Technical Design Manual

A GUI for controlling and supervising multiple robots remotely

Version 1.0
30th November 2019

Prepared by

- Jaliyagoda A.J.N.M.
- Tennakoon T.M.P.B.
- Karunarathna S.D.D.D

Table of Contents

Hardware Information

ESP32 Development Board

ESP32 is a microcontroller development board that is a product of Espressif, which is developed for the applications of IoT and wearable electronics.

It contains inbuilt Wi-Fi and Bluetooth communications modules with UART, SPI, I2C, I2S and CAN bus communication interfaces

Product Documentation: https://docs.espressif.com/projects/esp-idf/en/latest/i

Features

1. CPU and Memory
   - Xtensa® single-/dual-core 32-bit LX6 microprocessor(s), up to 600 MIPS
   - 448 KB ROM
   - 520 KB SRAM
   - 16 KB SRAM in RTC
   - QSPI supports multiple flash/SRAM chips

2. Clocks and Timers
   - Internal 8 MHz oscillator with calibration
   - Internal RC oscillator with calibration
   - External 2 MHz ~ 60 MHz crystal oscillator (40 MHz only for Wi-Fi/BT functionality)
   - External 32 kHz crystal oscillator for RTC with calibration
   - Two timer groups, including 2 x 64-bit timers and 1 x main watchdog in each group
   - One RTC timer
   - RTC watchdog

3. Advanced Peripheral Interfaces
   - 34 × programmable GPIOs
   - 12-bit SAR ADC up to 18 channels
   - 2 × 8-bit DAC
   - 10 × touch sensors
   - 4 × SPI
   - 2 × I2S
   - 2 × I2C
   - 3 × UART
   - 1 host (SD/eMMC/SDIO)
   - 1 slave (SDIO/SPI)
   - Ethernet MAC interface with dedicated DMA and IEEE 1588 support
   - CAN 2.0
   - IR (TX/RX)

- Motor PWM
- LED PWM up to 16 channels
- Hall sensor

4. WiFi
- Support Station-only mode, AP-only mode, Station/AP-coexistence mode
- Support IEEE-802.11B, IEEE-802.11G, IEEE802.11N and APIs to configure the protocol mode
- Support WPA/WPA2/WPA2-Enterprise and WPS

5. Bluetooth
- Compliant with Bluetooth v4.2 BR/EDR and BLE specifications
- Class-1, class-2 and class-3 transmitter without external power amplifier
- Multi-connections in Classic BT and BLE

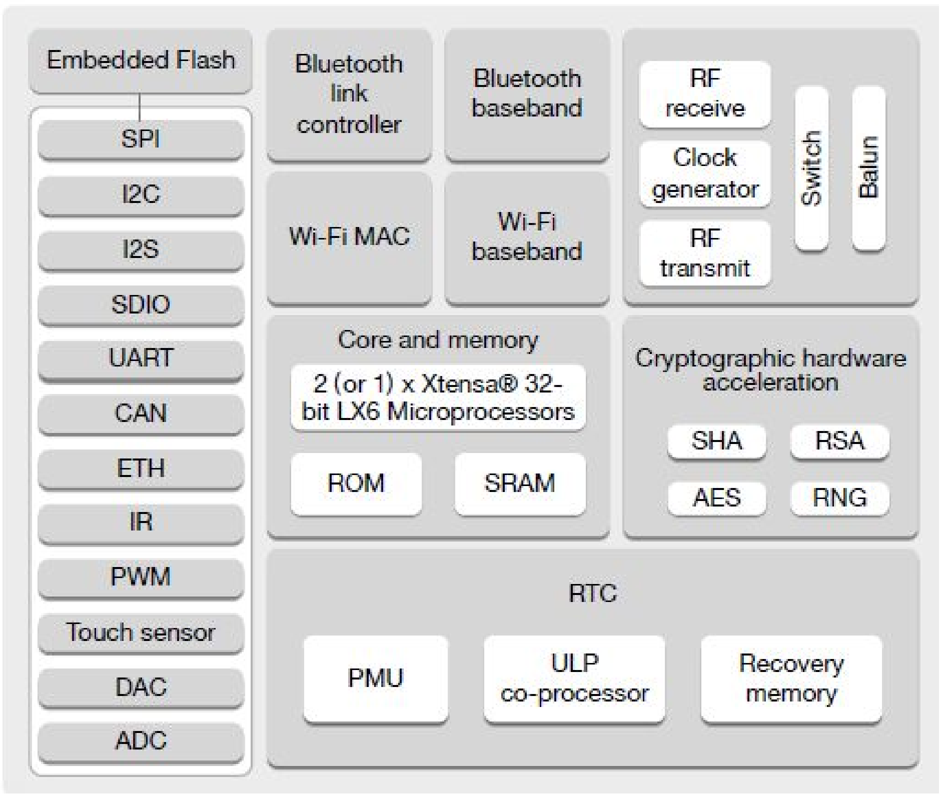


Figure: Functional Block Diagram

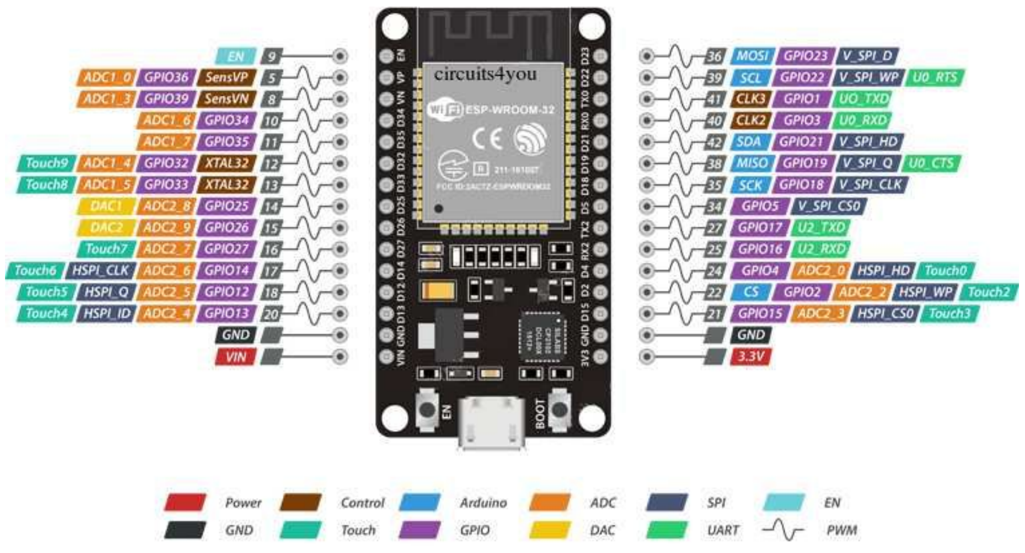


Figure: ESP32 Dev Board Pinout

Motors

The robot using specially modified pair of Servo motors (**SG90**) and need to configure the degree of the servo that stops the motions in define.h file, on the first time of use.

```
#define RIGHT_DEFAULT 93
#define LEFT_DEFAULT 90
```

Elementary Functions

void motorSetup()	Setup motors
void motorWrite(uint8_t left, uint8_t right)	Write the directions and turn on motors
void motorStop()	Stop the motors
void forward(float distance)	Move forward the robot by given distance
void reverse(float distance)	Move backward the robot by given distance

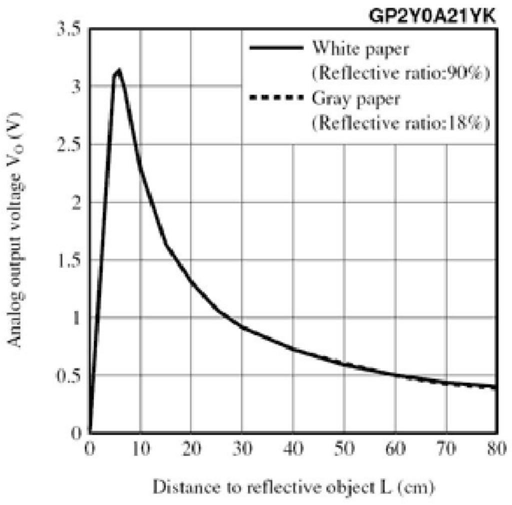
Functions works with the GY-511 module

(Those functions works only when the Compass is enabled)

void turn(float angle)	Angle in degrees, [-180, 180]
void turnLeft(float angle)	Angle in degrees [0, 180]
void turnRight(float angle)	Angle in degrees [0, 180]

Proximity sensor providing Analog reading which is inversely proportional to the distance in some range.

Robot using the module, GP2Y0A21YK0F as the proximity sensor and it can measure the distance between 10cm - 80cm (Please note that sensor may give wrong readings for distances less than 10cm)



Elementary Functions:

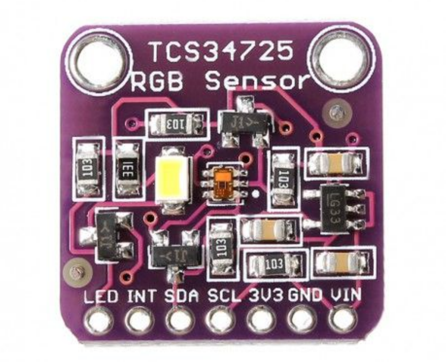
void beginDistanceSensor()	Setup the distance sensor
float getDistance()	Get the distance as a float
void distanceSensorTest()	Test / Debug sensor functionalities over Serial Monitor

Color Sensor

Robot using **TCS34725** module as the color sensor. It is connected to the main microcontroller via the I2C bus, and its' I2C address is **0x29**. Default Gain is x4 and integration is 700ms.

Here the LED which is connected to the sensor is controlled by the Interrupt pin which was available in the module.

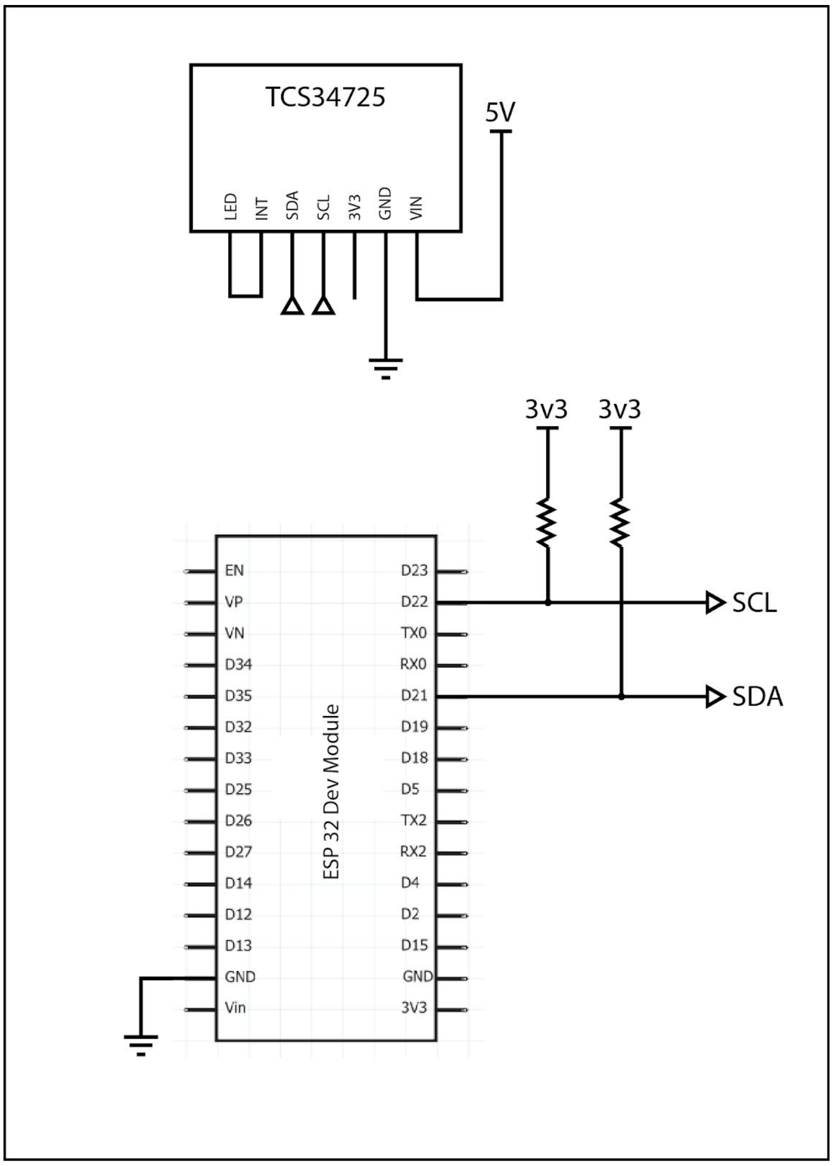
Other than reading R, G, B values of the environment, it can measure Color Intensity and LUX (Lumens per square meter) also.



Elementary Functions:

void beginColorSensor()	Setup the Color Sensor
void generateGammaTable()	Prepare a Gamma Table for simulate RGB colors
void colorSensorTest()	Test / Debug sensor functionalities over Serial Monitor
uint8_t getColorCode()	Returns the color [None=0, Red=1, Green=2, Blue=3]

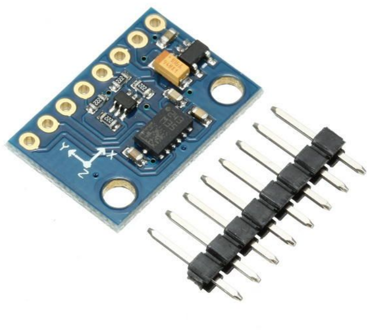
Schematics Diagram

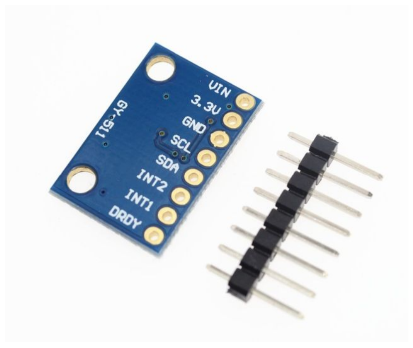


Compass & Accelerometer

Robot using the **GY-511** module, which is based on LSM303. It is a 6DOF sensor which can provide Acceleration and Magnetometer readings on x, y, z-axis.

The module is connected to the main microcontroller via the I2C bus, and the I2C address of Accelerometer is **0x19** and the I2C address of the magnetometer is **0x1E**.

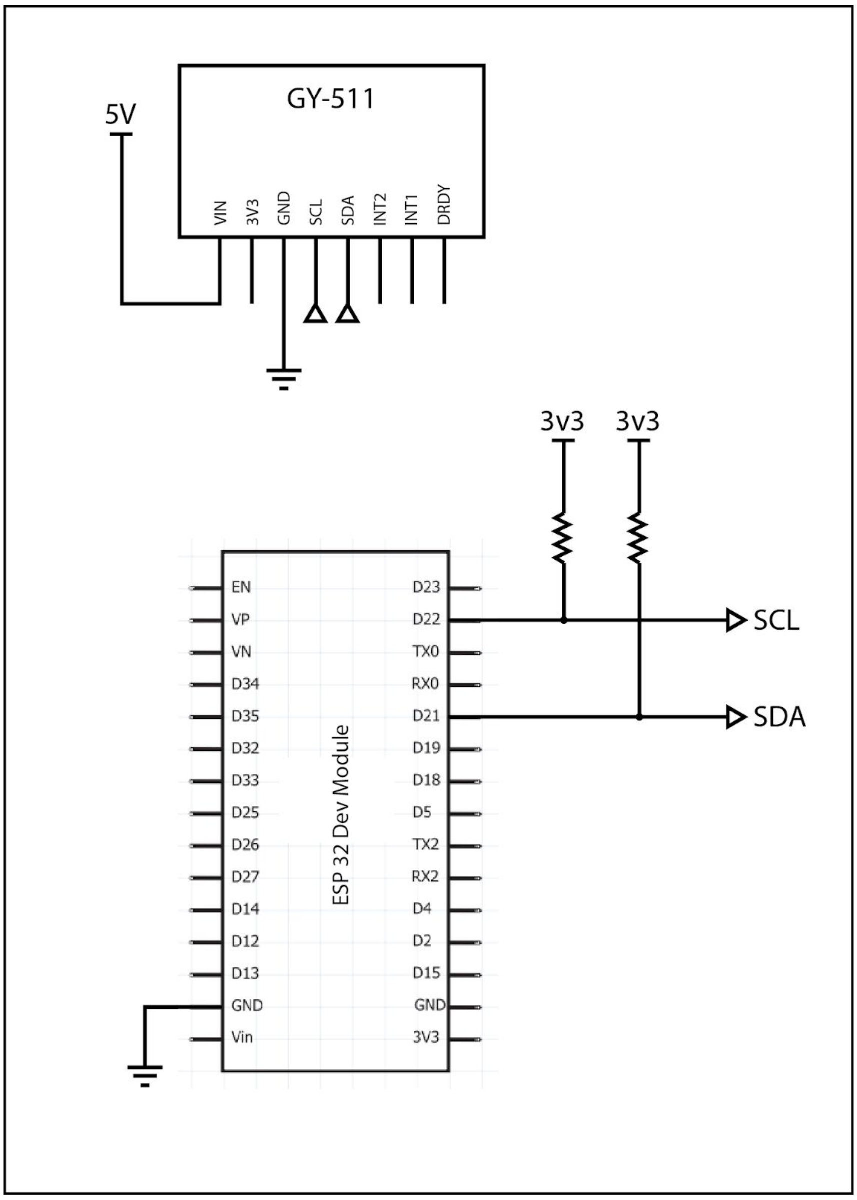




Elementary Functions:

void beginMagSensor()	Setup the Magnetometer sensor
void beginAccelSensor()	Setup the Accelerometer Sensor
float getAngle()	Get the heading direction in degrees
void magSensorTest()	Test magnetometer functionalities via Serial Monitor
void accelSensorTest()	Test accelerometer functionalities via Serial Monitor
void displayMagSensorDetails()	Display ranges and other related information about the magnetometer on Serial Monitor
void displayAccelSensorDetails()	Display ranges and other related information about the compass on Serial Monitor

Schematics Diagram



IR Transmitter and Receiver

IR Tx & Rx unit contains 6 Separate IR Receivers (**1338B**) and 1 IR Transmitter channel with 6 x IR Emitter LEDs. Can transmit a maximum of a 32-bit long binary signal without ECC or 27bit binary signals with two-bit error detection and one-bit error correction (5 parity bits are using with *Hamming Code* implement)

When a signal received, the function, '*parseRmtData*' will be called with data, length and the sensor ID as an interrupt.

Elementary Functions:

void beginInfared()	Setup IR Transmitters and Receiver
void sendWaveform(unsigned int value, int len)	Send a given binary signal
void sendTestSignal()	Seds a random binary signal
unsigned int encodedValue(unsigned int value)	Hamming code encoding
unsigned int decodedValue(unsigned int value)	Hamming code decoding
void parseRmtData(rmt_data_t* items, size_t len, uint8_t rId)	Receive IR Data (interrupt function)

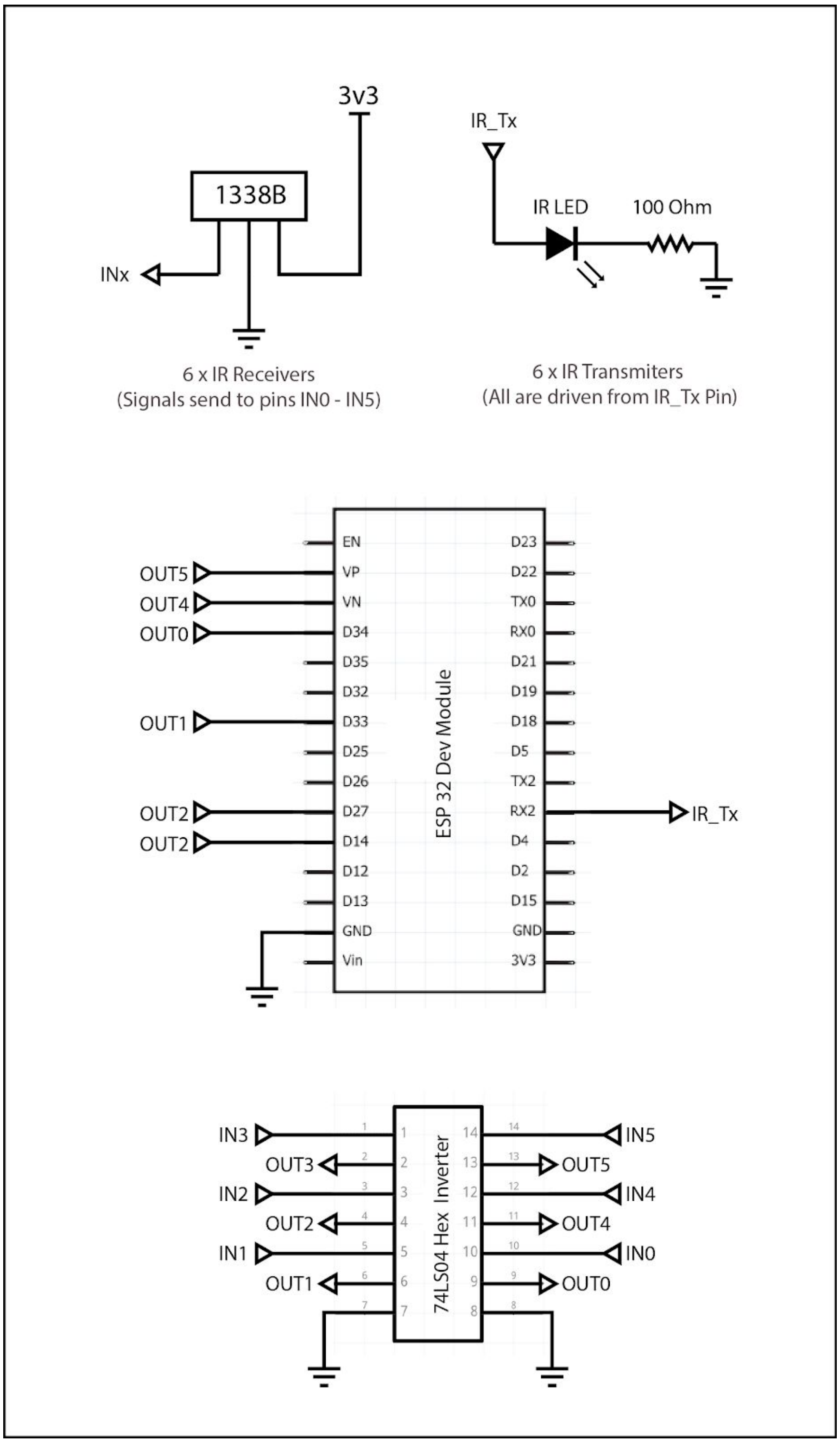
Hardware level private functions

void sendBit(short pulseHigh, short pulseLow)	Generate the waveform and transmit
unsigned int charToInt(char* bitArray, int len)	Convert an int to char array of it’s binaries

Interrupt Functions

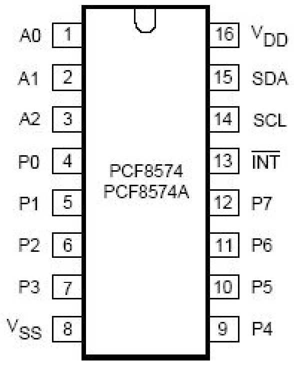
extern "C" void irReceive_0(uint32_t *data, size_t len)	
extern "C" void irReceive_1(uint32_t *data, size_t len)	
extern "C" void irReceive_2(uint32_t *data, size_t len)	
extern "C" void irReceive_3(uint32_t *data, size_t len)	
extern "C" void irReceive_4(uint32_t *data, size_t len)	
extern "C" void irReceive_5(uint32_t *data, size_t len)	

Schematics Diagram



I2C Port Expander

The robot comes with an I2C Port Expander module, based on **PCF8574 IC**, which gives an additional 8 I/O pins, can be controlled using the I2C bus. I2C Address of the unit is **0x3F**.



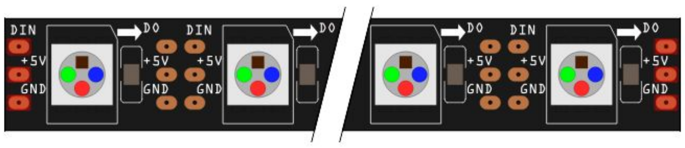
Elementary Functions:

void portsBegin()	Setup PCF
void expander.pinMode(pin_num, pinMode);	Define the pin mode
void expander.digitalWrite(pin_num, value);	Write a digital value
int expander.digitalRead(pin_num)	Read the pin status

Pixel LED Strip

The robot comes with 20 x RGB LEDs which can be individually addressed and can show any color, which is a combination of 8bit RGB values.

WX2812 is the name of the module of the LEDs, which are in the form factor of 5050



Elementary Functions:

void ledBegin()	Setup the LED Strip
void neoPixelTest()	Run a test script
void pixelShowColor(int r, int g, int b)	Show the given RGB value on the whole strip
void pixelColorWave(int r, int g, int b)	Show a wave of the given color
void pixelOff()	Off all the LEDs on the strip

.

WiFi Monitor

When the WiFi monitor was enabled, Robot and its some features can be controlled by Web Interface using provided APIs. Here the robot works as a Server and replying to /GET requests received into it's IP.

Elementary Functions:

void beginWiFiMonitor()	Setup WiFi Station and provide listeners.
void handleRoot()	What should be done when the root directory was called
void handleNotFound()	What should be done when a non existing directory was called
void robotStatus()	Reply the details of the robot
void getAngleToMonitor()	Reply the current heading direction
void getColorToMonitor()	Reply the color [None, R, G, B]
void getDistanceToMonitor()	Reply to the proximity sensor reading

API Level Functions:

void getMag()	Reply the Magnetometer readings (JSON)
void getAccel()	Reply the Compass readings (JSON)
void getDist()	Reply the Proximity sensor readings (JSON)
void getColor()	Reply the color sensor readings (JSON)
void handleMotion()	Write Forward, backward, rotation, stop status to the motors depending on the input args
void handleRotation()	Handle CW or CCW rotation according to the input args
void handlePixelLED()	Handle strip colors according to the input args

API Guide:

http://***{Robot IP}***/

http://***{Robot IP}***/status
    Response: {“status”:“”,“version”:””,“IP”:“”}

http://***{Robot IP}***/mag
    Response: {“x”:””,“y”:””,“z”:””,“heading”:””}

http://***{Robot IP}***/accel
    Response: {“x”:””,“y”:””,“z”:””}

http://***{Robot IP}***/dist
    Response: {“raw”:””,“filtered”:””}

http://***{Robot IP}***/color
    Response: {“R”:””,“G”:””,“B”:””,“temp”:””,“lux”:””,}

http://***{Robot IP}***/motor?dir=**{DIR}**
    DIR = [forward, backward, rotCW, rotCCW, stop]
    Response: { “status”: ””}

http://***{Robot IP}***/turn?ang=**{Angle}**
    Angle = [-180, 180]
    Response: { “status”: ””}

http://***{Robot IP}***/pixelLED/all?R=**{$1}**&G=**{$2}**&B=**{$3}**
    $1 = Red Value [0, 255]
    $2 = Green Value [0, 255]
    $3 = Blue Value [0, 255]
    Response: { “status”: ””}

OTA Uploading

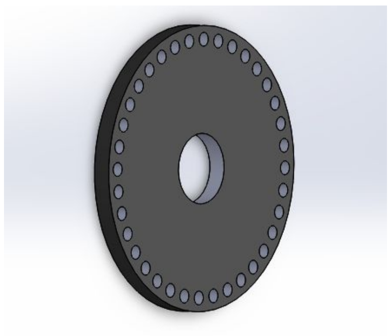
OTA upload (On the air upload) is a special feature provided with WiFi that we can upload a program into the robot through the WiFi network.

Options available to control the hostname. We can set a password for OTA upload, after that, only who has the correct password can upload the codes into the robot.

Optical Encoder Unit

Encoder Unit is specially designed for this robot, which will suit in size and shape. Encoder wheels help the robot to go in forward and backward by a specific distance. The theoretical accuracy of the encoder wheels is limited to 2mm.

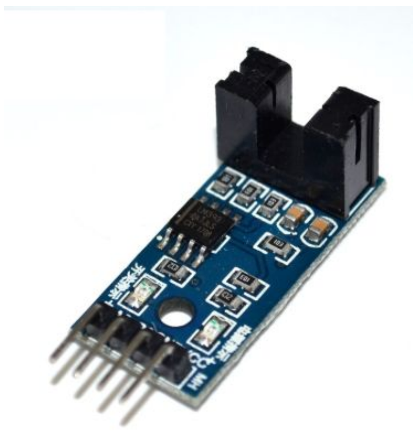
Encoder wheel system has two units. First one is the wheel with 36 holes in its perimeter, which helps the encoder sensor to identify the rotation.



Robot wheel as an optical Encoder Disk

The second part of the Encoder Unit is the encoder sensor. It is a combination of IR Transmitter LED, IR Optical sensor and a comparator circuit with a low pass filter circuit. The module can output both analog and digital output of the IR optical sensor reading. The digital value is obtained by applying it to the comparator circuit and compared with predefined value. In this design, only the digital output is used.

The angle of rotation of the wheel or the distance the robot traveled is considered as the number of the rising edge detected by the optical encoder sensor. The microcontroller is programmed with its inbuild pulse counter unit to capture these digital pulses.



Optical Encoder Counter Module

Compatibility and Performance measurements

Performance Test

At the performance side, we had to focus on both the performance of the robots and the performance of the web platform for the GUI. As for the robots we have the following areas that

we need to focus on,

- Taking turns
- Speed
- Calibration of compass sensor readings
- Calibration of color sensors

And for the web platform

- Communication delays
- Accessibility

Test 1

Objective:

Identifying the best performing speed for the robots to take turns.

Methodology:

Test robots for different speeds and take sensor readings at each speed. Tabulate the results and identify the most suitable speed for robots.

Hardware requirements:

- LSM303 Compass Sensor
- Robot

Procedure:

1. Connect the compass module and the IR sensors.
2. Place a guiding sheet with all the angles marked for a compass.
3. Upload a script to the robot that takes 90-degree turns.
4. Place the robot on the guiding sheet.
5. Apply minimum possible speed for the robot.
6. After taking the turn to calculate the error.

7. Increase the speed and repeat.

Dependencies:

- Compass reading depends on geolocation.
- IR sensor readings depend on ambient light.

Assumptions:

- Environment changes do not affect compass readings.

Observations:

Speed	Angle (Degrees)	Error (Degrees)
(2,2)	91.6	1.6
(4,4)	92.4	2.4
(6,6)	94.8	4.8
(8,8)	101.7	11.7
(10,10)	99.8	9.8
(12,12)	104.2	14.2
(14,14)	105.3	15.3
(16,16)	102.1	12.1

Conclusion:

Lower speed allows the robot to take angles accurately. With increasing speeds the accuracy of angles decreases. However, under lower speeds controlling the servos becomes difficult since they can work at different speeds to the same input. Also, we need to increase the speed as much as possible to finish the task quickly as it can. Therefore a speed like (6,6) or (4,4) is suitable for the speed of the robots when taking turns.

Unit Test

For unit testing, we need to isolate every single sensor and actuator. We need to test them individually to identify the accuracy and limitations of the particular unit. Mainly there are 4 components that we need to test.

- Servo Motors
- LSM303 Compass Sensor
- IR Proximity Sensor (Distance Sensor)
- TCS34725 Color Sensor

Test 2

Objective:

Identifying the scope of the IR sharp sensor; minimum and maximum distances the sensor can identify, and the variation of the reading against the distance.

Methodology:

Place an object at different distances from the robot and take distance sensor readings.

Hardware requirements:

- GP2Y0A21YK0F IR Proximity Sensor
- A flat object which has a white surface
- The robot

Environment Requirements:

- Closed arena for robots to move (Covered from ambient light)

Procedure:

1. Connect the distance sensor to the robot.
2. Place robot in the arena.
3. Place the object 2 cm away from the robot. Take sensor reading.
4. Increase the distance 2cm at a time and take sensor reading.
5. Take readings up to 10 cms.
6. Increase the distance by 10 cms.
7. Take reading up to 100 cms.

Dependencies:

- IR sensor readings depend on ambient light.
- Readings depend on the surface and color of the object.

Assumptions:

- No ambient light in the arena.
- The object surface is smooth.

Observations:

Distance (cm)	IR Sharp sensor
2	10
4	10
6	10
8	10
10	12
20	18
30	27
40	38
50	44
60	47
70	46
80	47
90	48
100	47

Conclusion:

The IR Sharp sensor which we used isn't capable of handling objects that are less than 10 cms away from the sensor. Therefore it outputs the minimum value it can represent to those objects which are 10. However, from 10 cms to 40cm Proximity sensor can identify the distance of objects with reasonable accuracy. From 50cms and upwards the sensor cannot identify because the object is too small to detect from 50cm away.

Load/Stress Test

Test 3

Objective:

Test how many devices that are in the same environment (same room) can communicate without signal interference and test whether the server is able to handle that much of clients without slowing down.

Methodology:

Connect devices to the platform one by one and identify the limitation on the number of devices that can be connected to the WiFi network.

Hardware requirements:

- Up to 10 ESP32 boards
- Centralized Server

Environment Requirements:

- Closed room

Software Requirements:

- NodeJS server to handle HTTP requests.

Procedure:

1. Upload Client-side script to each ESP32 board
2. Choose a single board and try to send an HTTP GET request to the server simultaneously.
3. Observe the behavior of the server and identify whether the requests are successful or not.
4. Add another device and repeat steps 2 and 3.
5. Test up to 7 devices.
6. Try to respond to get request and get the results as well.

Observations:

No of Boards	Board Receiving Requests		Board sending Requests	
	Request	Response	Request	Response
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	1	1	1	1

1 - Successful request or response
0 - Unsuccessful request or response

Conclusion:

The server can handle up to 7 boards without slowing down.

Algorithms and Data Structures

Collision free path calculating algorithm

Given the source and destination coordinates of two objects, the algorithm is capable of calculating the collision-free path for the two objects. However, since predicting the time a collision may happen is too difficult to handle cause the hardware imperfectness (Two robots may move at different speeds) the second object starts to move only after the first one arrived at the respective destination. Then the algorithms need to create two paths where the first one needs to calculate a path where it avoids the source cell of the second robot and when calculating the path of the second robot it needs to avoid the destination cell of the first robot.

The Algorithm in JavaScript

```
function calculatePath(src0, dest0, src1, dest1) {

      //Calculating source and destination rows and columns
      var src0_row = Math.floor(src0/16);
      var src0_col = Math.floor(src0%16);

      var src1_row = Math.floor(src1/16);
      var src1_col = Math.floor(src1%16);

      var dest0_row = Math.floor(dest0/16);
      var dest0_col = Math.floor(dest0%16);

      var dest1_row = Math.floor(dest1/16);
      var dest1_col = Math.floor(dest1%16);


      var path0 = [];
      var path0_data = [];
      var path1 = [];
      var path1_data = [];

      //Calculating path 0
      var factor = -1;
      if(src0_row <= dest0_row){
        factor = 1;
      }

      while(src0_row != dest0_row){
        if(factor==1)
          path0.push('U');
```

```
  else
    path0.push('D');
  src0_row = src0_row+(factor);
  path0_data.push(src0_row*16+src0_col);


//Detect if an object lies on the column path and if avoid that
  if(path0_data[path0_data.length-1] == src1) {
    path0_data.pop();
    path0.pop();
    if(src0_col != 15) {
      path0.push('R');
      path0.push('U');
      path0_data.push((src0_row-1)*16 + src0_col+1);
      path0_data.push((src0_row)*16 + src0_col+1);
      src0_col += 1;
    }
    else {
      path0.push('L');
      path0.push('U');
      path0_data.push((src0_row-1)*16 + src0_col-1);
      path0_data.push(src0_row*16 + src0_col-1);
      src0_col -= 1;
    }
  }
}

factor = -1;
if(src0_col <= dest0_col){
  factor = 1;
}
while(src0_col != dest0_col){
  if(factor==1)
    path0.push('R');
  else
    path0.push('L');
  src0_col = src0_col+(factor);
  path0_data.push(src0_row*16+src0_col);
}

factor = -1;

//Calculating path 1
if(src1_row <= dest1_row){
  factor = 1;
}

while(src1_row != dest1_row) {
  if(factor==1)
    path1.push('U');
  else
    path1.push('D');
  src1_row = src1_row+(factor);
  path1_data.push(src1_row*16+src1_col);



  //Detect if an object is on the column path if so avoid that
  if(path1_data[path1_data.length-1] == dest0) {
    path1_data.pop();
    path1.pop();
    if(src1_col != 15) {
      path1.push('R');
      path1.push('U');
      path1_data.push((src1_row-1)*16 + src1_col+1);
      path1_data.push((src1_row)*16 + src1_col+1);
      src1_col += 1;
    }
    else {
      path1.push('L');
      path1.push('U');
      path1_data.push((src1_row-1)*16 + src1_col-1);
      path1_data.push(src1_row*16 + src1_col-1);
      src1_col -= 1;
    }
  }

}

factor = -1;
if(src1_col <= dest1_col){
  factor = 1;
}

while(src1_col != dest1_col){
  if(factor==1)
    path1.push('R');
  else
    path1.push('L');
  src1_col = src1_col+(factor);
  path1_data.push(src1_row*16+src1_col);
}

//Detect objects that lise on row path and if so avoid that
for(var i=0;i<path1_data.length;++i)
{
  if(path1_data[i] == dest0 && (path1[i] == 'L' || path1[i]=='R')){
    var j=0;
    for(j=0;j<path1.length;++j){
      if(path1[j] == 'R' || path1[j] == 'L'){
          break;
        }
      }
      --j;
      if(Math.floor(path1_data[j]/16) != 15){
        path1.splice(j,0,'U');
        path1_data.splice(j,0,path1_data[j]);
        for(j=j+1;j<path1.length;++j)
        {
          path1_data[j] += 16;
        }
        path1.push('D');
        path1_data.push(path1_data[path1_data.length-1]-16);
      }
      else {
        path1.splice(j,1);
        path1_data.splice(j,1);
        for(j=j+1;j<path1.length;++j)
        {
          path1_data[j] -= 16;
        }
        path1.push('U');
        path1_data.push(path1_data[path1_data.length-1]+16);

      }
        break;
    }
  }

  for(var i=0;i<path0_data.length;++i)
  {
    if(path0_data[i] == src1 && (path0[i] == 'L' || path0[i]=='R')){
      var j=0;
      for(j=0;j<path0.length;++j) {
        if(path0[j] == 'R' || path0[j] == 'L'){
          break;
        }
      }
      --j;
      if(Math.floor(path0_data[j]/16) != 15){
        path0.splice(j,0,'U');
        path0_data.splice(j,0,path0_data[j]);
        for(j=j+1;j<path0.length;++j)
        {
          path0_data[j] += 16;
        }
        path0.push('D');
        path0_data.push(path0_data[path0_data.length-1]-16);
      }
      else {
            path0.splice(j,1);
            path0_data.splice(j,1);
            for(j=j+1;j<path0.length;++j)
            {
              path0_data[j] -= 16;
            }
            path0.push('U');
            path0_data.push(path0_data[path0_data.length-1]+16);

          }
            break;
        }
      }


      document.write(path0+"<br>");
      document.write(path0_data+"<br>");

      document.write(path1+"<br>");
      document.write(path1_data+"<br>");
}
```

- **Worst Case Time Complexity - O(n)** n = number of rows
- **Worst Case Space Complexity - O(n)**

Links and References

1. ESP-32 Datasheet
(https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pd
)

2. Web RTC
(https://medium.com/@ahmetmermerkaya/webrtc-live-video-stream-broadcasting-on
e-to-many-and-watching-with-rtmp-and-hls-82980ccfe890)

3. NodeJS Server
(https://www.tutorialsteacher.com/nodejs/create-nodejs-web-server)

4. PID based encoder
(http://www.ijicic.org/ijicic-150313.pdf)

5. https based web server
(https://www.unixmen.com/lets-encrypt-secure-apache-web-server-ubuntu-16-04/)