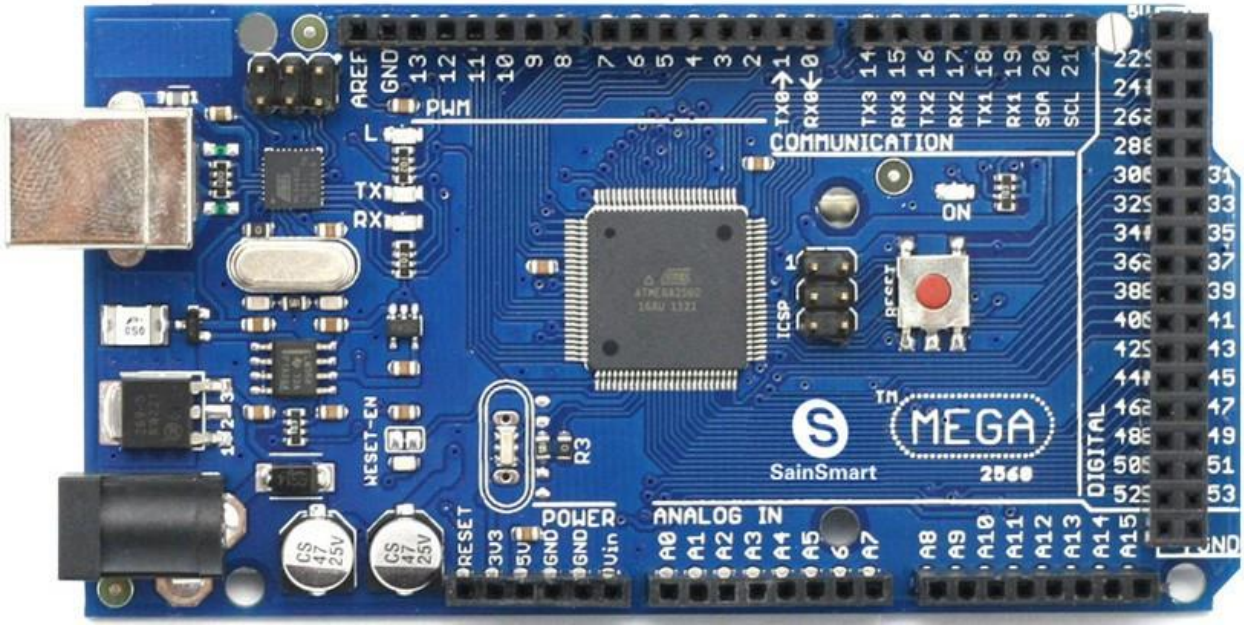
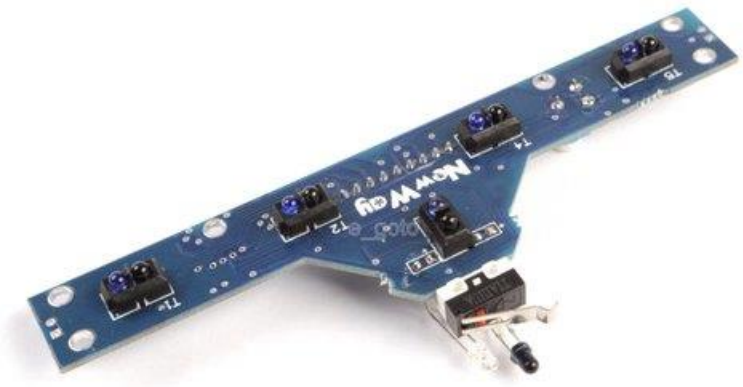
I.Các thiết bị

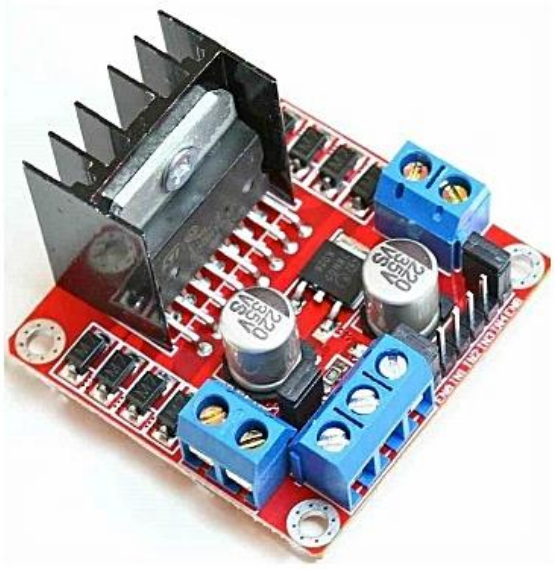
Board mạch xử lí : 1 Adruino Mega 2560 R3



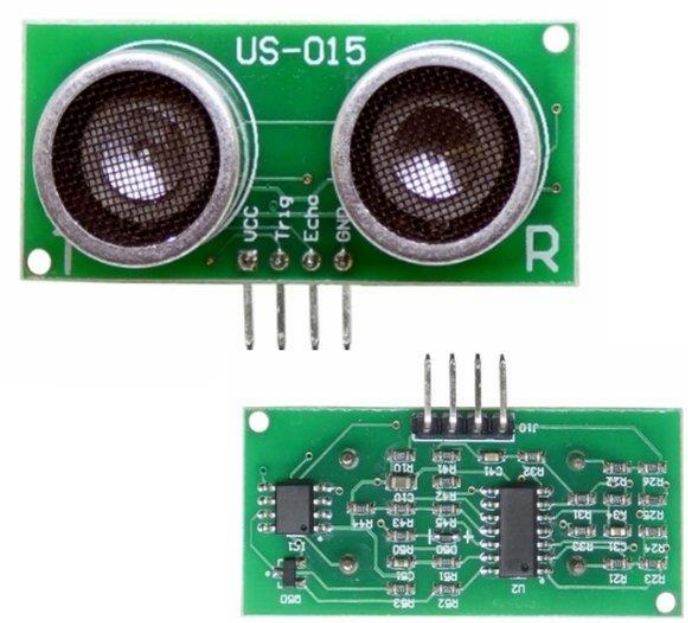
2 thanh dò line ghép nối tiếp



Board mạch điều khiển động cơ : mạch cầu H L298N



Board nhận biết vật cản:



II. Các thông số có thể chỉnh định

1.Kp, Ki , Kd , : bộ số cho Bộ điều khiển hồi tiếp PID (các thông số ở đây được tinh chỉnh khi cắm pin 12V lúc ổn định). Các thông số trong bài là do kiểm nghiệm để phù hợp với động cơ của xe. Người dùng có thể tự chỉnh tăng giảm tùy ý.

Kp: làm cho xe chạy bám tốt nếu tăng vừa phải, lớn quá làm xe chạy lệch.

Ki: càng lớn làm xe mất ổn định.

Kd: càng lớn làm xe bám mượt nhưng sẽ bám không nhanh, xe chạy nhanh có thể lệch khỏi line.

2.speed_val: tốc độ cho động cơ tính theo đơn vị vòng/phút (tối đa 200 vòng/ phút). (có thể tăng giảm tùy vào địa hình chạy, chạy chậm thì bám cua tốt hơn).

Không nên để max tốc độ, hợp lí nhất là 80-120.

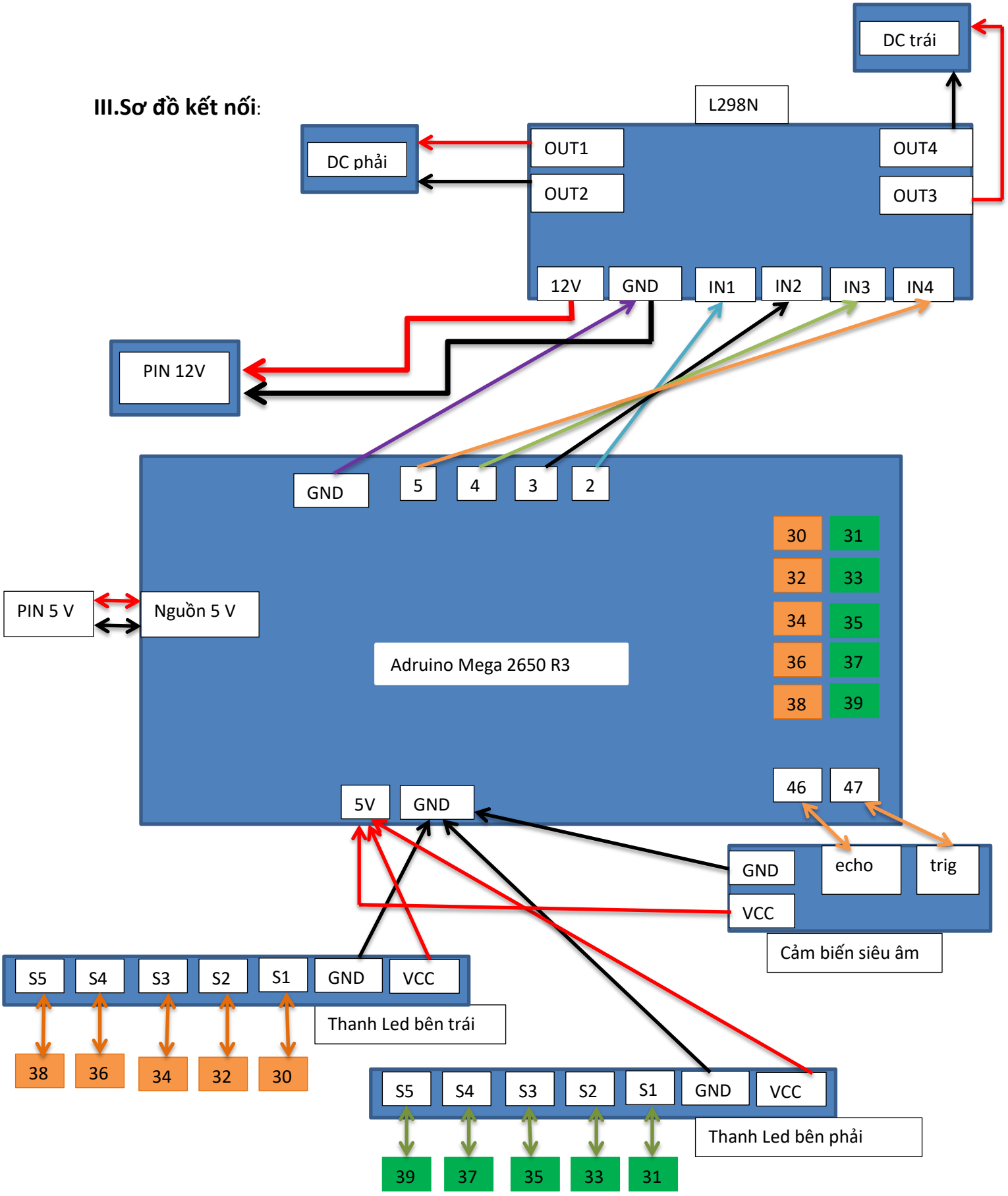
3.dis: thiết đặt khoảng cách để xe tạm dừng khi thấy vật cản tính bằng đơn vị cm.

4. wdt_enable(WDTO_4S)

Hàm thiết đặt WatchDog Timer, khi vi xử lý bị treo sẽ reset sau 1 khoảng thời gian. Khoảng thời gian đó có thể tự thiết đặt là :

THRESHOLD VALUE	CONSTANT NAME	SUPPORTED ON
15 ms	WDTO_15MS	ATMega 8, 168, 328, 1280, 2560
30 ms	WDTO_30MS	ATMega 8, 168, 328, 1280, 2560
60 ms	WDTO_60MS	ATMega 8, 168, 328, 1280, 2560
120 ms	WDTO_120MS	ATMega 8, 168, 328, 1280, 2560
250 ms	WDTO_250MS	ATMega 8, 168, 328, 1280, 2560
500 ms	WDTO_500MS	ATMega 8, 168, 328, 1280, 2560
1 s	WDTO_1S	ATMega 8, 168, 328, 1280, 2560
2 s	WDTO_2S	ATMega 8, 168, 328, 1280, 2560
4 s	WDTO_4S	ATMega 168, 328, 1280, 2560
8 s	WDTO_8S	ATMega 168, 328, 1280, 2560

III.Sơ đồ kết nối:



IV. Nguyên lí hoạt động

Xử lí dò line : 2 thanh dò line có nhiệm vụ đọc line liên tục, màu đen cho mức 0 (đèn báo line sáng) , màu trắng cho mức 1 (đèn báo line tắt) . Tín hiệu này được gửi về Vi xử lí Adruino Mega 2560 R3 để xử lí tính toán.

Xử lí điều chỉnh tốc độ: dùng dữ liệu thanh dò line gửi về, vi xử lí sẽ tính toán bằng giải thuật PID và xuất ra độ rộng xung phù hợp cho module L298 để điều khiển tốc độ 2 động cơ trái và phải.

Cách dùng Module L298N:

+ Chân IN1 và IN3 nhận xung từ Adruino Mega R3: với độ rộng đã điều chỉnh, với mỗi độ rộng khác nhau, áp ra ở các cặp chân OUT sẽ khác nhau

+ Chân IN2 và IN4 nhận tín hiệu số (digital) : mức LOW và HIGH sẽ cho chiều quay động cơ ngược nhau .

Nguồn :

+ Nguồn 12V : có thể là Pin lipo 12V hoặc Pin khác để xe chạy ổn định.

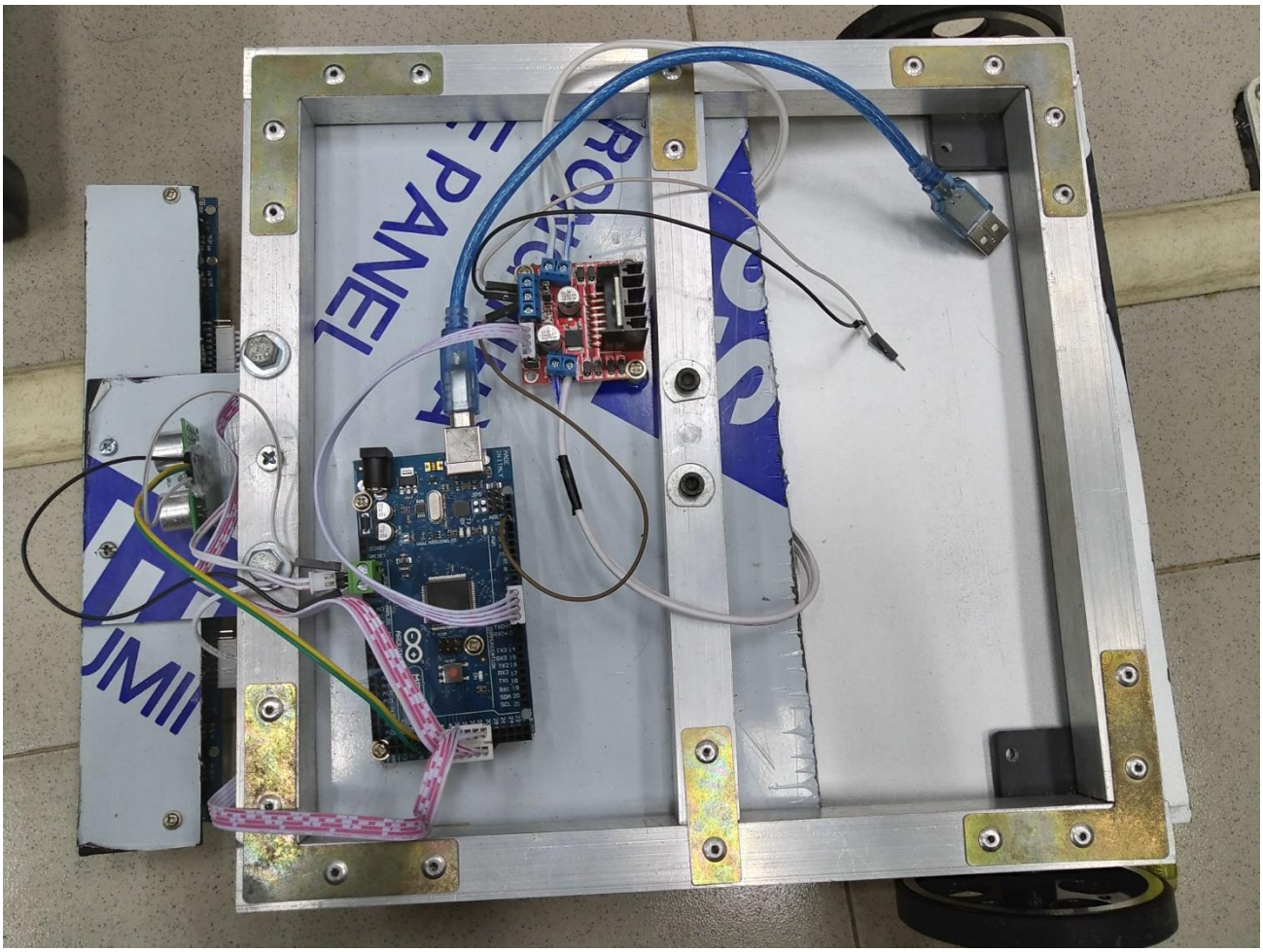
+ Nguồn 5V : có thể là in 5V hoặc Cục sạc dự phòng cho điện thoại để cấp qua cổng nạp – rất tiện lợi.

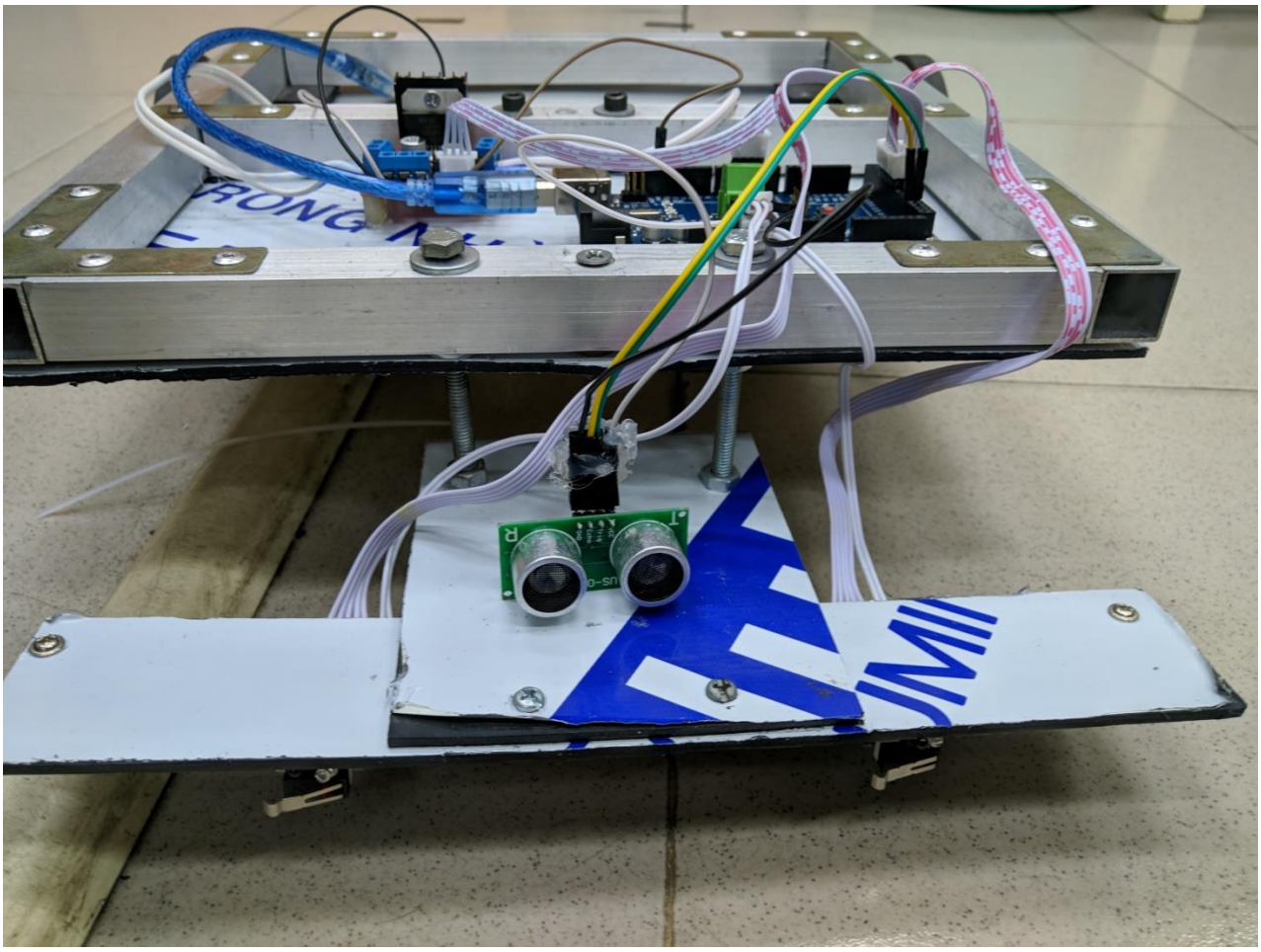
V.Hình cảnh thực tế

- Vi xử lí và module L298 được đính bằng keo silicon, có thể gỡ ra dễ dàng để bắt ốc cố định tại vị trí mong muốn.

- Tấm phẳng phía sau xe cũng có thể bắt óc tại 2 lỗ ở góc dưới, tùy sở thích có thể khoan lỗ và cố định vị trí tùy thích.

- Dây động cơ cắm như hình, đầu dây có mực tô đậm thì cắm chân lẻ của ngõ ra OUT trong L298. (trong mục III, chân dương (+) kí hiệu màu đỏ , chân đất GND kí hiệu màu đen , các màu còn lại là tín hiệu)





VI. Đoạn code Chương trình

```
#include <avr/wdt.h>

volatile int16_t read_line = 0,

        a0 = 0, a1 = 0, a2 = 0, a3 = 0, a4 = 0, a5 = 0, a6 = 0, a7 = 0, a8 = 0, a9 = 0,
// các biến ghi nhớ trạng thái hiện tại của mỗi LED

        pre_a0 = 1, pre_a1, pre_a2, pre_a3, pre_a4, pre_a5, pre_a6, pre_a7,
pre_a8, pre_a9 = 1; // các biến chứa giá trị trước đó của các LED

volatile float   P = 0, I = 0, D = 0, PID_val = 0, // khởi tại các biến cho bộ PID
```

```
        pre_pre_error = 0, pre_error = 0, error = 0,// sai số cảu lần quét led hiện tại, quá khứ
        // hệ số quẹo tại các khúc cua
        Kp = 80, Ki = 0.1, Kd = 45; // giá trị cho bộ PID


volatile uint16_t speed_val = 100, vach = 0; // tốc độ mỗi động cơ, cao nhất là 200 vòng/phút
volatile int16_t Right_motor = 0, Left_motor = 0;


const int trig = 47;     // chân trig của HC-SR04
const int echo = 46;     // chân echo của HC-SR04
unsigned long duration; // biến đo thời gian
int distance,// biến lưu khoảng cách
   dis=20;// khoảng cách so với vật cản muốn xe dừng lại
// hàm reset khi nhấc bánh xe
void ResetBoard(uint8_t time_val)
{
  wdt_enable(time_val);
  while (1) {}
}


void setup()
```

```
{
  // cấu hình Watch Dog Timer
  wdt_enable(WDTO_4S);
  //cấu hình chân cho con cảm biến siêu âm
  pinMode(trig, OUTPUT);  // chân trig sẽ phát tín hiệu
  pinMode(echo, INPUT);   // chân echo sẽ nhận tín hiệu
  // thanh led bên trái
  pinMode(30, INPUT);
  pinMode(32, INPUT);
  pinMode(34, INPUT);
  pinMode(36, INPUT);
  pinMode(38, INPUT);
  // thanh led bên phải
  pinMode(31, INPUT);
  pinMode(33, INPUT);
  pinMode(35, INPUT);
  pinMode(37, INPUT);
  pinMode(39, INPUT);
  // chân quy định chiều quay động cơ
  pinMode(3, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(2, OUTPUT);
```

```
  pinMode(4, OUTPUT);
  digitalWrite(3, LOW);
  digitalWrite(5, LOW);
  // đổi tốc độ thành độ rộng xung để tính toán
  speed_val = speed_val * 1.275;
}
void loop()
{
  Read_led();
  Distance();
  Caculator();
  Test();
  delay(10);
  wdt_reset();
}
void Read_led()
{
  a0 = digitalRead(30);
  a1 = digitalRead(32);
  a2 = digitalRead(34);
  a3 = digitalRead(36);
  a4 = digitalRead(38);
```

```
  a5 = digitalRead(31);
  a6 = digitalRead(33);
  a7 = digitalRead(35);
  a8 = digitalRead(37);
  a9 = digitalRead(39);
  read_line = a0 * 512 + a1 * 256 + a2 * 128 + a3 * 64 + a4 * 32 + a5 * 16 + a6 * 8 + a7 * 4 + a8 * 2  + a9; //chuyển chuỗi nhị phân 10 bit "a0a1a2a3a4a5a6a7a8a9" thành thập phân
  if (((a0 || a9) == 0) || ((a8 || a1) == 0) || ((a8 || a0) == 0) || ((a9 || a1) == 0) || ((a8 || a2) == 0) || ((a9 || a2) == 0))
  {
    vach++;
  }

}

// kiểm tra nếu gần vật chắn hay đi qua vạch sẽ reset ( dùng watch dog timer reset)
void Test()
{
  if ((vach > 5))
  {
    Stop();
```

```
    delay(100);
  }

  while ((distance < dis))
  {
    analogWrite(2, 0); // cấp xung cho 2 động cơ chạy
    analogWrite(4, 0);
    Distance();
  }
  Run();
}
//Sai số hóa vị trí
int16_t  Error()
{
  if ((read_line == 1022) && (pre_a8 == 0))
  {
    pre_a9 = a9;
    pre_a0 = a0;
    return 4;
  }
  else if (read_line == 1020)
  {
```

```
  return 3.5;
}
else if ((read_line == 1021) && (pre_a7 == 0))
{
  pre_a8 = a8;
  return 3;
}
else if (read_line == 1017)
{
  return 2.5;
}
else if (((read_line == 1019) || (read_line == 1009)) && (pre_a6 == 0))
{
  pre_a7 = a7;
  return 2;
}
else if (read_line == 1011)
{
  return 1.5;
}
else if ((read_line == 1015) && (pre_a5 == 0))
{
```

```
  pre_a6 = a6;
  return 1;
 }
 else if (read_line == 999)
 {
  return 0.5;
 }
 else if ((read_line == 1007) && (pre_a4 = 0))
 {
  pre_a5 = a5;
  return 0.3;
 }
 else if ((read_line == 975) || ((read_line == 1023) && ((pre_a4 == 0) || (pre_a5
== 0))))
 {
  return 0;
 }
 else if (read_line == 991)
 {
  pre_a4 = a4;
  return -0.3;
 }
```

```
else if (read_line == 927)
{
  return -0.5;
}
else if ((read_line == 959) && (pre_a4 == 0))
{
  pre_a3 = a3;
  return -1;
}
else if (read_line == 831)
{
  return -1.5;
}
else if (((read_line == 895) || (read_line == 575)) && (pre_a3 == 0))
{
  pre_a2 = a2;
  return -2;
}
else if (read_line == 639)
{
  return -2.5;
}
```

```
  else if ((read_line == 767) && (pre_a2 == 0))
  {
    pre_a1 == a1;
    return -3;
  }
  else if (read_line == 255)
  {
    return -3.5;
  }
  else if ((read_line == 511) && (pre_a1 == 0))
  {
    pre_a9 = a9;
    pre_a0 = a0;
    return -4;
  }
  else if ((read_line == 1023) && (((pre_a0 == 0) && (pre_a1 == 0)) || ((pre_a9 == 0) && (pre_a8 == 0))))
  {
    if (pre_a0 == 1)
    {
      return 5.5;
    }
```

```
   else
   {
     return -5.5;
   }
  }
  return 0;
}

// thuật toán PID trong điều khiển để đạt được vị trí mong muốn ( đưa sai số gần về 0)
void PID()
{
  P = error - pre_error;
  I = pre_error + error;
  D = error - 2 * pre_error + pre_pre_error;
  PID_val = PID_val + (Kp * P) + (Ki * I * 0.5) + (Kd * D);
  pre_error = error;
  pre_pre_error = pre_error;
}

// cấp xung cho cầu H
void Caculator()
```

```
{
  error = Error();
  PID();
  Right_motor = constrain((speed_val -   PID_val), 20, 160); // giới hạn độ rộng xung 2 bánh trong khoảng (20,180)
  Left_motor = constrain((speed_val +  PID_val), 20, 160);
}
void Run()
{
  digitalWrite(3, LOW);
  digitalWrite(5, LOW);
  analogWrite(2, Right_motor); // cấp xung cho 2 động cơ chạy
  analogWrite(4, Left_motor);
}

// Tính toán khoảng cách từ cảm biến siêu âm
void Distance()
{
  /* Phát xung từ chân trig */
  digitalWrite(trig, 0);  // tắt chân trig
  delayMicroseconds(2);
  digitalWrite(trig, 1);  // phát xung từ chân trig
```

```
  delayMicroseconds(5);   // xung có độ dài 5 microSeconds
  digitalWrite(trig, 0);  // tắt chân trig


  /* Tính toán thời gian */
  // Đo độ rộng xung HIGH ở chân echo.
  duration = pulseIn(echo, HIGH);
  // Tính khoảng cách đến vật.
  distance = int(duration / 2 / 29.412);
}
void Stop()
{
  Right_motor = 0;
  Left_motor = 0;
}
```

VII. Chức năng các hàm trong code

1. **Void setup()**
   Cài đặt các chân, thiết lập các thông số ban đầu cho biến, không đặt biến tại đây.
2. **Void loop()**

   Vòng lặp chương trình , viết các hàm cần chạy liên tục tại đây

3. **#include <avr/wdt.h>**

   Thư viện của WatchDog Timer

4. **Int16_t Error()**

Sai số hóa vị trí , mỗi vị trí lệch sẽ có 1 sai số , điểm giữa có sai số là 0 , càng lệch so với line nhiều thì sai số càng lớn. Nếu lệch trái thì sai số dương, lệch phải thì sai số âm.

5. Void Read_led()

Đọc các trạng thái của thanh Led , mỗi led cho tương ứng là 1 bit , gom các bit này vào 1 thanh ghi và đổi ra giá trị thập phân.

6. Void Distance()

Tính toán khoảng cách và phát, nhận sóng cho con siêu âm

7. Void Caculator()

Gọi các hàm PID(), Error() sau đó gán độ rộng xung vào biến để xuất ra động cơ

8. Void PID()

PID là hàm xử lí sai số, dùng trong các hệ thống tự động, ở đây nó giúp đưa ra giá trị PID_val để xe có thể quẹo theo đường line

9. Void Test()

Kiểm tra các biến khoảng cách, số line bẫy đã qua , đã tới vạch đích chưa, phía trước có vật cản không, để cho xe dừng hoặc chạy bình thường.

10.Void Run()

Cấp xung cho module L298 để điều khiển động cơ

11.Void Stop()

Dừng xe

12. wdt_reset()

Khi vi xử lí bị treo không chạy sẽ cho reset để xe có thể khởi động lại từ đầu và chạy.