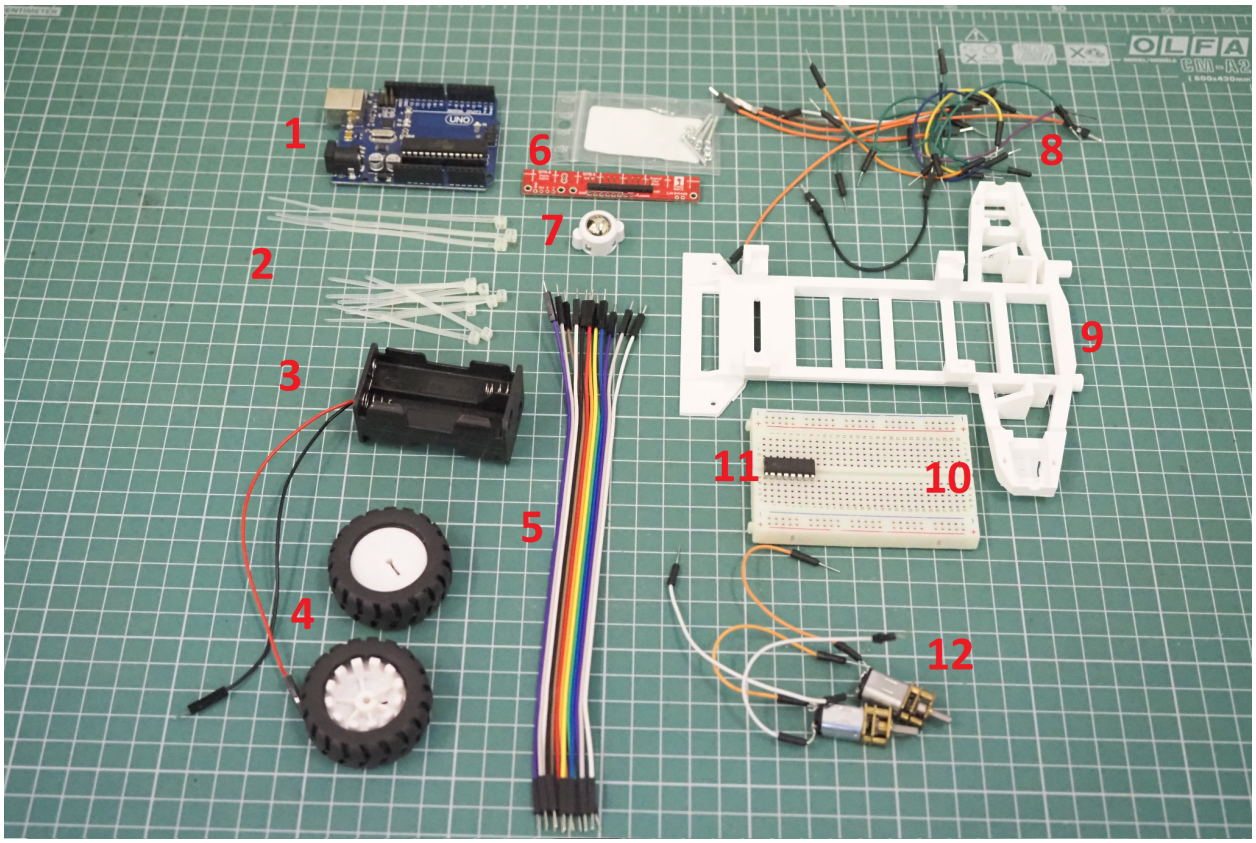
Introduction to robotics

11th lab: Line Follower Assembly Guide (v1.05)

Last update: 07.01.2024, 23:10

1. Components List



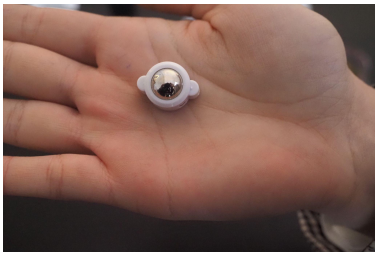
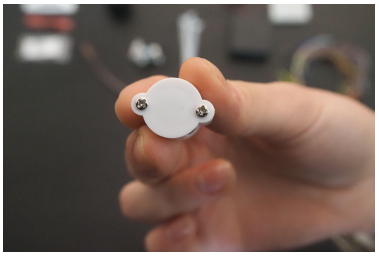
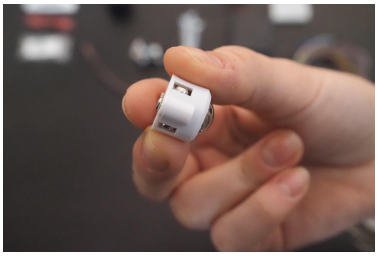
Useful guide: https://lastminuteengineers.com/l293d-dc-motor-arduino-tutorial/

1. Arduino Uno
2. Zip-ties
3. Power source (can be of different shape). In our case, a LiPo battery
4. Wheels (2)
5. Wires for the line sensor (female - male)
6. QTR-8A reflectance sensor, along with screws
7. Ball caster
8. Extra wires from the kit or lab
9. Chassis
10. Breadboard - medium (400pts)
11. L293D motor driver
12. DC motors (2)

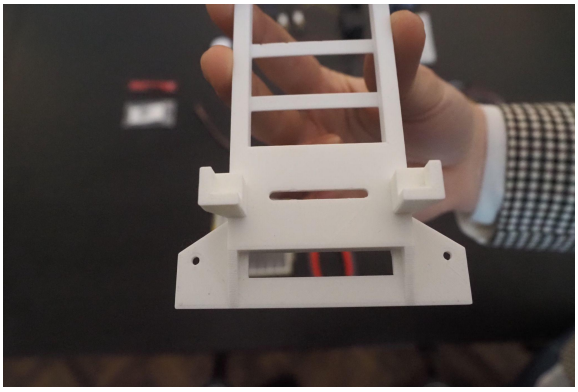
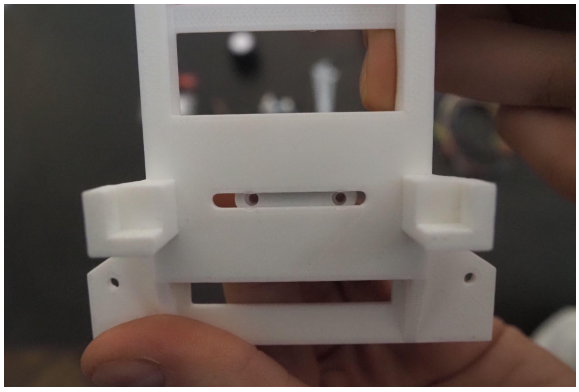
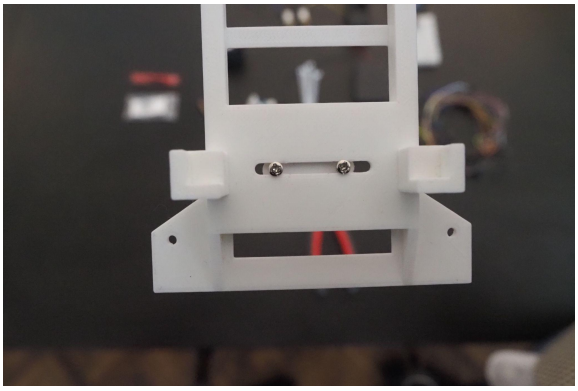
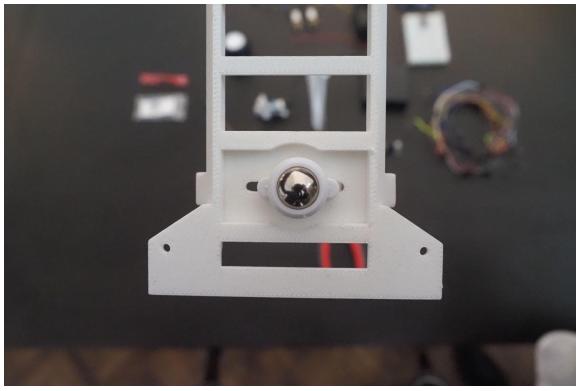
2. Assembly

2.1 Ball Caster (piece no. 7)

This is a tricky component, pay attention!

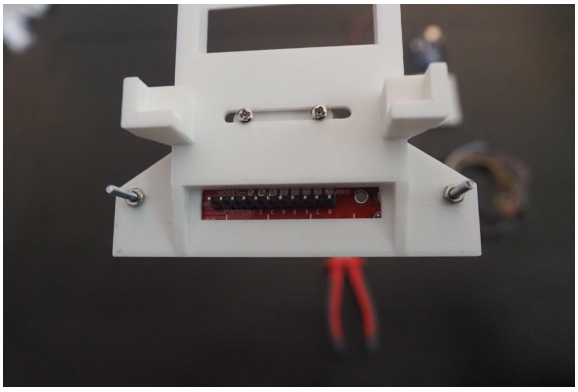
You will need to unscrew the 2 bolts, in order to fix it to the chassis. **Careful, as to not lose anything from inside the ball caster when removing the bolts!**

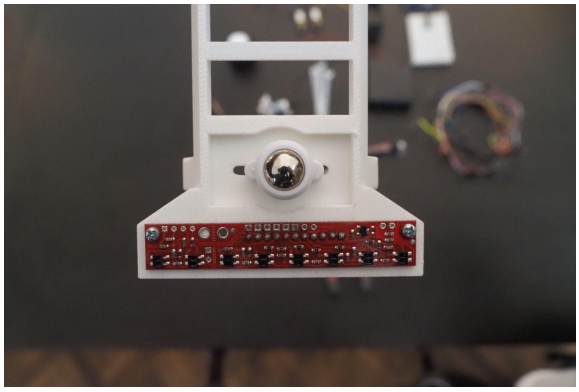





Fix it as in the picture and insert the screws back. Make sure that they are tight enough so that it doesn't wobble, but at the same time not too tight so that the metal ball stops moving due to pressure.

2.2 Line sensor (piece no. 6)

This is quite straightforward: just fix the sensor using 2 bolts and nuts provided. You should have an extra 2 bolts and 2 nuts - these are just in case you lose them, as they are small.
Just be careful to put it in the correct orientation!



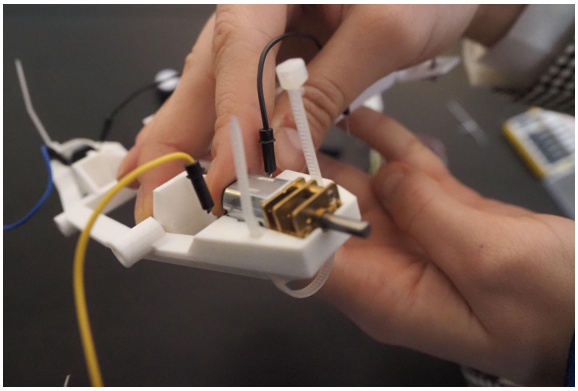


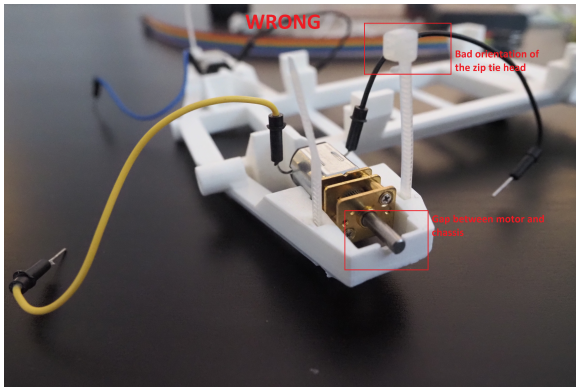
2.3 Motors (piece no. 12)

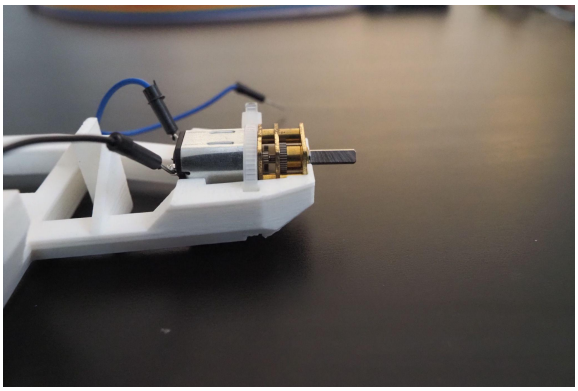
The motors are put on the sides and fixed with zip ties (4 pieces). First of all, make sure you understand how to use the zip ties orientation. Secondly, there are many wrong ways to do this (check the **wrong** pictures to be careful). Third, make sure you tighten them properly, as you don't want it moving or vibrating around.

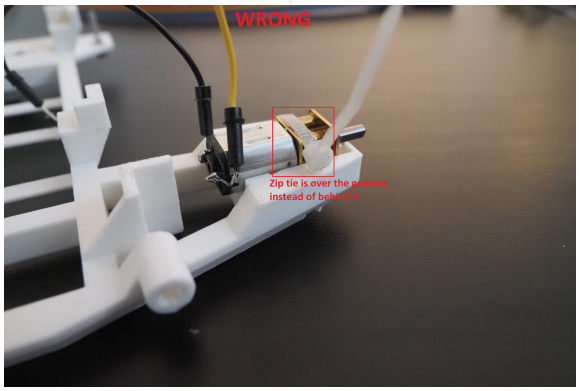
Step 1: place the motor in the socket
Step 2: connect the zip ties through the holes (**attention regarding the orientation of the zip tie, so it closes properly!)**
Step 3: tighten the zip tie correctly (**attention: make sure the motor is placed properly, with no gap and also the zip tie should be after the gearbox, not over it**)
Step 4: repeat for other motor

Correct **Wrong**



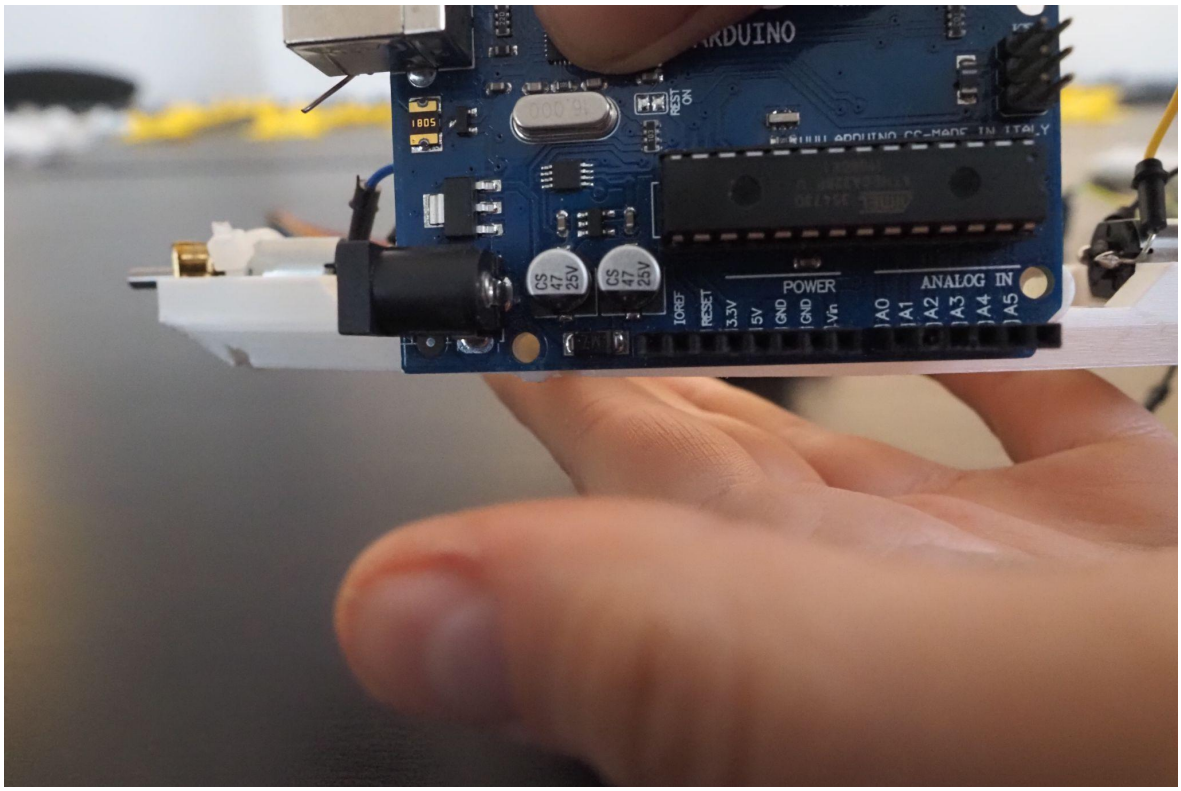




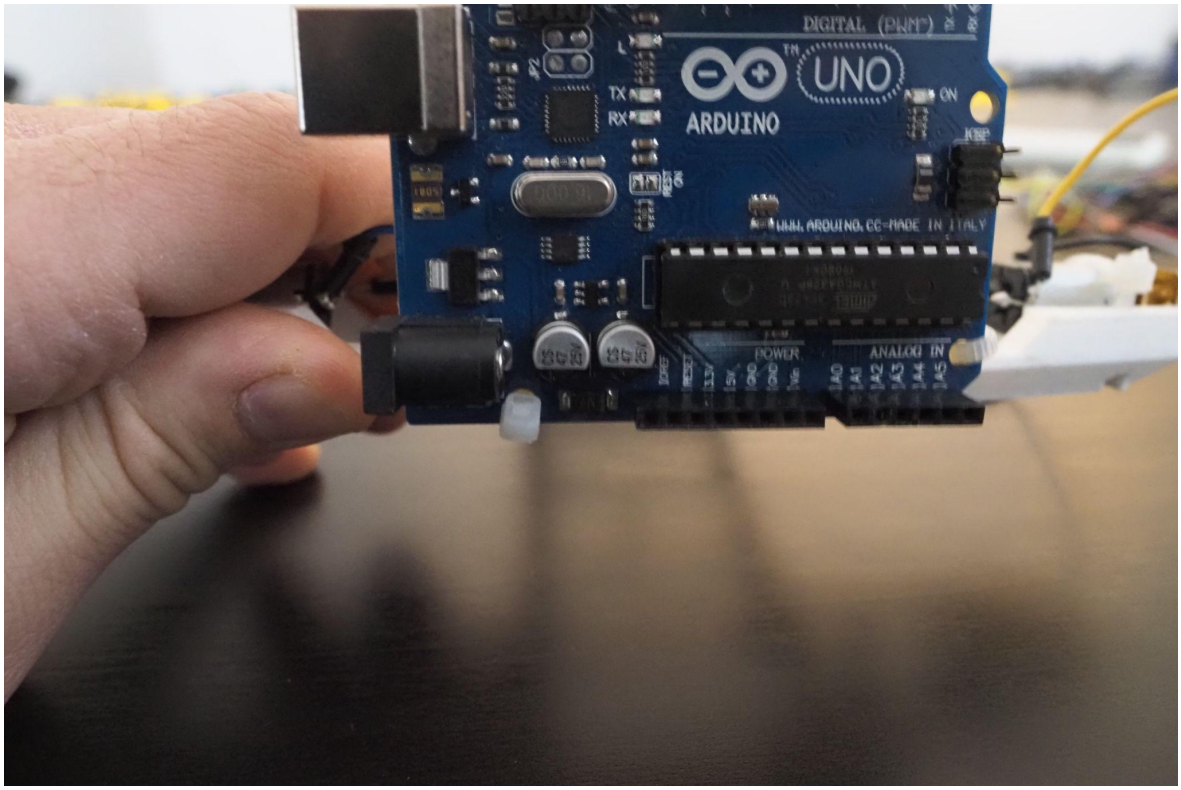


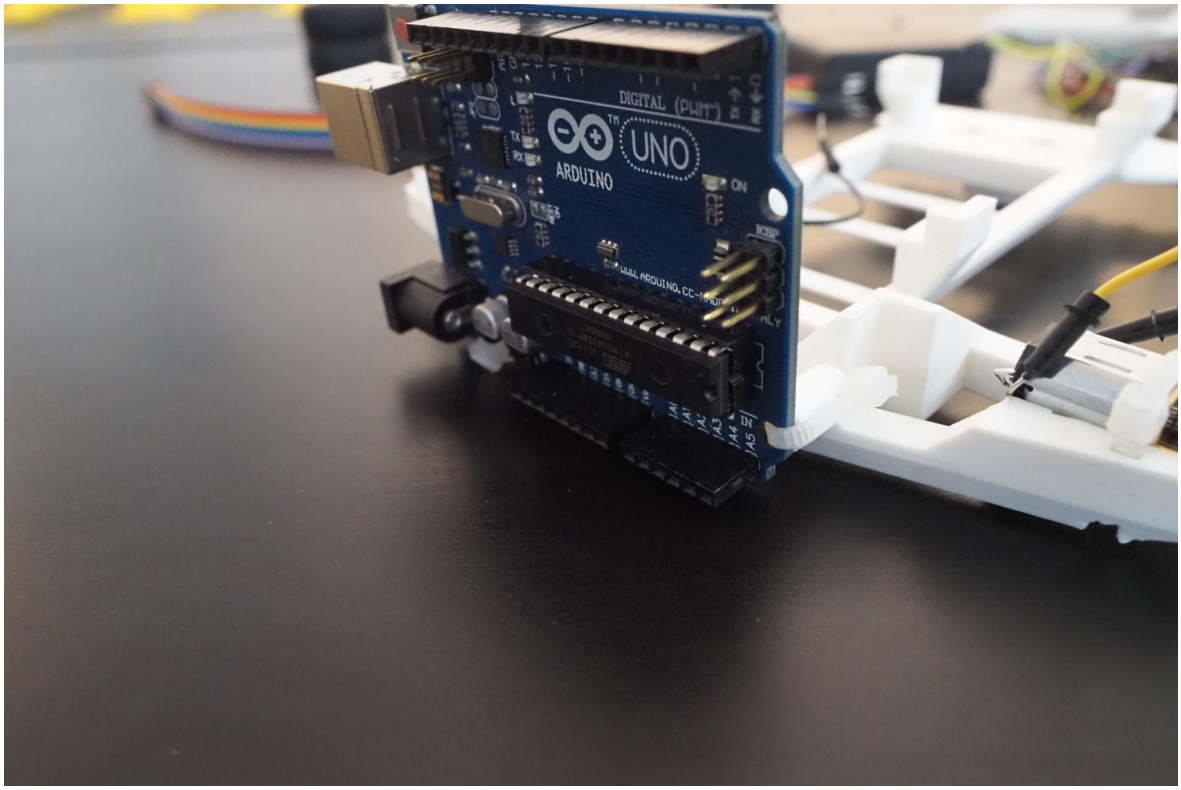
2.4 Arduino Uno (Piece no. 1)

Make sure to match the 2 holes from the arduino and just insert the zip ties. Quite easy.



Afterwards, make sure the zip tie on the left goes under, while the zip tie on the right goes to the side.

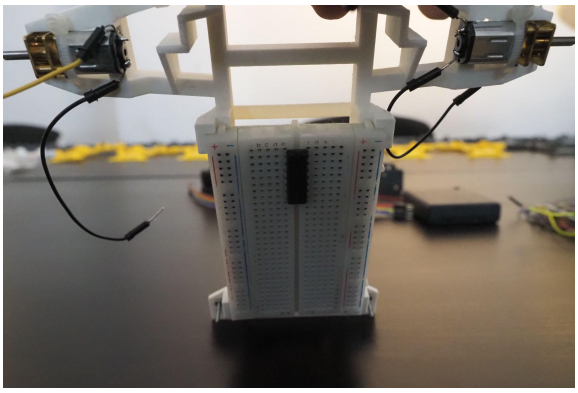

DIGITAL (PWM~)
UNO
ARDUINO
TX
RX
ON
POWER
ANALOG IN


DIGITAL (PWM~)
UNO
ARDUINO
ON

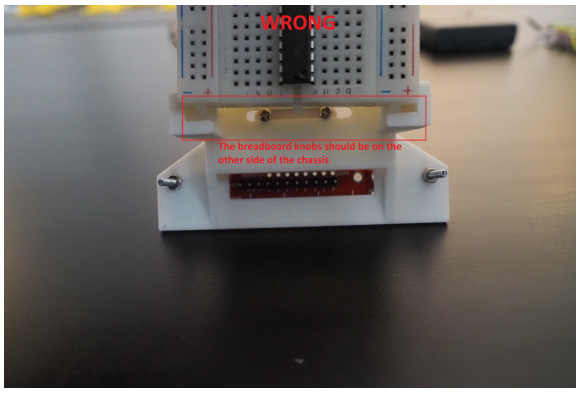
2.5 Breadboard 400pc (Piece no. 10)

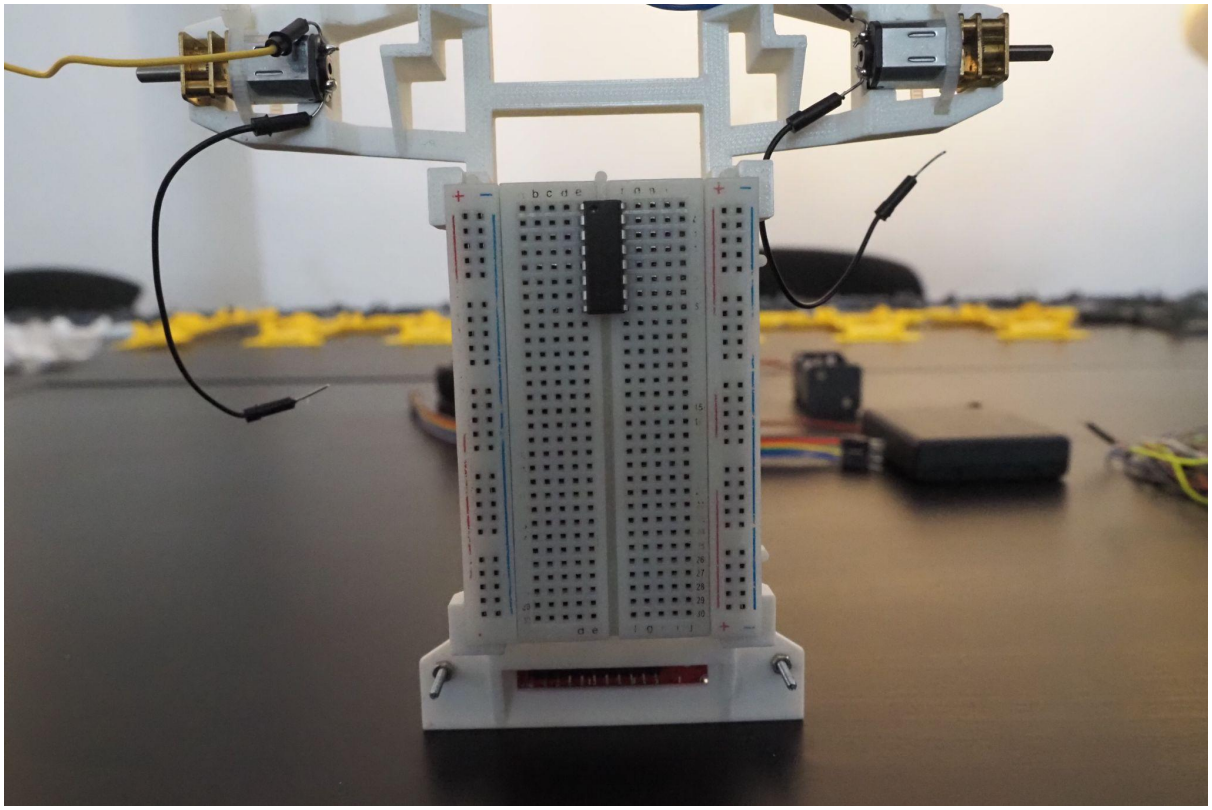
It is held in the slot using pressure. If it doesn't fit, pull the sides of the chassis a bit (that will hold the breadboard) to make room for it. Make sure to look carefully so that the orientation of the breadboard is correct. Parts of it have some knobs - those need to be on the side with the arduino, not the sensor (check picture).

Correct



Wrong





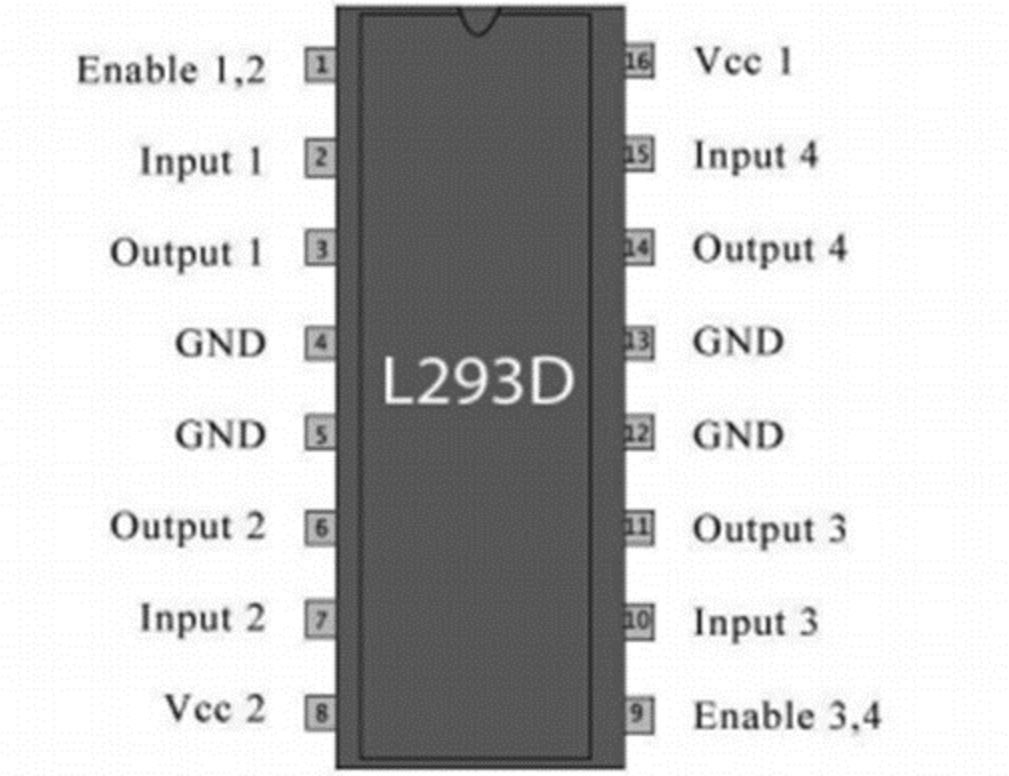
2.6 L293d driver (Piece no. 11)

Should be placed as we did in the laboratory, on the upper part of the breadboard.

2.7 Wire connections

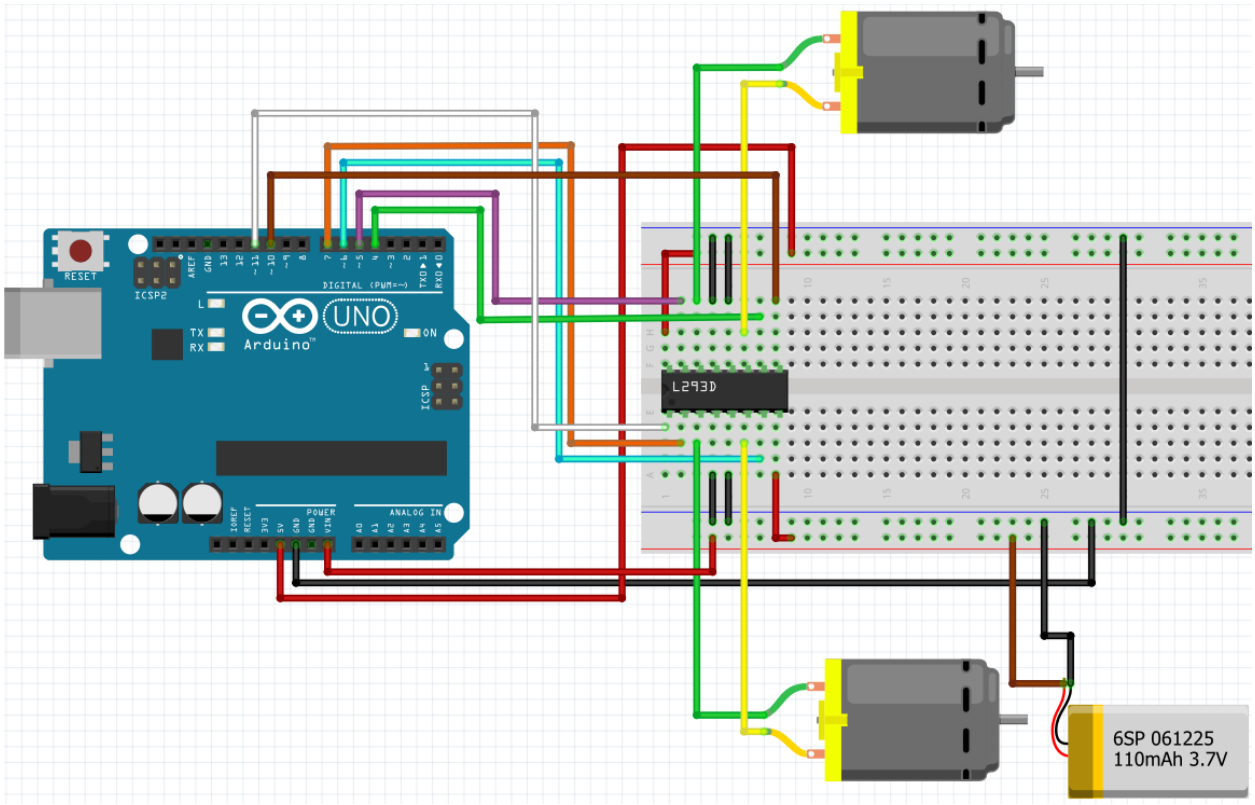
1. Connect the QTR8-A (sensors 2-7) to A0-A5 pins
2. Connect the QTR8-A VIN and GND to the 5V and GND column **(aka the one from Arduino, NOT the one from the battery)**
3. Connect the L293D driver to Arduino and to both motors
4. Connect the battery to VIN (understand the concept, first)

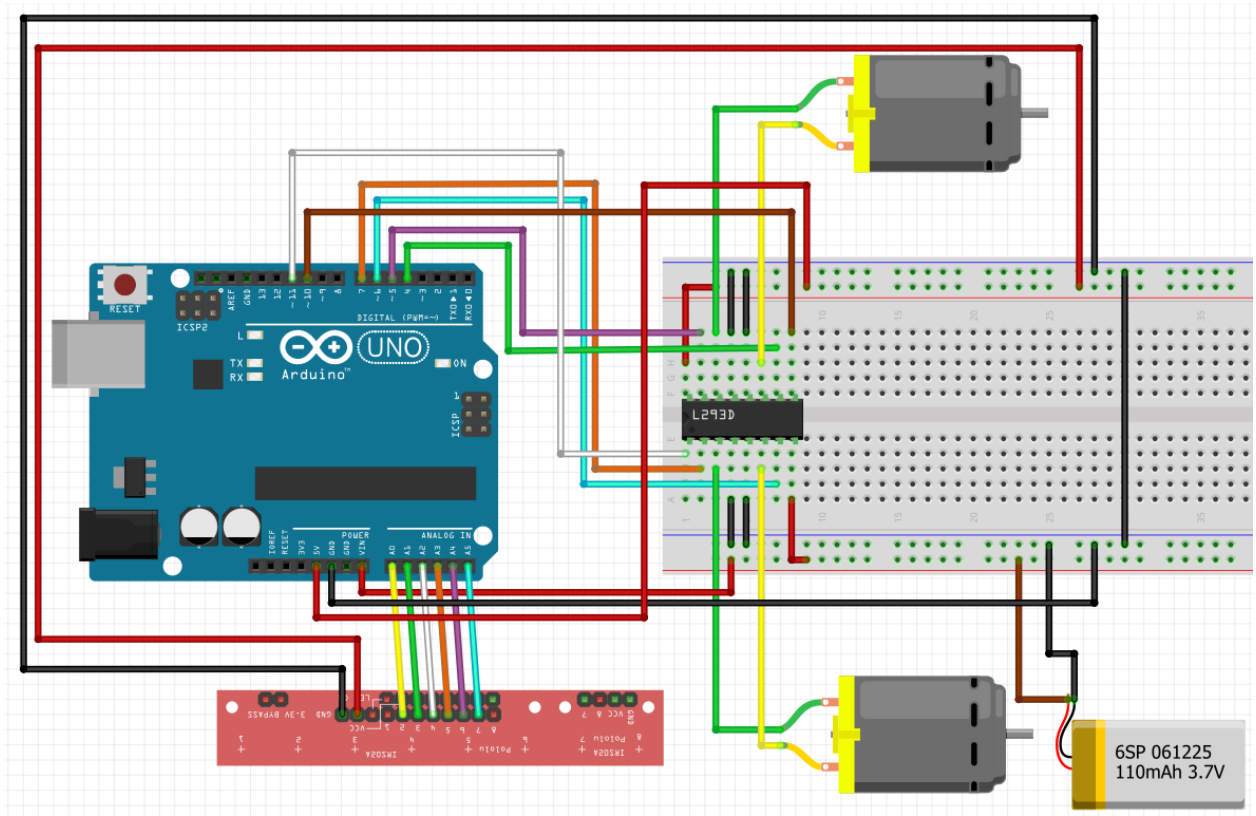
Connections:



Driver Connection table:

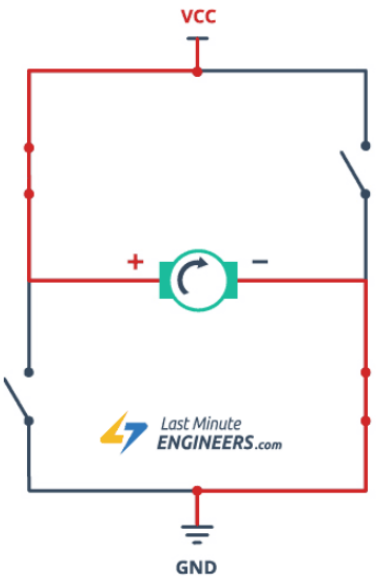
L293D	Connector Pin	Details
Enable 1,2 (1)	Arduino Pin 11	
Input 1 (2)	Arduino Pin 7	
Output 1 (3)	Motor1 Pin 1	
GND (4)	GND	
GND (5)	GND	
Output 2 (6)	Motor1 Pin 2	
Input 2 (7)	Arduino Pin 6	
VCC 2 (8)	Column with battery source (6V+)	
Enable 3,4 (9)	Arduino Pin 10	
Input 3 (10)	Arduino Pin 5	
Output 3 (11)	Motor2 Pin 2	
GND (12)	GND	
GND (13)	GND	
Output4 (14)	Motor2 Pin 1	
Input4 (15)	Arduino Pin 4	
VCC1 (16)	Column with Arduino 5V source	


L293D
6SP 061225
110mAh 3.7V

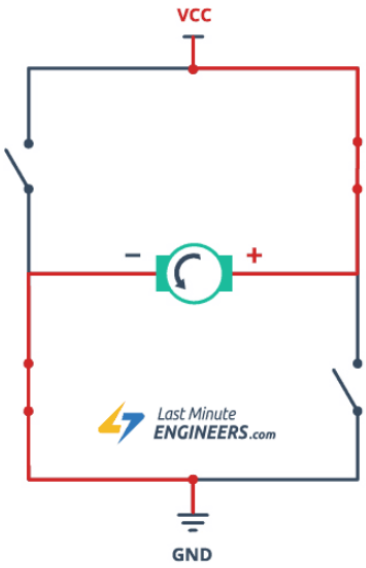

L293D
6SP 061225
110mAh 3.7V

By changing the polarity of the input voltage, we can change the spinning direction of a DC motor. This can be achieved by using an H-bridge circuit, which consists of four switches with the motor in the center, forming an H-like arrangement.

FORWARD (HIGH-LOW) ***BACKWARD (LOW-HIGH)***



Working of H-Bridge



Working of H-Bridge

Closing two specific switches at a time reverses the polarity of the voltage applied to the motor, which results in changing the spinning direction of the motor.

The input pins are the direction control pins.

IN1	IN2	Spinning Direction
Low(0)	Low(0)	Motor OFF
High(1)	Low(0)	Forward
Low(0)	High(1)	Backward
High(1)	High(1)	Motor OFF

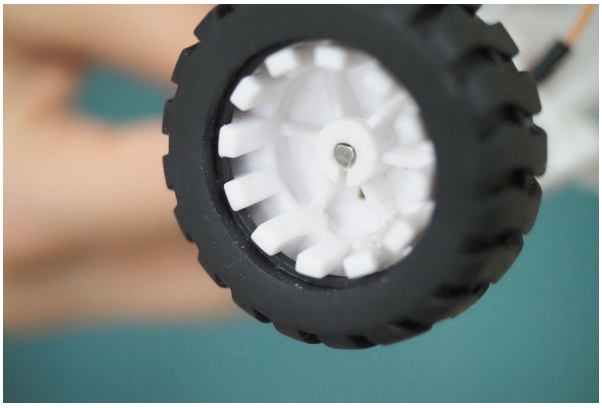
L293D = a dual-channel H-Bridge motor driver capable of driving a pair of DC motors individually

Source: *https://lastminuteengineers.com/l293d-dc-motor-arduino-tutorial/*

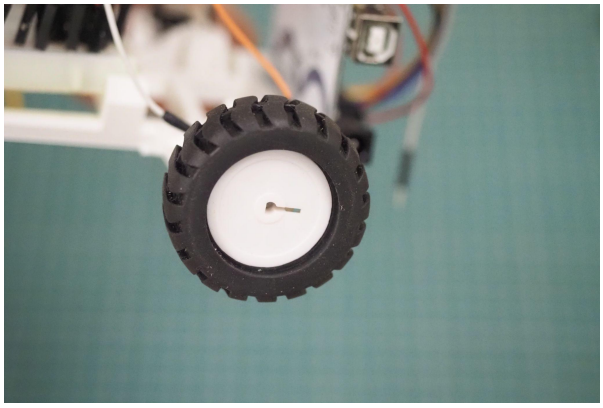
2.8 Wheels (Piece no. 12)

For the wheels, you just have to match the shape of the motor axis with the shape of the hole. Make sure you put the wheels in the correct orientation.

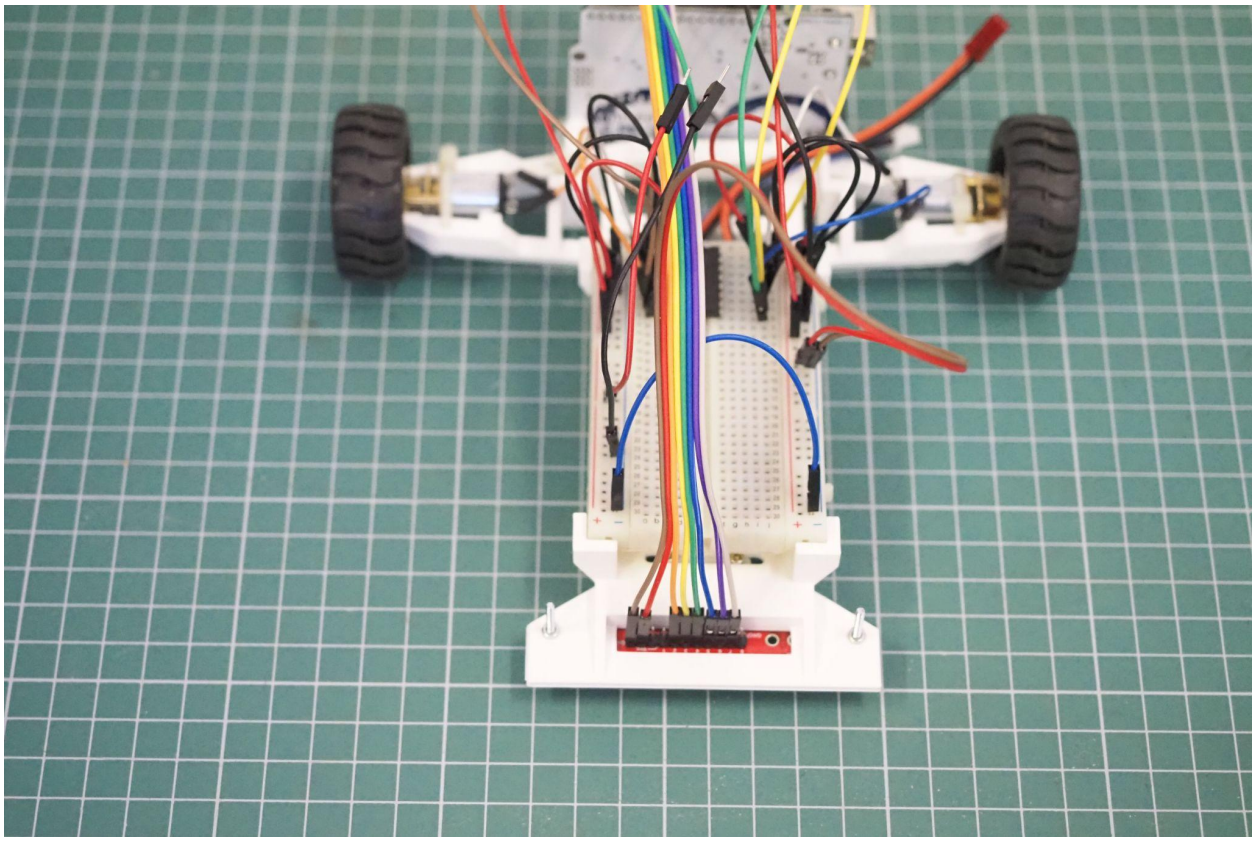
Correct



Wrong



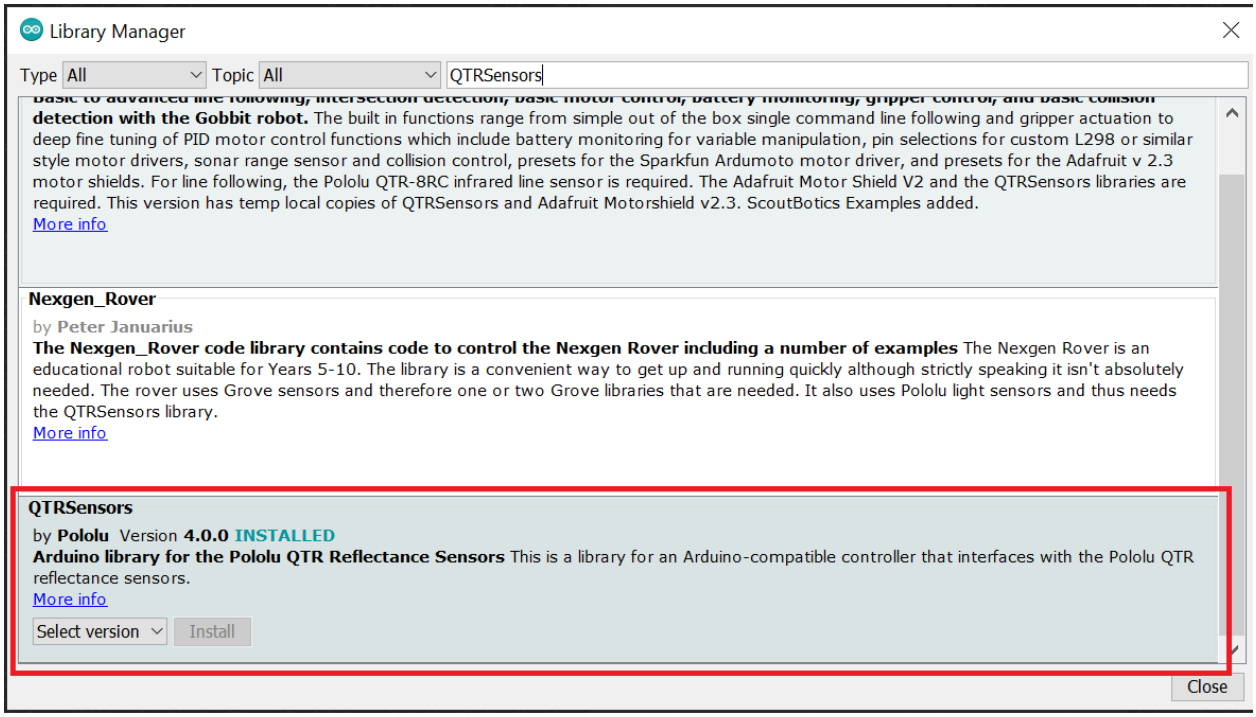
Good luck!



3. Code

3.1 Test the sensor array

3.1.1 Install the Pololu QTRSensors library



3.1.2 Print RAW sensor values

1. Go to File -> Examples -> QTRSensors -> QTRARawValuesExample
2. Upload the code
3. Move the sensor over a black line slowly and look at the changing values
4. Infer the

3.1.3 Print calibrated sensor values

1. Go to File -> Examples -> QTRSensors -> QTRAExample
2. Upload the code
3. Once uploaded, move the sensor slowly over the black line such that all small sensor read both black and white values while calibrating
4. Once done, it will print the read values, but this time calibrated. Notice the difference.

3.2 Function for controlling both motors.

Test it by running setMotorsSpeed(255, 255)

```
const int m11Pin = 7;
const int m12Pin = 6;
const int m21Pin = 5;
const int m22Pin = 4;
const int m1Enable = 11;
const int m2Enable = 10;

int m1Speed = 0;
int m2Speed = 0;

void setup() {
 // put your setup code here, to run once:
 pinMode(m11Pin, OUTPUT);
 pinMode(m12Pin, OUTPUT);
 pinMode(m21Pin, OUTPUT);
 pinMode(m22Pin, OUTPUT);
 pinMode(m1Enable, OUTPUT);
 pinMode(m2Enable, OUTPUT);
}

void loop() {
 // put your main code here, to run repeatedly:
 setMotorSpeed(255, 255);\

}


// each argument takes values between -255 and 255. The negative values represent the motor speed
in reverse.
void setMotorSpeed(int motor1Speed, int motor2Speed) {
 // remove comment if any of the motors are going in reverse
 //  motor1Speed = -motor1Speed;
 //  motor2Speed = -motor2Speed;
 if (motor1Speed == 0) {
  digitalWrite(m11Pin, LOW);
  digitalWrite(m12Pin, LOW);
  analogWrite(m1Enable, motor1Speed);
```

```
  }
  else {
   if (motor1Speed > 0) {
    digitalWrite(m11Pin, HIGH);
    digitalWrite(m12Pin, LOW);
    analogWrite(m1Enable, motor1Speed);
   }
   if (motor1Speed < 0) {
    digitalWrite(m11Pin, LOW);
    digitalWrite(m12Pin, HIGH);
    analogWrite(m1Enable, -motor1Speed);
   }
  }
  if (motor2Speed == 0) {
   digitalWrite(m21Pin, LOW);
   digitalWrite(m22Pin, LOW);
   analogWrite(m2Enable, motor2Speed);
  }
  else {
   if (motor2Speed > 0) {
    digitalWrite(m21Pin, HIGH);
    digitalWrite(m22Pin, LOW);
    analogWrite(m2Enable, motor2Speed);
   }
   if (motor2Speed < 0) {
    digitalWrite(m21Pin, LOW);
    digitalWrite(m22Pin, HIGH);
    analogWrite(m2Enable, -motor2Speed);
   }
  }
}
```

3.3 Minimal line following code, implementing a basic P

```
#include <QTRSensors.h>
const int m11Pin = 7;
const int m12Pin = 6;
const int m21Pin = 5;
const int m22Pin = 4;
const int m1Enable = 11;
const int m2Enable = 10;

int m1Speed = 0;
int m2Speed = 0;

// increase kp's value and see what happens
float kp = 1;
float ki = 0;
float kd = 0;

int p = 1;
int i = 0;
int d = 0;

int error = 0;
int lastError = 0;

const int maxSpeed = 255;
const int minSpeed = -255;

const int baseSpeed = 255;

QTRSensors qtr;

const int sensorCount = 6;
int sensorValues[sensorCount];
int sensors[sensorCount] = {0, 0, 0, 0, 0, 0};

void setup() {

  // pinMode setup
  pinMode(m11Pin, OUTPUT);
  pinMode(m12Pin, OUTPUT);
```

```
  pinMode(m21Pin, OUTPUT);
  pinMode(m22Pin, OUTPUT);
  pinMode(m1Enable, OUTPUT);
  pinMode(m2Enable, OUTPUT);

  qtr.setTypeAnalog();
  qtr.setSensorPins((const uint8_t[]){A0, A1, A2, A3, A4, A5}, sensorCount);

  delay(500);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH); // turn on Arduino's LED to indicate we are in calibration mode

  // calibrate the sensor. For maximum grade the line follower should do the movement itself, without
human interaction.
  for (uint16_t i = 0; i < 400; i++)
  {
    qtr.calibrate();
    // do motor movement here, with millis() as to not ruin calibration)
  }
  digitalWrite(LED_BUILTIN, LOW);

  Serial.begin(9600);

}

void loop() {
  // inefficient code, written in loop. You must create separate functions
  int error = map(qtr.readLineBlack(sensorValues), 0, 5000, -50, 50);

  p = error;
  i = i + error;
  d = error - lastError;

  int motorSpeed = kp * p + ki * i + kd * d; // = error in this case

  m1Speed = baseSpeed;
  m2Speed = baseSpeed;

  // a bit counter intuitive because of the signs
  // basically in the first if, you substract the error from m1Speed (you add the negative)
  // in the 2nd if you add the error to m2Speed (you substract the negative)
  // it's just the way the values of the sensors and/or motors lined up
```

```
  if (error < 0) {
    m1Speed += motorSpeed;
  }
  else if (error > 0) {
    m2Speed -= motorSpeed;
  }
  // make sure it doesn't go past limits. You can use -255 instead of 0 if calibrated programmed
properly.
  // making sure we don't go out of bounds
  // maybe the lower bound should be negative, instead of 0? This of what happens when making a
steep turn
  m1Speed = constrain(m1Speed, 0, maxSpeed);
  m2Speed = constrain(m2Speed, 0, maxSpeed);


  setMotorSpeed(m1Speed, m2Speed);


//  DEBUGGING
//  Serial.print("Error: ");
//  Serial.println(error);
//  Serial.print("M1 speed: ");
//  Serial.println(m1Speed);
//
//  Serial.print("M2 speed: ");
//  Serial.println(m2Speed);
//
//  delay(250);
}


// calculate PID value based on error, kp, kd, ki, p, i and d.
void pidControl(float kp, float ki, float kd) {
// TODO
}


// each arguments takes values between -255 and 255. The negative values represent the motor speed
in reverse.
void setMotorSpeed(int motor1Speed, int motor2Speed) {
  // remove comment if any of the motors are going in reverse
  //  motor1Speed = -motor1Speed;
```

```
  //  motor2Speed = -motor2Speed;
  if (motor1Speed == 0) {
    digitalWrite(m11Pin, LOW);
    digitalWrite(m12Pin, LOW);
    analogWrite(m1Enable, motor1Speed);
  }
  else {
    if (motor1Speed > 0) {
      digitalWrite(m11Pin, HIGH);
      digitalWrite(m12Pin, LOW);
      analogWrite(m1Enable, motor1Speed);
    }
    if (motor1Speed < 0) {
      digitalWrite(m11Pin, LOW);
      digitalWrite(m12Pin, HIGH);
      analogWrite(m1Enable, -motor1Speed);
    }
  }
  if (motor2Speed == 0) {
    digitalWrite(m21Pin, LOW);
    digitalWrite(m22Pin, LOW);
    analogWrite(m2Enable, motor2Speed);
  }
  else {
    if (motor2Speed > 0) {
      digitalWrite(m21Pin, HIGH);
      digitalWrite(m22Pin, LOW);
      analogWrite(m2Enable, motor2Speed);
    }
    if (motor2Speed < 0) {
      digitalWrite(m21Pin, LOW);
      digitalWrite(m22Pin, HIGH);
      analogWrite(m2Enable, -motor2Speed);
    }
  }
}
```

4. Grading

1. Implement calibration with automatic motor movement
2. Follow a curved line (not exaggerated)
3. Finish the line follower track
4. Finish the line follower track in a fast manner with visible control (implementing P, D and maybe I)