particle cloud

    Particle.variable("light", &light, INT);
    Particle.variable("tempF", &tempF, DOUBLE);
    Particle.variable("tempC", &tempC, DOUBLE);
    Particle.variable("humidity", &humidity, DOUBLE);
    dht.begin();
    Serial.begin(9600);
    delay(10000);
} //setup
void loop() {
  unsigned long now = millis();
  digitalWrite(led,HIGH); // Signal read sequence led
  // read sensors
  light = analogRead(photoCell);
  delay(100);
  humidity = dht.getHumidity();
  tempC = dht.getTempCelcius();
  tempF = dht.getTempFarenheit();
  // DHT Read ok?
  if (isnan(humidity) || isnan(tempF) || isnan(tempC)) {
    Serial.println("");
    Serial.println("Failed to read from DHT sensor!");
    Serial.println("humidity=" + String(humidity) + " tempF=" + String(tempF) + "
tempC=" + String(tempC));
    Serial.println("");
    return; // exit loop and try again
  }
  // Display to serial
  Serial.println();
  Serial.print("humidity=" + String(humidity) + " tempF=" + String(tempF) + "
tempC=" + String(tempC) + " light=" + String(light));
  delay(200);
```

```
  // Publish to thinkspeak
  if ((now - lastPublish) > publish_cycle) {
    Particle.publish("thingSpeakWrite_All", "{ \"1\": \"" + String(humidity) + "\"," +
       "\"2\": \"" + String(tempC) + "\"," +
       "\"3\": \"" + String(tempF) + "\"," +
       "\"4\": \"" + String(light) + "\"," +
       "\"k\": \"" + key + "\" }", 60, PRIVATE);
    lastPublish = now;
    Serial.println(" - Published!");
  } else {
      Serial.println();
  }
  digitalWrite(led,LOW);
  delay(2000);
} // loop
```