倒立振子yuna Document　Package

1. 概要
2. 基本仕様
3. 製作方法
4. 設計図
5. 回路図
6. プログラム

　・フローチャート

　　・Arduino　sketch

令和7年5月6日

**1．倒立振子yunaの概要**

　　ジャイロセンサーとN20モーター2個を使用しArduino　nanoで制御して倒立振子を作成。上部には100均購入した着せ替え人形yunaちゃんを搭載し安定した倒立振子を完成した。

**2．基本仕様**

・使用プロセッサー　Arduino Nano

・N20モーター　2個

・ジャイロセンサーMPU6050

・モータードライバーLV48MC-AH

・抵抗220Ω×2

・LED×2

・スライドスイッチ×1

・タクトスイッチ×1

・ダイオード　SD103A×1

・電源コネクター他×1

・コンデンサー10μF×1

・ユニバーサル基板

・車輪　34mmφ×2

・構造物部材、ビス、ナット他

**3．製作方法**

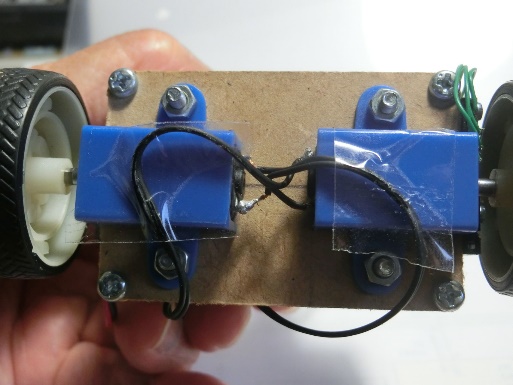
　　設計図に従って3段の基礎構造物を作成。下段に9V電池とジャイロセンサーを配置。

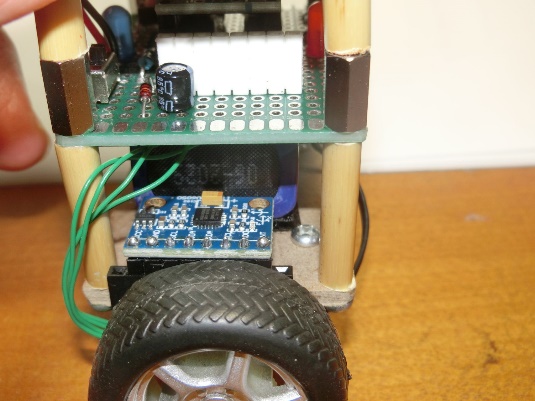
中断はユニバーサル基板を利用して電子部品を取り付け上、段はyunaちゃんを搭載。



中段に取り付けたLEDは行進方向の指標

として取り付け

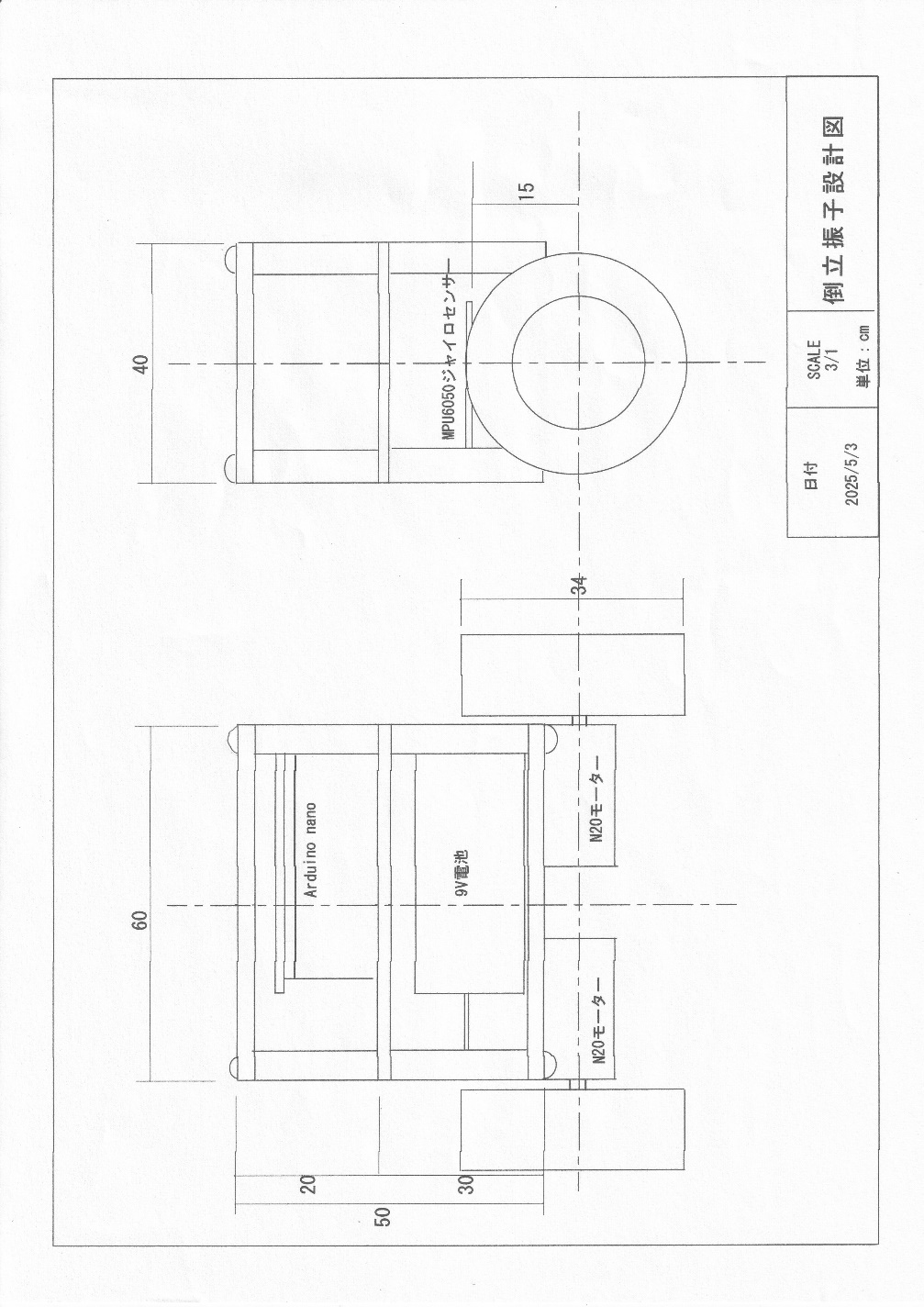
　N20モーターの取り付け状況

　ジャイロセンサーの取り付け位置

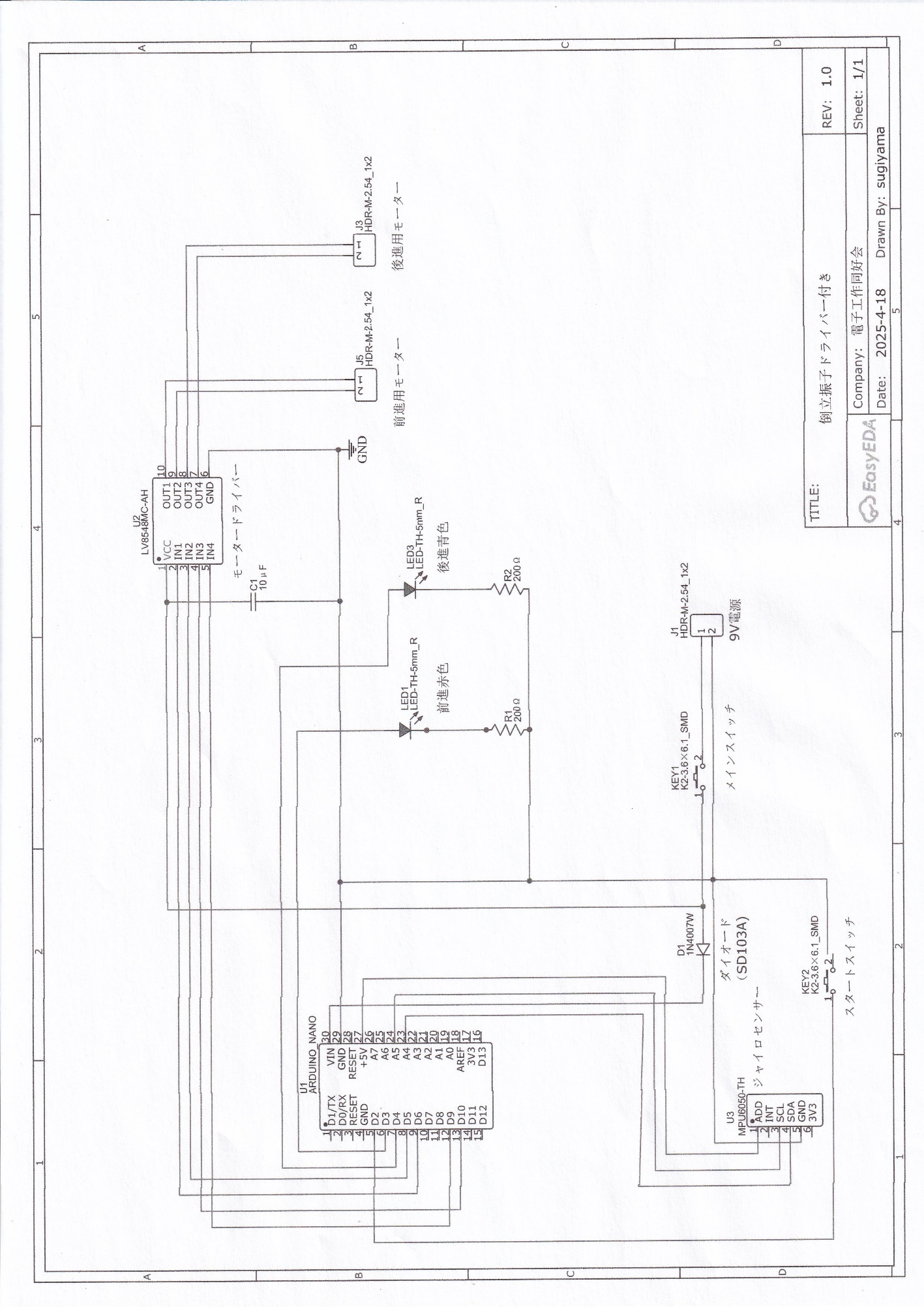
　中段の白色のコネクターは当初取り付けた

　位置。ハンチングが激しく安定しないので

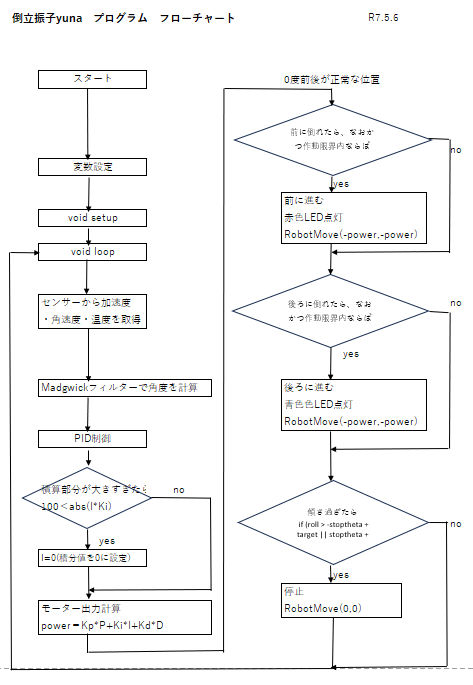
下段に位置を変更



**4．設計図**

**5．回路図**

**6．プログラム**

**・フローチャート**

**・スケッチ**

倒立振子yunaプログラム

令和7年5月6日

// 倒立振子 基本形

// https://shizenkarasuzon.hatenablog.com/entry/2019/02/16/181342

// https://qiita.com/coppercele/items/e4d71537a386966338d0

// https://qiita.com/coppercele/items/527228e3f08c53597bd1

//N20モーターで設計R4.4.7

//MPU6050を電池位置に変更。向きが反対なのでpowerをー（マイナス）とした。

//モータードライバーLV8548を取り付けてドライバーに9Vを供給してパワーアップした。

//モーターはN20モーター2個

//モーターの作動はRobotMoveを使用

# include <Wire.h> // I2C通信用ライブラリ

# include <MadgwickAHRS.h> // 角度と角速度にMadgwickフィルタをかけるライブラリ

Madgwick MadgwickFilter; // Madgwickフィルタのオブジェクトを設定

int \_spdL = 0 ;

int \_spdR = 0 ;

# define MPU6050\_PWR\_MGMT\_1 0x6B // Read and Write

# define MPU\_ADDRESS 0x68

int stoptheta = -35; // 倒れすぎたらモータを止める角度

float Kp = 700.0; // Pゲイン 全体的なモータのパワーに効く

float Ki = 9000.0; // Iゲイン バランスが崩れて全体が平行移動しだしたときに効く

float Kd = 3.0; // Dゲイン 激しく動いたときのブレーキの役割をする

//float target = 83.0; // 目標値。モジュールが横置きなら0前後、縦置きなら90前後

float target = 0.7;//目標値　モジュールが横置きなので0度とした。

// PIDパラは倒立しうる最小の値の組み合わせがベストらしい。これは大きいかもしれない。

// 目標角度は機体の前後の重量バランスによる。偏ると平行移動しがち。ある意味PIDパラより重要かも

float P, I, D, preP; // 偏差、その微分・積分、積分記録用

float dt, preTime; // 処理時間

float power = 0; // 連続回転サーボの出力(90からどれだけ動かすか)

void setup() {

Wire.begin();

Serial.begin(115200); //シリアル通信を115200bpsに設定

// 動作モードの読み出し

Wire.beginTransmission(MPU\_ADDRESS);

Wire.write(MPU6050\_PWR\_MGMT\_1); // MPU6050\_PWR\_MGMT\_1レジスタの設定

Wire.write(0x00);

Wire.endTransmission();

MadgwickFilter.begin(100); // Madgwickフィルタの周波数を100Hzに設定。

// ここを変えると角度の感度が変わる

// 100Hzだと妙に角度が暴れたりはしないけど、0°→90°になるまでちょっと計算に時間がかかるから、↓

// 起動したら少し待たないと倒立しない。かといってすぐ90°になるようにすると↓

// シリアル通信で見た限りは角度が暴れるから、うまく倒立しないかもしれない。

pinMode( 3 , OUTPUT); // 赤色LED

pinMode( 4 , OUTPUT); // 青色LED

pinMode( 5 , OUTPUT); // 左モーター反転

pinMode( 6 , OUTPUT); //　左モーター正転

pinMode( 9 , OUTPUT); //　右モーター反転

pinMode( 10 , OUTPUT); //　右モーター正転

pinMode(2,INPUT\_PULLUP); //　スタートボタン

while(((digitalRead(2)))==(true))

{ delay(100);

}

}

void loop() {

Wire.beginTransmission(0x68);

Wire.write(0x3B);

Wire.endTransmission(false);

Wire.requestFrom(0x68, 14, true);

while (Wire.available() < 14);

int16\_t axRaw, ayRaw, azRaw, gxRaw, gyRaw, gzRaw, Temperature;

// センサーから加速度、角速度、温度を取得

axRaw = Wire.read() << 8 | Wire.read();

ayRaw = Wire.read() << 8 | Wire.read();

azRaw = Wire.read() << 8 | Wire.read();

Temperature = Wire.read() << 8 | Wire.read();

gxRaw = Wire.read() << 8 | Wire.read();

gyRaw = Wire.read() << 8 | Wire.read();

gzRaw = Wire.read() << 8 | Wire.read();

// 加速度値を分解能で割って加速度(G)に変換する

float acc\_x = axRaw / 16384.0; //FS\_SEL\_0 16,384 LSB / g

float acc\_y = ayRaw / 16384.0;

float acc\_z = azRaw / 16384.0;

// 角速度値を分解能で割って角速度(degrees per sec)に変換する

float gyro\_x = gxRaw / 131.0; // (度/s)

float gyro\_y = gyRaw / 131.0;

float gyro\_z = gzRaw / 131.0;

// Madgwickフィルターを用いて、PRY（pitch, roll, yaw）を計算

MadgwickFilter.updateIMU(gyro\_x, gyro\_y, gyro\_z, acc\_x, acc\_y, acc\_z);

// rollの計算結果のみを取得する。pitchとyawは使わない。このrollが現在の角度になる。

// 前に傾いたら0+x°、後ろに傾いたら0-x°の値をとる。

float roll = MadgwickFilter.getRoll();

// Serialに表示

Serial.print("roll =\t"); Serial.print(roll); Serial.print(",\t");

dt = (micros() - preTime) / 1000000; // 処理時間を求める

preTime = micros(); // 処理時間を記録

// PID制御

// 目標角度から現在の角度を引いて偏差を求める

P = (target - roll) / 90; // -90~90を取るので180で割って-1.0~1.0にする

I += P \* dt; // 偏差を積分する

D = (P - preP) / dt; // 偏差を微分する

preP = P; // 偏差を記録する

// Serialに表示

Serial.print("P =\t"); Serial.print(P); Serial.print(",\t");

Serial.print("I =\t"); Serial.print(I); Serial.print(",\t");

Serial.print("D =\t"); Serial.print(D); Serial.print(",\t");

// 積分部分が大きくなりすぎると出力が飽和するので大きくなり過ぎたら0に戻す(アンチワインドアップ)

if (1000 < abs(I \* Ki))I = 0;

// 出力を計算する

power = Kp \* P + Ki \* I + Kd \* D;

// power = constrain(power, -50, 50); // ±50に制限

// 後ろに傾くとpowerはプラス、前に傾くとマイナス

Serial.print("power =\t"); Serial.print(power); Serial.print("\n");

// 角度を検知してモータを動作させる。(倒立振子の主動作)

// 前に傾いたら前に進んで相殺

// 目標角83°<現在の角度(例:100°)<83+65=148°なら前に進んで83°に減らそうとする。

if ( target > roll && roll > stoptheta + target) {

// 連続回転サーボが左右でちょうどよく回転するようにオフセットを調整したいけど、↓

// 軸がぶれてたり個体差があったりするからどうしようもないかもしれない。

// myservo1.write(92 + power); // 92で左前方に旋回、93,94,95で右前方に旋回

// myservo2.write(90 - power);

digitalWrite(3,HIGH);

digitalWrite(4,LOW);

RobotMove(-power, -power);

}

// 後ろに傾いたら後ろに進んで相殺

// 83-65=18°<現在の角度(例:70°)<目標角83°なら後ろに進んで83°に増やそうとする。

if (-stoptheta + target > roll && roll > target) {

// myservo1.write(90 + power);

// myservo2.write(93 - power); // 92で右後方に旋回（早すぎ）、93で左後方に旋回

digitalWrite(3,LOW);

digitalWrite(4,HIGH);

RobotMove(-power, -power);

}

// 倒れすぎたら停止(倒れても動き続けるとうるさいし事故るから)

// 148°より大、もしくは18°より小ならモーターを止め、PIDパラもゼロにする。

if (roll > -stoptheta + target || stoptheta + target > roll) {

// digitalWrite(13, LOW);

// myservo1.write(90);

// myservo2.write(90);

digitalWrite(3,HIGH);

digitalWrite(4,HIGH);

RobotMove(0, 0);

P = 0;

I = 0;

D = 0;

return;

}

}

void RobotMove(int spdL, int spdR){

if (spdL>=0) {

analogWrite(6, 255);

analogWrite(5, constrain(255-spdL, 0, 255));

}

else {

analogWrite(5, 255);

analogWrite(6, constrain(255+spdL, 0, 255));

}

if (spdR>=0) {

analogWrite(10, 255);

analogWrite(9, constrain(255-spdR, 0, 255));

}

else {

analogWrite(9, 255);

analogWrite(10, constrain(255+spdR, 0, 255));

}

\_spdL=spdL;

\_spdR=spdR;

}