RTES TERM PROJECT

GROUP-4 (TEAM-5)

Estimate distance measurement using

NodeMCU ESP8266 based on

RSSI technique(Arduino)

BY

Rajath Koratagere Manjunath (rkm350)
Abhinav Kannojia(ak7357)

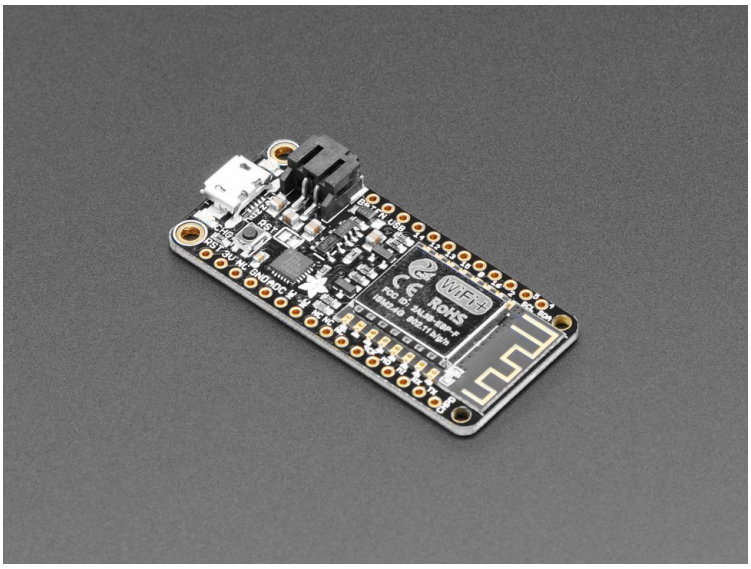
CONTENTS

1. Abstract

The use of WiFi is now a part of each human life. Airports, railways, bus-stand, home, markets everywhere now people uses WiFi because its reliability and low-cost. WiFi is also applicable in future tech of IoT (Internet of Things). In this work we used two NodeMCU (ESP8266 WiFi module) which is a easily programmable. NodeMCU which acts itself as a sensor node can be used as Access Point (AP) or as a STAtion (STA). We used one node as a AP and another as a STA. To locate a device distance measurement one of most important issue. There are lots of technique to find out the distance between two nodes (e.g. Time of Arrival (TOA), Time Difference of Arrival (TDOA) or Received Signal Strength (RSS) algorithms etc). In this work we use RSSI technique to determine the distance. First, we take around 300 sample data (RSSI values) and find the the standard deviation to calculate how much the RSSI values are spread out and use curve fitting technique to find suitable equation for estimate distance. Then, we compared the estimated distance with actual distance to find the error level in percentage. We are success to reduce the average error level up to 8.32%.

2. Introduction

Heart of this bot is an ESP8266 WiFi microcontroller clocked at 80 MHz and at 3.3V logic. This microcontroller contains a Tensilica chip core as well as a full WiFi stack. We have programmed the



microcontroller using the Arduino IDE. This is wired up a high-quality SiLabs CP2104 USB-Serial chip that can upload code at a blistering 921600 baud for fast development time. It also has auto-reset so no noodling with pins and reset button pressings. The CP2104 has better driver support than the CH340 and can do very high speeds without stability issues.

To make it easy to use for portable projects, it has been added up with a connector for any of our 3.7V Lithium polymer batteries and built in battery charging.

We are using 4 pin HCSR04 sensors for object detection in this project. **HC-SR04 Ultrasonic (US) sensor** is a 4 pin module, whose pin names are Vcc, Trigger, Echo and Ground respectively. This sensor is a very popular sensor used in many applications where measuring distance or sensing objects are required. The module has two eyes like projects in the front which forms the Ultrasonic transmitter and Receiver. The sensor works with the simple high school formula that

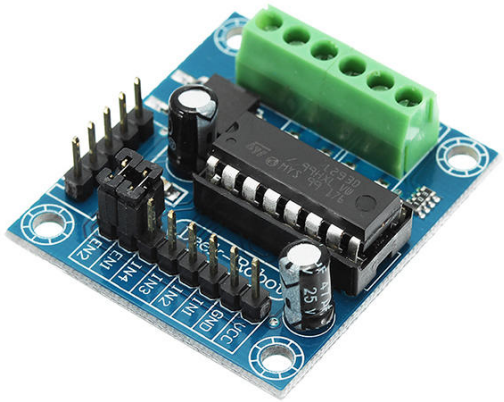
Distance = Speed × Time

The Ultrasonic transmitter transmits an ultrasonic wave, this wave travels in air and when it gets objected by any material it gets reflected back toward the sensor this reflected wave is observed by the Ultrasonic receiver module

In total we have used three of these ultrasonic sensors for detecting obstacles in front, left and right for adding accuracy to our bot. Basic intention was to traverse through a Maze of obstacles through 3 way lookup mechanism which tracks obstacles on all three sensors and responds the distance calculated by sending an echo signal.
Code snippet below shows the function for basic pin setup on our bot which helps in detecting obstacles for all three sensors.

```
bool detectObstacles() {
  // Detect the front obstacles.

  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  frontDistance = (pulseIn(echoPinCenter, HIGH) / 2) / 74;
  delay(10);

  // Detect the left obstacles.
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  leftDistance = (pulseIn(echoPinLeft, HIGH) / 2) / 74;
  delay(10);

  // Detect the right obstacles.
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  rightDistance = (pulseIn(echoPinRight, HIGH) / 2) / 74;

  delay(10);

  if ((frontDistance < 5) or (leftDistance < 5) or (rightDistance < 5)) {

    return true;

  }
  return false;
}
```

For motor control we are using the L293D motor controller. The L293D is a 16-pin Motor Driver IC which can control up to two DC motors simultaneously, in any direction.



Following code snippet explains, how we are controlling four basic control movements of our bot using L293D motor controller.

```
void moveForward() {

  digitalWrite(left_positive, HIGH);
  digitalWrite(left_negetive, LOW);
  digitalWrite(right_positive, LOW);
  digitalWrite(right_negetive, HIGH);
  analogWrite(pwm2, speed2);

}

void moveLeft() {

  digitalWrite(left_positive, HIGH);
  digitalWrite(left_negetive, LOW);
  digitalWrite(right_positive, HIGH);
  digitalWrite(right_negetive, LOW);
  analogWrite(pwm2, speed2);

}


void moveRight() {
  digitalWrite(left_positive, LOW);
  digitalWrite(left_negetive, HIGH);
  digitalWrite(right_positive, LOW);
  digitalWrite(right_negetive, HIGH);
  analogWrite(pwm2, speed2);

}

void stopMove() {

  digitalWrite(left_positive, LOW);
  digitalWrite(left_negetive, LOW);
  digitalWrite(right_positive, LOW);
  digitalWrite(right_negetive, LOW);

}
```

3. Algorithm Implementation

In actual arduino implementation of our bot, we created a function named rssiCalculate() for taking a moving point average of rssi values to stabilize the varying WiFi signals to obtain reliable WiFi strength. Below code snippet shows the implementation of this function.

```
float rssiCalculate() {
  //  Slide the moving average window to the left and then add the current RSSI to the final window.
  float rssiAve = 0;
  for (int j = 0; j < 500; j++) {

    for (int i = 0; i < 500; i++) {
      rssiArr[i] = rssiArr[(i + 1) % 500];
    }
    rssiArr[499] = WiFi.RSSI();


    // Calculate the average RSSI strength

  }
  for (int i = 0; i < 500; i++) {
    rssiAve = rssiAve + rssiArr[i];
    delay(3);
  }
  rssiAve = rssiAve / 500;

  return rssiAve;
}
```

When our bot starts we are trying to check signal strength by checking **aveRssi** i.e average rssi value and move in the direction of a larger RSSI value. We notice that taking a 500 point average guarantees that the value average does not vary more than 2 dBm at a particular point. Hence if the robot moves in the opposite direction, it turns and makes an obtuse angle. This guarantees that it moves inside the limit cycle which guarantees asymptotic convergence. We noticed that the RSSI values do not vary considerably after the RSSI decreases less than -56 dBm. So if the RSSI values are less than -56 for two consecutive iterations, it makes a turn at an angle greater than 90 degrees. The reasoning of this is similar that the bot is guaranteed to move to closer to the beacon.

```
void loop() {
  if (iterationCount == 0) {
    float aveRssi = rssiCalculate();
    Serial.println(aveRssi);
    Serial.println(previousRssi);
    if (aveRssi < -56 and maxLengthCount == 2) {
      moveLeft();
      delay(750);
      maxLengthCount = 0;
    }
    else if (aveRssi < -56) {
      maxLengthCount++;
    }
    else {
      maxLengthCount = 0;
    }
```

```
    if (aveRssi > -43) {
      exit(0);
    }
    if (aveRssi > previousRssi) {
      previousRssi = aveRssi;
    }
    else if (aveRssi < previousRssi - 5) {
      moveLeft();
      delay(750);
    }
    else{
      moveForward();
      delay(500);
    }
  }
  else {
    bool obstacle = detectObstacles();
    if (obstacle) {
      if (obstacleCount == 0) {
        moveLeft();
        delay(500);
      }
      else if (obstacleCount == 1) {
        moveRight();
        delay(500);

      }
      else if(obstacleCount >1) {
        moveRight();
        delay(500);
      }
      obstacleCount++;
    }
    else {

      obstacleCount = 0;
      moveForward();
      delay(500);

    }
  }
  stopMove();
  delay(500);
  iterationCount = (iterationCount + 1) % 5;
}
```

It is continuously checking for obstacles and avoiding them. A safe distance is maintained from the obstacles so as to not collide while making decisions to move towards the beacon. The obstacles are checked for more often as checking the RSSI strength every iteration is slower and would considerably slow down the convergence. The algorithm has been tested multiple times to verify convergence and has succeeded every test along with the obstacle avoidance to converge to the beacon. The average time taken by the Robot to reach the beacon is 7 minutes and varies with the initial positions and obstacles.

Care has to be taken so as to connect only one device to the beacon as connecting multiple devices changes the RSSI signals to all the devices connected. As the values in the snippet are for just one device, they have to be changed in case of connecting multiple devices.