ENGIN 104 Fall 2020

University of Massachusetts Boston

Final Project Writeup

Walter – The Insect Bot

Name: Antonio Moura

Abstract

For this project, I used the Arduino Uno Board to build a robot that looks like an insect with 4 legs. The robot similarly to an insect it should be able to move, detect light, and avoid any obstacles. I was only able to make the robot move and avoid obstacles. To program it to do the task I mentioned, I used the Arduino IDE.

Introduction

This project is about a Walking Mechanism that looks like an insect. Walter uses the same technology as the Vbug 1.5 (also known as Walkman), a famous 6-legged robot created by Mark Tilden (The founder of beam robotic). Walter and Walkman's movement and behavior are like insects. They prove that we can create many different robots that can copy various animals' actions and behaviors. Walter is only able to walk, avoid the objects, and detect the light. It is an excellent robot for beginners to build because you'll get a new experience, and it is not difficult to make one.

I chose to do this robot because I wanted to build some robotic for my final project, so it would be my first step to becoming a mechanical engineer. I have worked with different electronic devices, but this is the first time I made a robot from scratch and on my own. I got this robot from the internet; the link is below:

https://www.instructables.com/WALTER-Arduino-Contest/

Methods

I wanted to build the original Walter, with the capability to follow the light like a real bug, but I had no photodiodes, so I had to make some changes. My Walter is easier than the original one--only walks and avoid obstacles. Also, I had to use different components from the original. The electrical circuit and schematic are also different from the original Walter, but the code is the same. Therefore, my Walter's setup includes:

<u>Components and Supplies:</u>

- 1 Arduino Uno board
- 1 Small breadboard
- 3 Ultrasonic Sensor – HC- SR04
- 5 MG90S Metal Geared Micro Servo
- 2 Power Supplies (total of 4 batteries of 1.5V to each power supply)
- 3 Different type of glues (I used a lot of glue), hot glue, plastic stee epoxy glue, and super glue
- 4 Thin metal rods easy to shape

<u>Circuit Diagram</u>

The electrical circuit I built is represented at the following diagram (figure 1):

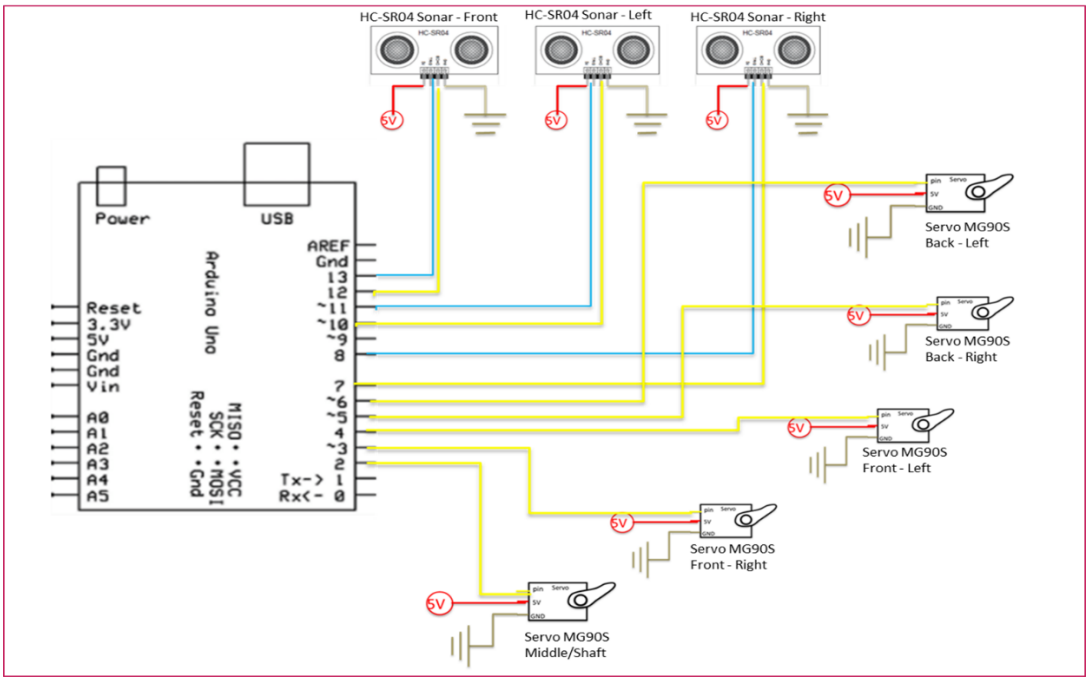

Figure 1. Walter's circuit diagram

As we can see in the figure 1, each servos and ultrasonic sensors are getting 5V as power supply, and they are all connected to a specific digital pin in Arduino Uno Board.

Schematic of Setup

The Circuit I made is a version for Arduino Uno, where two power supplies are needed. The schematic of the breadboard/ Arduino board is represented in the figure 2.

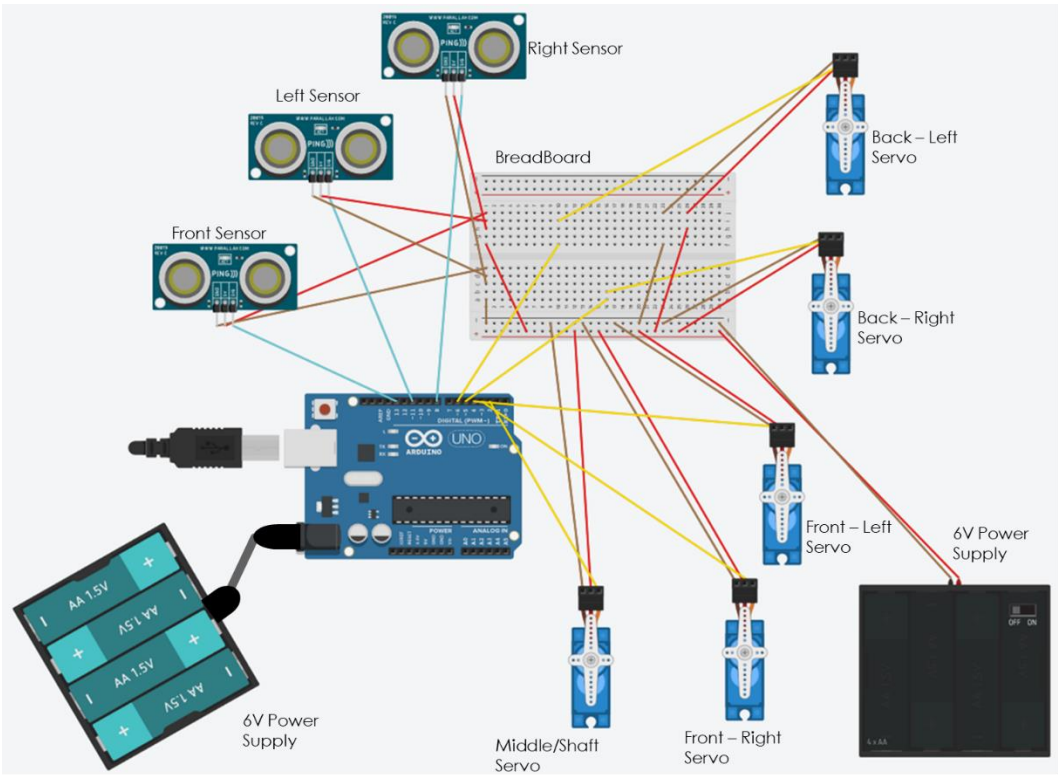

Figure 2. Schematic of Walter

The only hardware of my robot is the 4 thin metal rods for the legs, that are connected to the setup at the servos (figure 3). By using a lot of glue and tape I added the 4 thin metal rods (Walter's legs) to the circuit by gluing them to the servos.

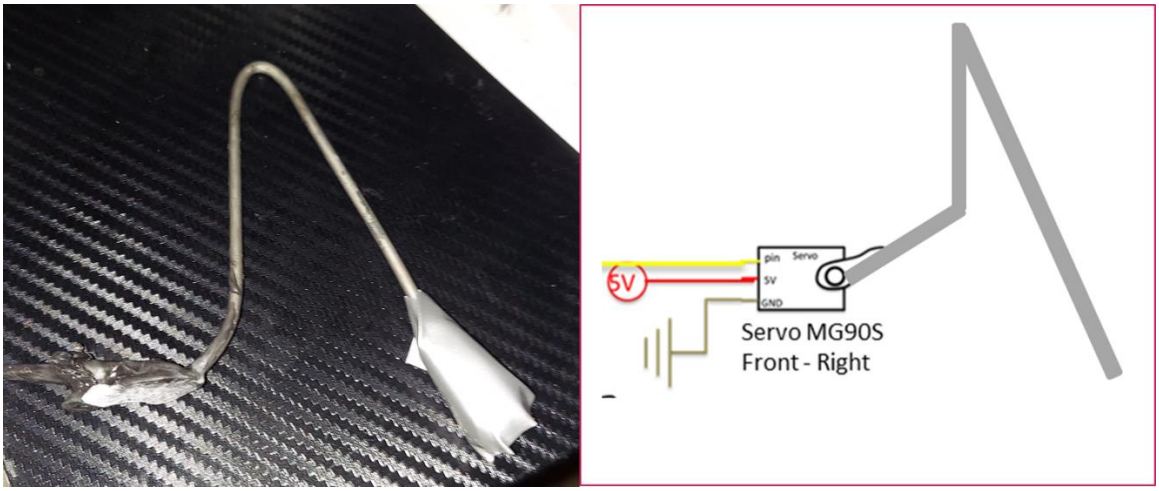

Figure 3. The Thin rods connected to the setup

Code

I program my robot using Arduino IDE. The code I used is from the internet, is the same as the original Walter. The code is very long, and I placed it in the Appendix. But I will show the first part of the code and summarize the main points. First, here is the link of the source: https://content.instructables.com/ORIG/F29/1R18/JAWTUA92/F291R18JAWTUA92.txt

```
const int ANGLE_mid_Shaft = 1520;
const int ANGLE_mid_FLeft = 1550;
const int ANGLE_mid_FRight = 1570;
const int ANGLE_mid_BLeft = 1450;
const int ANGLE_mid_BRight = 1450;

const int ANGLE_sweep = 250;        // **Set this value (in microseconds) to determine how wide the servos will sweep (legs' step width). Bigger value means wider sweep angle.
const int ANGLE_res = 10;           // **Set the servo movement resolution (in microseconds) at least at minimum servo's default dead band width (highest resolution) or more (less resolution). Example: SG90 servo dead band width is 10 microseconds.
int sweepSPEED;                     // variable to determine how fast the servos will sweep.
int sweepSPEED_Rand[3] = {4, 6, 8}; // **Servo speed (gait speed) will change randomly in 3 modes. Set the speed (in miliseconds) for each mode. Smaller value means faster.

const int ANGLE_turnMAX = ANGLE_sweep * 1.5;     // **Set this value to determine how much maximum the bot will turn toward light. Bigger value means bigger turn.
const int ANGLE_turnNARROW = ANGLE_sweep * 0.25; // **Set this value to determine how much maximum the bot will turn avoiding objects at its sides in narrow space. Bigger value means bigger turn.

const int SONAR_sum = 3; // Amount of sonars used.
const int PHOTO_sum = 4; // Amount of photodiodes used.
int PIN_trig[SONAR_sum] = {13, 11, 8};   // ***Set arduino pins connected to ultrasonic sensors' trigger pins; {front, left, right}.
int PIN_ec[SONAR_sum] = {12, 10, 7};     // ***Set arduino pins connected to ultrasonic sensors' echo pins; {front, left, right}.
int PIN_PHOTO[PHOTO_sum] = {2, 3, 1, 0}; // ***Set arduino analog input pins connected to photodiodes; {front left, front right, back left, back right}.

const int distRotate = 25;       // **Configure minimum distance (in cm) between the robot and obstacle before the robot avoid it by rotating.
const int distRetreat = 10;      // **Configure minimum distance (in cm) between the robot and obstacle before the robot avoid it by retreating.
const int distTurn = 20;         // **Configure minimum distance (in cm) between the robot and obstacle before the robot avoid it by turning.
const int counter_gait_max = 8;  // **Configure how many steps the robot will take to avoid obstacle (when rotating or retreating).

// **Configure how long the bot rest & run (in milliseconds).
const int RUN_time = 25000;
const int REST_time = 3000;
```

This first part of the code is the initialization of the variables. So the code works as following:

1. The variables are initialized, where it gave values to the variables of the angles of the legs, it set values to the maximum and minimum time it takes for the servos to change their position (moving ability), it set a max and min value which relates to the action of the robot according to the distance of the object detected by the ultrasonic sensor.

2. Therefore, after initializing the variables, all the components are set to their specific pins.

3. By using the variables with their respective values, the code will make the robot moves at three different speed (zero speed, medium, and high) depending on the value of time it takes to for the servos to rotate.

4. Also, when it detects a object depending on the distance the robot will retreat and rotate, or will simply turn.

5. The light detecting part, does not apply to my robot.

Results

My robot was able to walk but with some difficulties because of the weight. It could not do the required actions according to the sensors. It can only walk on places with a lot of friction, like carpet. I wanted to solder everything, but I was having trouble doing it, so I decided to use the breadboard (more weight). The figure 4 and 6 is to show how the components were assempled And the figure 6 is the result.

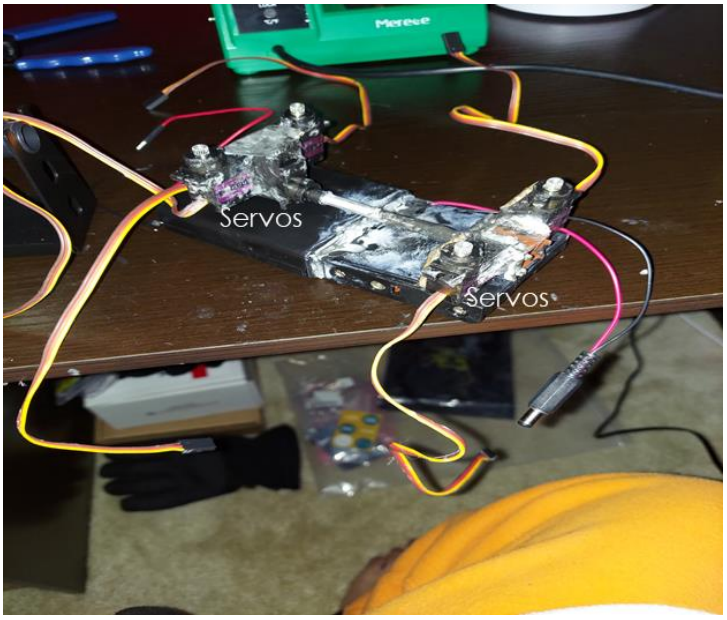

Figure 4. All the servos glued at each other making the "skeleton" of the robot

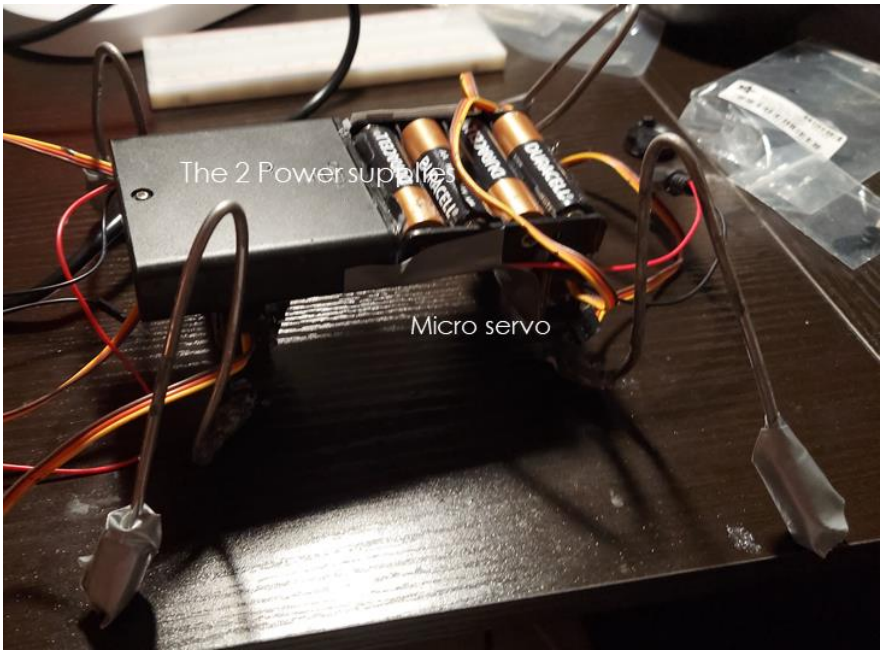

Figure 5. The Power supplies glued at each other along with the servos and the thin rods (legs)

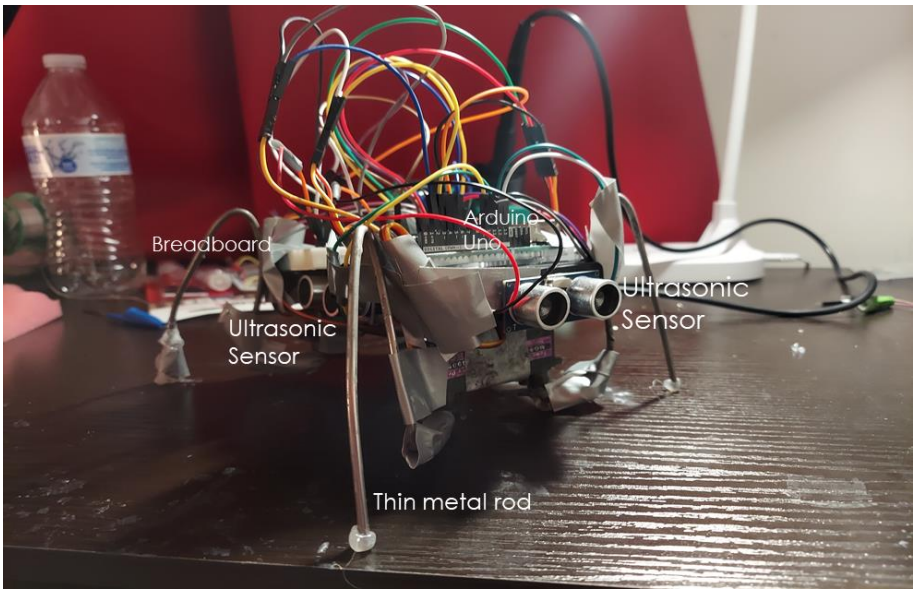

Figure 6. My version of Walter-- Walter L3003

This project was very interesting. I learned how to better make electrical circuits, specially how to work with the transmission wires. I also learned that when building a robot weight becomes a very important feature to take in consideration.

Conclusions

My Walter can move, although it struggles to do it. The sensor works just fine, but Walter cannot avoid the obstacles. I feel accomplished because I was able to do my first robot. It is fun to build it. For the improvements I would make sure that the legs are at the same size and shape and try to use the Arduino pro mini and the smartphone power bank instead of the two 6V power supplies.

Appendix

The code used for Walter:

```
/*

WALTER - THE 4 LEGGED PHOTOVORE


This Arduino sketch is my attemp to build a 5 servo quadruped (4 legged) robot named "WALTER".


In order to use this sketch, you might need to change some values for your convenient or to adjust to your own hardware set up.

Find (Ctrl + F) these marks to easily search which values might need to be changed:

  - **** : These marks mean they are servos' center potition and have to calibrated (your robot legs' potitioning when it's idle).

  - ***  : These marks mean arduino pins asignment (sensors & servos connection to arduino). Refer to this when you build the robot.

  - **   : These marks mean that the values optionally can be changed according to your taste (legs' step width, how much to turn when sensing light/obstacle, etc). Just leave it as is if you don't know what you were doing.


You can use this sketch at your own risk.. and, it's provided as is and.. uh..

What're those all about copyrights stuff people use to write at published source code??

The point is I don't want to be held responsible if something bad happened when you were using this codes.

Have fun!


Yohanes Martedi - 2015

*/


#include <Servo.h>


// ****Calibrate servos' center angle (in microseconds because we'll use "xx.writeMicroseconds();" command). Start with 1500.

const int ANGLE_mid_Shaft = 1520;

const int ANGLE_mid_FLeft = 1550;

const int ANGLE_mid_FRight = 1570;

const int ANGLE_mid_BLeft = 1450;

const int ANGLE_mid_BRight = 1450;


const int ANGLE_sweep = 250;        // **Set this value (in microseconds) to determine how wide the servos will sweep (legs' step width). Bigger value means wider sweep angle.
```

const int ANGLE_res = 10; // **Set the servo movement resolution (in microseconds) at least at minimum servo's default dead band width (highest resolution) or more (less resolution). Example: SG90 servo dead band width is 10 microseconds.

int sweepSPEED; // variable to determine how fast the servos will sweep.

int sweepSPEED_Rand[3] = {4, 6, 8}; // **Servo speed (gait speed) will change randomly in 3 modes. Set the speed (in miliseconds) for each mode. Smaller value means faster.

const int ANGLE_turnMAX = ANGLE_sweep * 1.5; // **Set this value to determine how much maximum the bot will turn toward light. Bigger value means bigger turn.

const int ANGLE_turnNARROW = ANGLE_sweep * 0.25; // **Set this value to determine how much maximum the bot will turn avoiding objects at its sides in narrow space. Bigger value means bigger turn.

const int SONAR_sum = 3; // Amount of sonars used.

const int PHOTO_sum = 4; // Amount of photodiodes used.

int PIN_trig[SONAR_sum] = {13, 11, 8}; // ***Set arduino pins connected to ultrasonic sensors' trigger pins; {front, left, right}.

int PIN_ec[SONAR_sum] = {12, 10, 7}; // ***Set arduino pins connected to ultrasonic sensors' echo pins; {front, left, right}.

int PIN_PHOTO[PHOTO_sum] = {2, 3, 1, 0}; // ***Set arduino analog input pins connected to photodiodes; {front left, front right, back left, back right}.

const int distRotate = 25; // **Configure minimum distance (in cm) between the robot and obstacle before the robot avoid it by rotating.

const int distRetreat = 10; // **Configure minimum distance (in cm) between the robot and obstacle before the robot avoid it by retreating.

const int distTurn = 20; // **Configure minimum distance (in cm) between the robot and obstacle before the robot avoid it by turning.

const int counter_gait_max = 8; // **Configure how many steps the robot will take to avoid obstacle (when rotating or retreating).

// **Configure how long the bot rest & run (in milliseconds).

const int RUN_time = 25000;

const int REST_time = 3000;

// ID's for sonars:

int SONAR_id;

const int FRONT = 0;

const int LEFT = 1;

const int RIGHT = 2;

// ID's for photodiodes:

const int FRONT_LEFT = 0;

const int FRONT_RIGHT = 1;

```
const int BACK_LEFT = 2;

const int BACK_RIGHT = 3;


// Variables for photodiodes reading:

int PHOTO_Front_Left;

int PHOTO_Front_Right;

int PHOTO_Back_Left;

int PHOTO_Back_Right;


const int SONAR_TrigSig = 10;          // Duration (in S) of trigger signal the sensors needed to produce ultrasonic sound (already specified by the products, don't change this value).

const unsigned long SONAR_MaxEc = 50000;  // Maximum duration (in S) of the echo signal given by the sensors (already specified by the products, don't change this value).

const float SOUND_speed = 0.034;        // The speed of sound on air in S/cm (already specified by sciene, avatar Aang is needed to do air bending if this value is wanted to be changed).

int distance[SONAR_sum];               // Results of the calculation of distance.


// Ddeclaration of servos:

Servo SERVO_shaft;

Servo SERVO_front_left;

Servo SERVO_front_right;

Servo SERVO_back_left;

Servo SERVO_back_right;


// Variables for each servos' angles:

int ANGLE_shaft = ANGLE_mid_Shaft;

int ANGLE_front_left = ANGLE_mid_FLeft;

int ANGLE_front_right = ANGLE_mid_FRight;

int ANGLE_back_left = ANGLE_mid_BLeft;

int ANGLE_back_right = ANGLE_mid_BRight;


// Angle manipulation for middle servo (shaft).

const int ANGLE_max_Shaft = ANGLE_mid_Shaft + ANGLE_sweep;

const int ANGLE_min_Shaft = ANGLE_mid_Shaft - ANGLE_sweep;

int ANGLE_sweep_val;


// Variables for recording current angles of each servos:
```

```
int ANGLE_shaft_record;
int ANGLE_front_left_record;
int ANGLE_front_right_record;
int ANGLE_back_left_record;
int ANGLE_back_right_record;

// Variables for servos angles correction according to light detection:
int LIGHT_left;
int LIGHT_right;
// Variables for servos angles correction according to sonar detection:
int SONAR_left;
int SONAR_right;

// That things such as flags, counters, records that I'm always not sure how to explain. :(
int ANGLE_prev;
int flag_shaft_reverse;
int flag_transition_rotate;
int flag_transition_start = 1;
int flag_rest = 0;
int flag_RUN_time = 0;
int rotate_random;
int counter_gait;

void setup() {
  // Serial.begin(9600); // Serial.. you know, checking & debugging..

  SERVO_shaft.attach(2);        // ***Set up horizontal (shaft) servo's signal pin on arduino.
  SERVO_front_left.attach(4);   // ***Set up front-left servo's signal pin on arduino.
  SERVO_front_right.attach(3);  // ***Set up front-right servo's signal pin on arduino.
  SERVO_back_left.attach(6);    // ***Set up back-left servo's signal pin on arduino.
  SERVO_back_right.attach(5);   // ***Set up back-right servo's signal pin on arduino.

  // Get the servos ready at their middle angles.
  SERVO_shaft.writeMicroseconds(ANGLE_mid_Shaft);
  SERVO_front_left.writeMicroseconds(ANGLE_mid_FLeft);
```

```
  SERVO_front_right.writeMicroseconds(ANGLE_mid_FRight);
  SERVO_back_left.writeMicroseconds(ANGLE_mid_BLeft);
  SERVO_back_right.writeMicroseconds(ANGLE_mid_BRight);

  // Setting pins for sonars, both pinMode and value.
  for(SONAR_id = 0; SONAR_id < SONAR_sum; SONAR_id++) {
    pinMode(PIN_trig[SONAR_id],OUTPUT);
    pinMode(PIN_ec[SONAR_id],INPUT);
    digitalWrite(PIN_trig[SONAR_id], LOW);
  }

  randomSeed(analogRead(5));  // Using analog pin 5 to generate random values to be used later.
  SONAR_READ_ALL();           // Initiate first sonar reading before doing anything,
  SONAR_READ_ALL();           // and actually I forget why I have to call it twice for the first time. But believe me, it's needed.
  START();                    // This function make sure which legs are lifted randomly at the first step.

  delay(3000); // **dum.. daa.. dum..
}

void loop() {
  int state = random(0,2);

  if(state == 0) {
    REST();
  }
  else {
    int randomSPEED = random(0,3);
    sweepSPEED = sweepSPEED_Rand[randomSPEED];
    RUN();
    flag_rest = 0;
  }
}

void REST() {
  if(flag_rest == 0) {
```

```
    TRANSITION_GAIT();
    flag_rest = 1;
  }
  delay(REST_time);
}

void RUN() {
  unsigned long TIMER_state = millis();
  while((millis() - TIMER_state) <= RUN_time) {
    if(distance[FRONT] > distRotate) {
      flag_RUN_time = 0;
      while(flag_RUN_time == 0) {
        FORWARD();
      }
    }

    while(distance[FRONT] > distRetreat && distance[FRONT] <= distRotate) {
      while(distance[LEFT] > distance[RIGHT]) {
        ROTATE_LEFT_AVOID();
        break;
      }
      while(distance[LEFT] < distance[RIGHT]) {
        ROTATE_RIGHT_AVOID();
        break;
      }
      while(distance[LEFT] == distance[RIGHT]) {
        ROTATE_RANDOM_AVOID();
        break;
      }
    }

    while(distance[FRONT] <= distRetreat) {
      RETREAT_AVOID();
    }
  }
```

```
}

/*_____________________________________________________________________________________
######################################### GAIT #########################################*/

/*================================== SHAFT MOVEMENT ==================================*/

void SHAFT() {
  unsigned long TIMER_servo = millis();
  while((millis() - TIMER_servo) <= sweepSPEED) {
    while(ANGLE_shaft == ANGLE_mid_Shaft) {
      counter_gait++;
      SONAR_READ_ALL();
      LIGHT_COMPARE_EXECUTE();
      SIDE_AVOID();
      flag_RUN_time = 1;
      break;
    }
  }

  if(ANGLE_prev < ANGLE_shaft && ANGLE_shaft < ANGLE_max_Shaft) {
    ANGLE_prev = ANGLE_shaft;
    ANGLE_shaft += ANGLE_res;
  }
  else if(ANGLE_shaft >= ANGLE_max_Shaft) {
    ANGLE_prev = ANGLE_shaft;
    ANGLE_shaft -= ANGLE_res;
  }
  else if(ANGLE_prev > ANGLE_shaft && ANGLE_shaft > ANGLE_min_Shaft) {
    ANGLE_prev = ANGLE_shaft;
    ANGLE_shaft -= ANGLE_res;
  }
  else if(ANGLE_shaft <= ANGLE_min_Shaft) {
    ANGLE_prev = ANGLE_shaft;
    ANGLE_shaft += ANGLE_res;
```

```
  }
  SERVO_shaft.writeMicroseconds(ANGLE_shaft);
}

void SHAFT_REVERSE() {
  if(ANGLE_prev < ANGLE_shaft) {
    ANGLE_prev = ANGLE_shaft + 1;
  }
  else if(ANGLE_prev > ANGLE_shaft) {
    ANGLE_prev = ANGLE_shaft - 1;
  }
}

/*================================ END OF SHAFT MOVEMENT ================================*/


/*===================================== TRANSITION =====================================*/

void TRANSITION_GAIT() {
  ANGLE_front_left_record = ANGLE_front_left;
  ANGLE_front_right_record = ANGLE_front_right;
  ANGLE_back_left_record = ANGLE_back_left;
  ANGLE_back_right_record = ANGLE_back_right;
  ANGLE_shaft_record = ANGLE_shaft;

  int flag = HIGH;
  int counter = 0;
  while(flag == HIGH) {
    SHAFT();
    LIGHT_left = 0;
    LIGHT_right = 0;

    counter++;

    ANGLE_front_left = map(counter, 1, ((ANGLE_sweep * 2) / ANGLE_res), ANGLE_front_left_record, ANGLE_mid_FLeft);
```

```
    ANGLE_front_right = map(counter, 1, ((ANGLE_sweep * 2) / ANGLE_res), ANGLE_front_right_record, ANGLE_mid_FRight);

    ANGLE_back_left = map(counter, 1, ((ANGLE_sweep * 2) / ANGLE_res), ANGLE_back_left_record, ANGLE_mid_BLeft);

    ANGLE_back_right = map(counter, 1, ((ANGLE_sweep * 2) / ANGLE_res), ANGLE_back_right_record, ANGLE_mid_BRight);


    SERVO_shaft.writeMicroseconds(ANGLE_shaft);

    SERVO_front_left.writeMicroseconds(ANGLE_front_left);

    SERVO_front_right.writeMicroseconds(ANGLE_front_right);

    SERVO_back_left.writeMicroseconds(ANGLE_back_left);

    SERVO_back_right.writeMicroseconds(ANGLE_back_right);


    if(counter == ((ANGLE_sweep * 2) / ANGLE_res)) {

      flag = LOW;

      START();

      flag_transition_rotate = 0;

    }

  }

}


void TRANSITION_START() {

  if(ANGLE_shaft == ANGLE_mid_Shaft || (ANGLE_shaft > ANGLE_mid_Shaft && ANGLE_shaft > ANGLE_prev) || (ANGLE_shaft < ANGLE_mid_Shaft && ANGLE_shaft < ANGLE_prev)) {

    ANGLE_sweep_val = 0;

  }

  else {

    flag_transition_start = 0;

  }

}


void START() {

  ANGLE_prev = random((ANGLE_shaft), (ANGLE_shaft + 2));

  if(ANGLE_prev == ANGLE_shaft) {

    ANGLE_prev -= 1;

  }

  flag_transition_start = 1;

}
```

```
int ROTATE_RANDOM() {
  return random(0, 2);
}

/*================================== END OF TRANSITION ==================================*/


/*========================================= WALK =========================================*/

void FORWARD() {
  while(flag_transition_rotate == 2) {
    TRANSITION_GAIT();
  }
  flag_transition_rotate = 1;

  while(flag_shaft_reverse == 0) {
    SHAFT_REVERSE();
    break;
  }
  flag_shaft_reverse = 1;

  while(flag_transition_start == 1) {
    SHAFT();
    TRANSITION_START();
    WALK();
  }

  SHAFT();
  ANGLE_sweep_val = ANGLE_sweep;
  WALK();
}

void RETREAT() {
  while(flag_transition_rotate == 2) {
```

```
    TRANSITION_GAIT();
  }
  flag_transition_rotate = 1;

  while(flag_shaft_reverse == 1) {
    SHAFT_REVERSE();
    break;
  }
  flag_shaft_reverse = 0;

  while(flag_transition_start == 1) {
    SHAFT();
    LIGHT_left = 0;
    LIGHT_right = 0;
    TRANSITION_START();
    WALK();
  }

  SHAFT();
  LIGHT_left = 0;
  LIGHT_right = 0;
  ANGLE_sweep_val = (ANGLE_sweep * -1);
  WALK();
}

void WALK() {
  if(ANGLE_shaft >= ANGLE_mid_Shaft && ANGLE_prev < ANGLE_shaft) {
    ANGLE_front_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, ((ANGLE_mid_FLeft - ANGLE_sweep_val) + LIGHT_left + SONAR_left), ANGLE_mid_FLeft);
    ANGLE_front_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, ((ANGLE_mid_FRight - ANGLE_sweep_val) - LIGHT_right - SONAR_right), ANGLE_mid_FRight);
    ANGLE_back_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, ((ANGLE_mid_BLeft + ANGLE_sweep_val) - LIGHT_left - SONAR_left), ANGLE_mid_BLeft);
    ANGLE_back_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, ((ANGLE_mid_BRight + ANGLE_sweep_val) + LIGHT_right + SONAR_right), ANGLE_mid_BRight);
  }
  else if(ANGLE_shaft >= ANGLE_mid_Shaft && ANGLE_prev > ANGLE_shaft) {
```

```
  ANGLE_front_left = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FLeft, ((ANGLE_mid_FLeft + ANGLE_sweep_val) - LIGHT_left - SONAR_left));

  ANGLE_front_right = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FRight, ((ANGLE_mid_FRight + ANGLE_sweep_val) + LIGHT_right + SONAR_right));

  ANGLE_back_left = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BLeft, ((ANGLE_mid_BLeft - ANGLE_sweep_val) + LIGHT_left + SONAR_left));

  ANGLE_back_right = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BRight, ((ANGLE_mid_BRight - ANGLE_sweep_val) - LIGHT_right - SONAR_right));

 }

 else if(ANGLE_shaft < ANGLE_mid_Shaft && ANGLE_prev > ANGLE_shaft) {

  ANGLE_front_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, ((ANGLE_mid_FLeft + ANGLE_sweep_val) - LIGHT_left - SONAR_left), ANGLE_mid_FLeft);

  ANGLE_front_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, ((ANGLE_mid_FRight + ANGLE_sweep_val) + LIGHT_right + SONAR_right), ANGLE_mid_FRight);

  ANGLE_back_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, ((ANGLE_mid_BLeft - ANGLE_sweep_val) + LIGHT_left + SONAR_left), ANGLE_mid_BLeft);

  ANGLE_back_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, ((ANGLE_mid_BRight - ANGLE_sweep_val) - LIGHT_right - SONAR_right), ANGLE_mid_BRight);

 }

 else if(ANGLE_shaft < ANGLE_mid_Shaft && ANGLE_prev < ANGLE_shaft) {

  ANGLE_front_left = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FLeft, ((ANGLE_mid_FLeft - ANGLE_sweep_val) + LIGHT_left + SONAR_left));

  ANGLE_front_right = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FRight, ((ANGLE_mid_FRight - ANGLE_sweep_val) - LIGHT_right - SONAR_right));

  ANGLE_back_left = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BLeft, ((ANGLE_mid_BLeft + ANGLE_sweep_val) - LIGHT_left - SONAR_left));

  ANGLE_back_right = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BRight, ((ANGLE_mid_BRight + ANGLE_sweep_val) + LIGHT_right + SONAR_right));

 }


 SERVO_front_left.writeMicroseconds(ANGLE_front_left);

 SERVO_front_right.writeMicroseconds(ANGLE_front_right);

 SERVO_back_left.writeMicroseconds(ANGLE_back_left);

 SERVO_back_right.writeMicroseconds(ANGLE_back_right);

}


/*==================================== END OF WALK ====================================*/


/*======================================== ROTATE ========================================*/
```

```
void ROTATE_LEFT() {
  while(flag_transition_rotate == 1) {
    TRANSITION_GAIT();
  }
  flag_transition_rotate = 2;

  while(flag_shaft_reverse == 2) {
    SHAFT_REVERSE();
    break;
  }
  flag_shaft_reverse = 3;

  while(flag_transition_start == 1) {
    SHAFT();
    LIGHT_left = 0;
    LIGHT_right = 0;
    TRANSITION_START();
    ROTATE();
  }

  SHAFT();
  LIGHT_left = 0;
  LIGHT_right = 0;
  ANGLE_sweep_val = ANGLE_sweep;
  ROTATE();
}

void ROTATE_RIGHT() {
  while(flag_transition_rotate == 1) {
    TRANSITION_GAIT();
  }
  flag_transition_rotate = 2;

  while(flag_shaft_reverse == 3) {
```

```
    SHAFT_REVERSE();
    break;
  }
  flag_shaft_reverse = 2;

  while(flag_transition_start == 1) {
    SHAFT();
    LIGHT_left = 0;
    LIGHT_right = 0;
    TRANSITION_START();
    ROTATE();
  }

  SHAFT();
  LIGHT_left = 0;
  LIGHT_right = 0;
  ANGLE_sweep_val = (ANGLE_sweep * -1);
  ROTATE();
}

void ROTATE() {
  if(ANGLE_shaft >= ANGLE_mid_Shaft && ANGLE_prev < ANGLE_shaft) {
    ANGLE_front_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, (ANGLE_mid_FLeft + ANGLE_sweep_val), ANGLE_mid_FLeft);
    ANGLE_front_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, (ANGLE_mid_FRight - ANGLE_sweep_val), ANGLE_mid_FRight);
    ANGLE_back_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, (ANGLE_mid_BLeft - ANGLE_sweep_val), ANGLE_mid_BLeft);
    ANGLE_back_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_max_Shaft, (ANGLE_mid_BRight + ANGLE_sweep_val), ANGLE_mid_BRight);
  }
  else if(ANGLE_shaft >= ANGLE_mid_Shaft && ANGLE_prev > ANGLE_shaft) {
    ANGLE_front_left = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FLeft, (ANGLE_mid_FLeft - ANGLE_sweep_val));
    ANGLE_front_right = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FRight, (ANGLE_mid_FRight + ANGLE_sweep_val));
    ANGLE_back_left = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BLeft, (ANGLE_mid_BLeft + ANGLE_sweep_val));
```

```
  ANGLE_back_right = map(ANGLE_shaft, ANGLE_max_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BRight, (ANGLE_mid_BRight - ANGLE_sweep_val));

 }

 else if(ANGLE_shaft < ANGLE_mid_Shaft && ANGLE_prev > ANGLE_shaft) {

  ANGLE_front_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, (ANGLE_mid_FLeft - ANGLE_sweep_val), ANGLE_mid_FLeft);

  ANGLE_front_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, (ANGLE_mid_FRight + ANGLE_sweep_val), ANGLE_mid_FRight);

  ANGLE_back_left = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, (ANGLE_mid_BLeft + ANGLE_sweep_val), ANGLE_mid_BLeft);

  ANGLE_back_right = map(ANGLE_shaft, ANGLE_mid_Shaft, ANGLE_min_Shaft, (ANGLE_mid_BRight - ANGLE_sweep_val), ANGLE_mid_BRight);

 }

 else if(ANGLE_shaft < ANGLE_mid_Shaft && ANGLE_prev < ANGLE_shaft) {

  ANGLE_front_left = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FLeft, (ANGLE_mid_FLeft + ANGLE_sweep_val));

  ANGLE_front_right = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_FRight, (ANGLE_mid_FRight - ANGLE_sweep_val));

  ANGLE_back_left = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BLeft, (ANGLE_mid_BLeft - ANGLE_sweep_val));

  ANGLE_back_right = map(ANGLE_shaft, ANGLE_min_Shaft, ANGLE_mid_Shaft, ANGLE_mid_BRight, (ANGLE_mid_BRight + ANGLE_sweep_val));

 }


 SERVO_front_left.writeMicroseconds(ANGLE_front_left);

 SERVO_front_right.writeMicroseconds(ANGLE_front_right);

 SERVO_back_left.writeMicroseconds(ANGLE_back_left);

 SERVO_back_right.writeMicroseconds(ANGLE_back_right);

}


/*==================================== END OF ROTATE ====================================*/


/*____________________________________________________________________________________________

/*##################################### END OF GAIT #####################################*/



/*================================= ULTRASONIC READING =================================*/


void SONAR_READ_ALL() {
```

```
  for(SONAR_id = 0; SONAR_id < SONAR_sum; SONAR_id++) {
    distance[SONAR_id] = int (SONAR_READ(SONAR_id));
  }
}

float SONAR_READ(int index) {
  float SONAR_distance;

  digitalWrite(PIN_trig[index], HIGH);
  delayMicroseconds(SONAR_TrigSig);
  digitalWrite(PIN_trig[index], LOW);

  float SONAR_EcInterval = float (pulseIn(PIN_ec[index], HIGH, SONAR_MaxEc));

  while(SONAR_EcInterval > 0.0) {
   SONAR_distance = SONAR_EcInterval * (SOUND_speed / 2.0);
   break;
  }
  while(SONAR_EcInterval == 0.0) {
   SONAR_distance = 501.0;
   break;
  }
  return SONAR_distance;
}

/*============================= END OF ULTRASONIC READING  =============================*/


/*==================================== LIGHT DETECT ====================================*/

void LIGHT_COMPARE_EXECUTE() {
  //PHOTO_FLeft_RAW = analogRead(PIN_PHOTO[FRONT_LEFT]);
  //PHOTO_FRight_RAW = analogRead(PIN_PHOTO[FRONT_RIGHT]);
  PHOTO_Front_Left = analogRead(PIN_PHOTO[FRONT_LEFT]);
  PHOTO_Front_Right = analogRead(PIN_PHOTO[FRONT_RIGHT]);
```

```
PHOTO_Back_Left = analogRead(PIN_PHOTO[BACK_LEFT]);
PHOTO_Back_Right = analogRead(PIN_PHOTO[BACK_RIGHT]);

if((PHOTO_Front_Left + PHOTO_Front_Right) >= (PHOTO_Back_Left + PHOTO_Back_Right)) {
  int LIGHT_Sensitivity = 50;
  if(LIGHT_COMPARE() > LIGHT_Sensitivity) {
    LIGHT_left = LIGHT_COMPARE();
    LIGHT_right = 0;
  }
  else if(LIGHT_COMPARE() < -LIGHT_Sensitivity) {
    LIGHT_left = 0;
    LIGHT_right = LIGHT_COMPARE();
  }
  else {
    LIGHT_left = 0;
    LIGHT_right = 0;
  }
}
else {
  if(PHOTO_Back_Left > PHOTO_Back_Right) {
    LIGHT_right = 0;
    LIGHT_left = ANGLE_turnMAX;
  }
  else if(PHOTO_Back_Left < PHOTO_Back_Right) {
    LIGHT_right = -ANGLE_turnMAX;
    LIGHT_left = 0;
  }
  else {
    int RANDOM_back_light = random(0,2);

    if(RANDOM_back_light == 0) {
      LIGHT_right = 0;
      LIGHT_left = ANGLE_turnMAX;
    }
    else if(RANDOM_back_light == 1) {
```

```
      LIGHT_right = -ANGLE_turnMAX;
      LIGHT_left = 0;
     }
    }
  }
}

int LIGHT_COMPARE() {
  int LIGHT_rate;
  if(PHOTO_Front_Left > PHOTO_Front_Right) {
    LIGHT_rate = PHOTO_Front_Left;
  }
  else if(PHOTO_Front_Right > PHOTO_Front_Left) {
    LIGHT_rate = PHOTO_Front_Right;
  }
  else {
    // choose to use one and comments the other of these variables bellow
    // LIGHT_rate = PHOTO_Front_Left;
    LIGHT_rate = PHOTO_Front_Right;
  }

  int LIGHT_compareRAW = PHOTO_Front_Left - PHOTO_Front_Right;
  LIGHT_compareRAW = map(LIGHT_compareRAW, -LIGHT_rate, LIGHT_rate, -ANGLE_turnMAX, ANGLE_turnMAX);;

  return LIGHT_compareRAW;
}

/*================================= END OF LIGHT DETECT ================================*/


/*===================================== BEHAVIOUR =====================================*/

void RETREAT_AVOID() {
  counter_gait = 0;
  while(counter_gait <= counter_gait_max) {
```

```
    RETREAT();
  }
}

void ROTATE_LEFT_AVOID() {
  counter_gait = 0;
  rotate_random = 2;
  while(counter_gait <= counter_gait_max) {
    ROTATE_LEFT();
  }
}

void ROTATE_RIGHT_AVOID() {
  counter_gait = 0;
  rotate_random = 2;
  while(counter_gait <= counter_gait_max) {
    ROTATE_RIGHT();
  }
}

void ROTATE_RANDOM_AVOID() {
  rotate_random = ROTATE_RANDOM();
  while(rotate_random == 0) {
    ROTATE_LEFT_AVOID();
  }
  while(rotate_random == 1) {
    ROTATE_RIGHT_AVOID();
  }
}

void SIDE_AVOID() {
  if(distance[LEFT] <= distTurn && distance[RIGHT] > distTurn) {
    LIGHT_left = 0;
    LIGHT_right = 0;
```

```
    SONAR_left = 0;
    SONAR_right = -(map(distance[LEFT], 0, distTurn, ANGLE_turnMAX, 0));
  }
  else if(distance[RIGHT] <= distTurn && distance[LEFT] > distTurn) {
    LIGHT_left = 0;
    LIGHT_right = 0;

    SONAR_right = 0;
    SONAR_left = map(distance[RIGHT], 0, distTurn, ANGLE_turnMAX, 0);
  }
  else if(distance[LEFT] <= distTurn && distance[RIGHT] <= distTurn) {
    LIGHT_left = 0;
    LIGHT_right = 0;

    if(distance[LEFT] < distance[RIGHT]) {
      SONAR_left = 0;
      SONAR_right = -(map(distance[LEFT], 0, distTurn, ANGLE_turnNARROW, 0));
    }
    else if(distance[RIGHT] < distance[LEFT]) {
      SONAR_right = 0;
      SONAR_left = map(distance[RIGHT], 0, distTurn, ANGLE_turnNARROW, 0);
    }
  }
  else {
    SONAR_right = 0;
    SONAR_left = 0;
  }
}

/*================================== END OF BEHAVIOUR ==================================*/
```