IGEN 230 – Line Following Robot Description Sheet

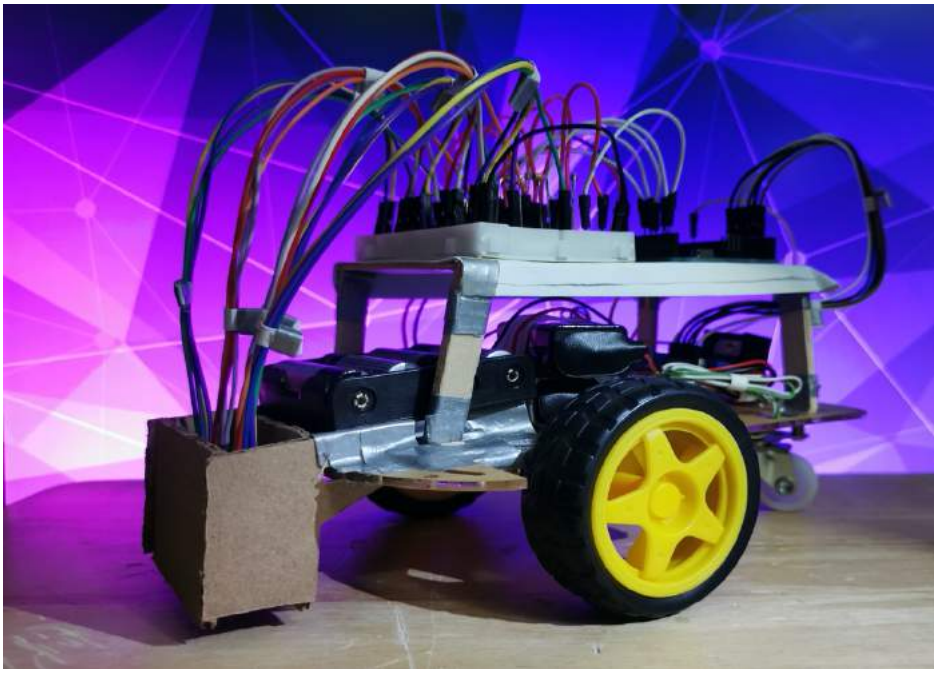
Design ideas and justifications

For the line-follower robot design, I was faced with a few unique design problems.
To start, we were limited by the allowed resources, having mainly cardboard to work with. All parts were held together by duct tape, with no other adhesive or support structures.

In all there were 2 main problems to address - available space / cable management, and 2 main components of concern - the sensor bracket as well as the power supply.

Available Space

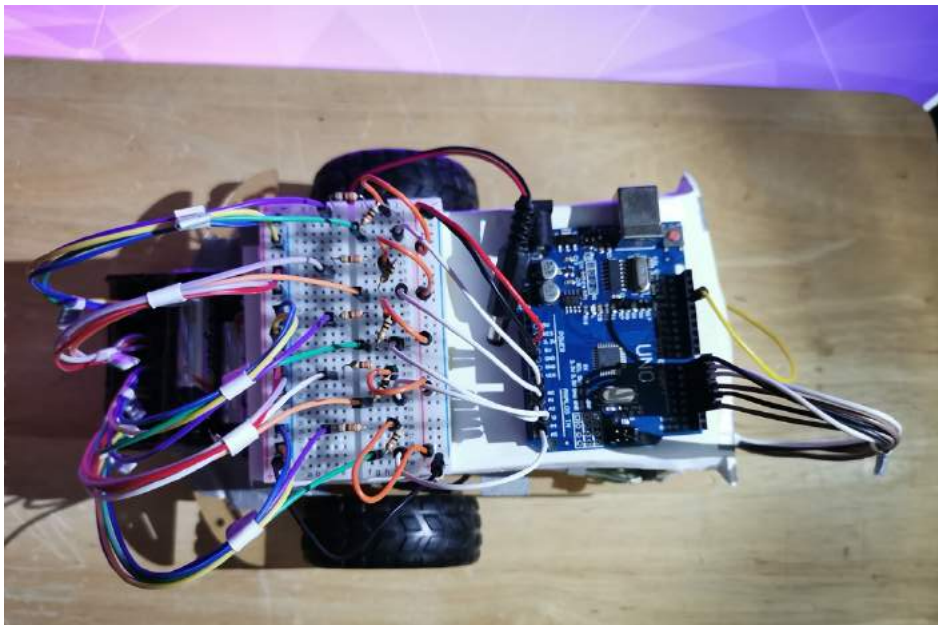
Initially while laying out all the necessary circuitry and components, a glaring issue arose while planning how to effectively use the given space. Clearly, the base plate did not have enough real estate to lay out the Arduino, breadboard, H-Bridge and so forth, which led to the creation of a loft above the main body.



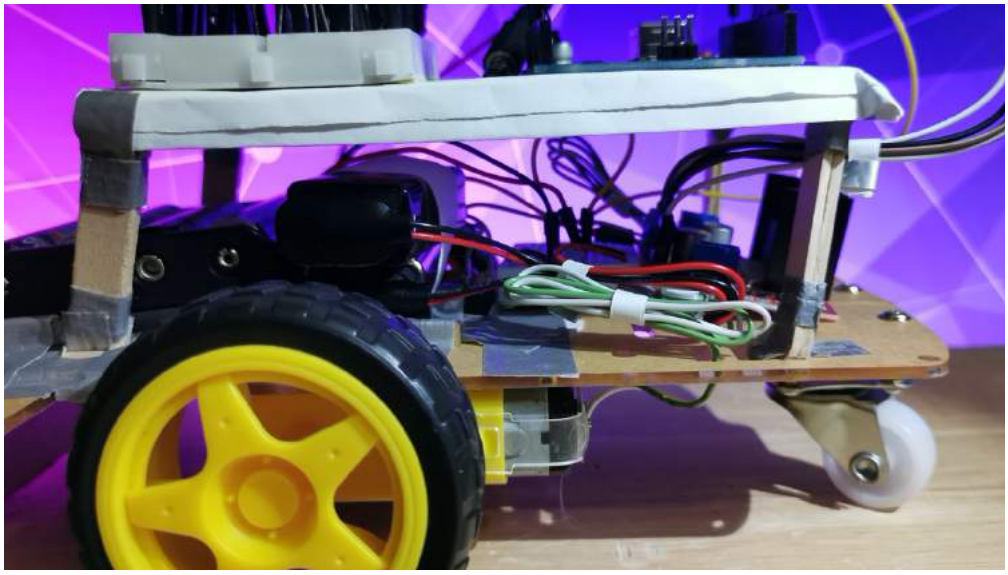
The platform was made of a sheet of printer paper and supported by columns. On the top I kept the Arduino and Breadboard, which were the components needing the most attention in rewiring and connecting the motors, power, and sensors. Under the platform was the H-Bridge, battery housing and breadboard for the power supplies (mentioned below)

Wire Management

The other noteworthy problem was that of wire management. This was paramount when troubleshooting issues, and by keeping the circuitry untangled, it made it clear at a glance which parts were necessarily connected to one another.



As shown in the photo above, all wires are bundled using paper and tape to keep groups of wires separate, and to shorten unnecessary long wires. Starting from the rear (right), the brown, black and white wires control the H-Bridge, while the wires in the middle are for the sensor circuit. At the head of the robot, wires are grouped by their individual sensor, and again are kept separate by paper/tape wrapping.

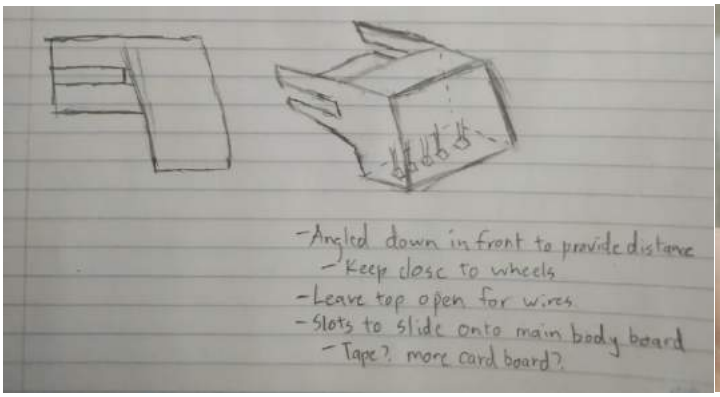


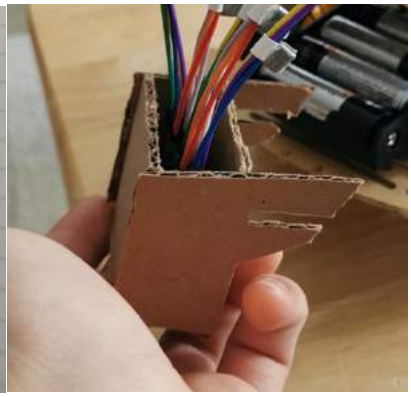
Looking further to the side, we see that the long 9V battery wires, as well as the motor connections to the H-Bridge are wrapped up, keeping them short and avoiding entanglement. To the left, a small bracket was made to house two 9V batteries, as well as the 1.5Vs used for powering the Arduino board.

The rest of the robot was comprised of two main components, a sensor bracket, and a breadboard for the power sources.

Sensor Bracket

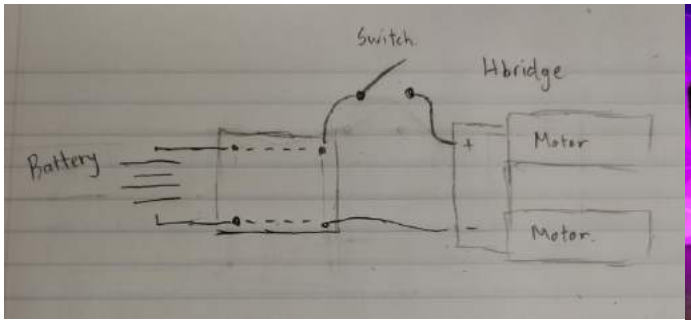
The sensor bracket was designed at a slight angle and slotted to the rear of the base plate. This would give the robot ample time to see the track ahead of it, while also being close enough to the pivot point (wheels) to rotate cleanly around sharper corners. The bracket was designed with two cuts which would allow for it to slot onto the plate snugly, requiring minimum resources to maintain rigidity.

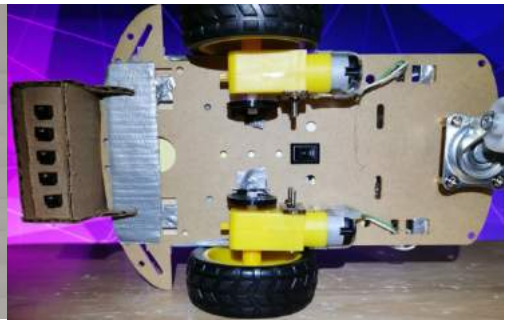




Power Supply

While I believe running the motors using two 9V batteries were overkill, the advantage was allowing me to have greater control over the speed the motors operated at - being able to go as low as 65 (from 100), while the lowest speed on one battery was ~ 90.





Sensor bracket & power switch

Shown above is the breadboard wiring for the power supply to the H-Bridge. On the left, are the two 9V batteries connected in series, running into a small breadboard. This was necessary to connect the small switch included in the IGEN package.

Note: While I don't believe the switch was required in the design, it made turning the robot on/off much easier which is why I went through the trouble to do so.

My Code

Bot_Project

```
//Initial Setup
  //Motors
int RIn1 = 2;
int RIn2 = 3;
int LIn1 = 4;
int LIn2 = 7;
int RSpd = 6;
int LSpd = 5;
  //QRD1114 Sensors
const int SR2 = A0;
const int SR1 = A1;
const int SMid = A2;
const int SL1 = A3;
const int SL2 = A4;

  //Additional Constants
//Constant used for adjusting sensor output
int Sensitivity = 750.0;
//The value which establishes on/off the line
int LineSense = 4.0;
//Motor speed
int Spd = 75;
// Setup code here, to run once:
void setup() {
  pinMode(RIn1, OUTPUT);
  pinMode(RIn2, OUTPUT);
  pinMode(RSpd, OUTPUT);
  pinMode(LIn1, OUTPUT);
  pinMode(LIn2, OUTPUT);
  pinMode(LSpd, OUTPUT);

  Serial.begin(9600);
  pinMode(SR2, INPUT);
  pinMode(SR1, INPUT);
  pinMode(SMid, INPUT);
  pinMode(SL1, INPUT);
  pinMode(SL2, INPUT);
}
```

Bot_Project

```
//Define
void RMForward() {
  digitalWrite(RIn1, LOW);
  digitalWrite(RIn2, HIGH);
  analogWrite(RSpd,Spd);
}

void RMReverse() {
  digitalWrite(RIn1, HIGH);
  digitalWrite(RIn2, LOW);
  analogWrite(RSpd,Spd);
}

void LMForward() {
  digitalWrite(LIn1, LOW);
  digitalWrite(LIn2, HIGH);
  analogWrite(LSpd,Spd);
}

void LMReverse() {
  digitalWrite(LIn1, HIGH);
  digitalWrite(LIn2, LOW);
  analogWrite(LSpd,Spd);
}

void RMStop() {
  digitalWrite(RIn1, LOW);
  digitalWrite(RIn2, LOW);
}

void LMStop() {
  digitalWrite(LIn1, LOW);
  digitalWrite(LIn2, LOW);
}
```

Variables:

Sensitivity: The value to divide the voltage reading by to scale values to ~ 1
LineSense: The value which dictates whether the sensors are on/off the line
Spd: Speed of the motor

In the header, all the pins, inputs and outputs are defined, as well as the declaration of global variables to be used in tweaking the sensor outputs and motor speed. Here, functions are also written to be called on later in the script.

Bot_Project §

```
// Main code here, to run repeatedly:
void loop() {
  //Read and print each sensor
  int lineR2 = analogRead(SR2);
  int lineR1 = analogRead(SR1);
  int lineMid = analogRead(SMid);
  int lineL1 = analogRead(SL1);
  int lineL2 = analogRead(SL2);

  float lineR2V = (float)lineR2 * 5.0 / Sensitivity;
  float lineR1V = (float)lineR1 * 5.0 / Sensitivity;
  float lineMidV = (float)lineMid * 5.0 / Sensitivity;
  float lineL1V = (float)lineL1 * 5.0 / Sensitivity;
  float lineL2V = (float)lineL2 * 5.0 / Sensitivity;
```

Here, we see the global variables being used to adjust the sensitivity of the sensors.

In terms of logic, the robot follows a basic pattern beginning with scanning what is ahead.

-If it sees it is in a blank space, it will move forward until a line is found.

-If it sees black to its front, far left and right, it must be at a crossroad, and moves forward

-If it sees black beginning to appear to the left/right, it will turn in that direction until clear

-If all other conditions are not met, the robot is clear to move forward

```
  //Start creating If statements for conditions
    //If in the void, move until line is found
    if ((lineL2V < LineSense) && (lineL1V < LineSense) && (lineMidV < LineSense)
    && (lineR1V < LineSense) && (lineR2V < LineSense)) {
      Serial.print("No Line Found! \r");
      LMForward();
      RMForward();
      delay(50);
    }
    //If at a crossroad
    else if ((lineL2V > LineSense) && (lineR2V > LineSense)) {
      Serial.print("At a Junction \r");
      RMForward();
      LMForward();
      delay(50);
    }
    //Turn left condition
    else if ((lineL2V > LineSense) || ((lineL1V > LineSense) && (lineL2V > LineSense))) {
      Serial.print("Turning Left \r");
      RMForward();
      LMReverse();
      delay(60);
    }
    //Turn right condition
    else if ((lineR2V > LineSense) || ((lineR1V > LineSense) && (lineR2V > LineSense))) {
      Serial.print("Turning Right \r");
      RMReverse();
      LMForward();
      delay(60);
    }
    //Else move forward
    else {
      Serial.print("Moving Forward \r");
      RMForward();
      LMForward();
      delay(45);
    }
  RMStop();
  LMStop();
}
```