Temperature Sensors

- Thermocouples
- RTD
- Thermistors
- Semiconductor based IC
- IR temperature sensor

Thermocouples:

Consists of two dissimilar metals, current generated from thermoelectric effect (hall effect). It’s self powered and has a wide temperature range. Simple and rugged. But it’s output is non-linear and varying, you need a reference temperature to figure out what the temp is. Also you need to amplify the input signal because it very small and not detectable by logic. Doesn't need power source.

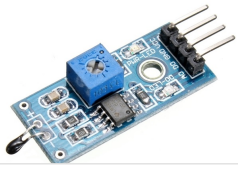
RTD:

It is the most precise temperature sensor, but also the most slowest sensor. Expensive and requires a power source to operate.

Buy 4Pin NTC Temperature Sensor Module Online at Low Price | Robu

Measure the temperature using a resistor! As resistance changes means we can measure temperature. Buy NOW 4pin NTC temperature sensor module Online at Robu.in

https://robu.in/product/4pin-ntc-thermistor-temperature-sensor-module/



-25 to 80 = range

LM35C or LM35 temperature sensor

Buy LM35 Temperature Sensor IC Online at Best Price in India | Robu.in

Buy LM35 an integrated analog temperature sensor IC online at the best price. Buy at the best price online at Robu.in. Shop NOW!!!

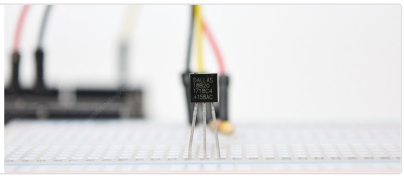
https://robu.in/product/lm35-to-92-3-board-mount-temperature-sensors/

DS18B20 sensor

Interfacing DS18B20 1-Wire Digital Temperature Sensor with Arduino

Learn about DS18B20 1-Wire Digital Temperature Sensor along with its Pinout, Specifications, Wiring & Arduino Code with Explanation

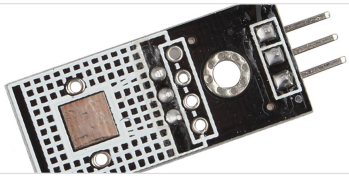
https://lastminuteengineers.com/ds18b20-arduino-tutorial/



Buy DS18B20 Temperature Sensor Module For Arduino Online | Robu.in

Get the Best Product To Plan Innovative Project Ideas! Start your DIY Planning Journey with Robu's Online Store! Buy NOW DS18B20 Temperature Sensor Module

https://robu.in/product/ds18b20-temperature-sensor-module-temperature-measurement-module-arduino/



nice website btw:

https://robu.in/product/ds18b20-temperature-sensor-module-temperature-measurement-module-arduino/

Last Minute Engineers

https://lastminuteengineers.com/

final bill of materials:

1x NTC thermistor sensor module

1x flame sensor

1x LM35

1x DS18B20 sensor

7 Displays to Output Data From Your Arduino

In this article we will take you through the different types of display available, where to get them, and how to set them up.

https://www.makeuseof.com/tag/arduino-displays-options/

LCD screens —> 8-12 pins.. but can be reduced to 4 pins via I2C protocol. check out.

LCD 16x2 screens optimal and are very lightweight in terms of power. SPI protocol for lcd screens? since spi is faster then i2c and given that spi also handles real time applications such as display screens just like i2c.

i2c can handle data transmissions upto 400kb/sec or sth …. keep this in mind and choose a lcd screen display

you need an i2c module for this aksjakjs

In-Depth: Interfacing an I2C LCD with Arduino

Learn to control I2C LCD with Arduino along with pinout, wiring, finding I2C address, adjusting contrast, arduino code, create and display custom characters

https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/

last minute engineers is a pretty good guy aksjdkjasd

LCD 16x2:

Buy 1602 LCD Display Blue Backlight Online at Lowest Price | Robu.in

Get the High Quality 1602 LCD Display Blue Backlight Online for your DIY application. For more offers and discount visit our Online Store

https://robu.in/product/basic-16x2-character-lcd-white-on-blue-on-blue-5v/

I2C module:

Buy I2C Serial Interface Adapter Module Online at Best Price | Robu.in

Buy NOW Lithium Battery at India's Best Online Shopping Store, This IIC/I2C Serial Interface adapter Module is used for LCD Display. For more info visit us

https://robu.in/product/iici2c-serial-interface-adapter-module/

Discord - A New Way to Chat with Friends & Communities

Discord is the easiest way to communicate over voice, video, and text. Chat, hang out, and stay close with your friends and communities.

https://discord.com/channels/@me/944291310116958208/998091755327070269

Learning A New Microcontroller - Steve Branam

Contents: Introduction Background For The Beginner The Peripherals System Complexity Support Software Do It Like Phil The Programs GPIO And Timers UART And...

https://www.embeddedrelated.com/showarticle/1455.php

white =coordinator (s2)

green =router (s2)

red = end device (s2c so buggy) apparently the most latest one??

XBee Module Interfacing with Arduino

In this tutorial we will interface XBee module with Arduino Uno board. The XBee connected with Arduino board will act as a receiver and it will communicate wirelessly with other XBee module which is serially connected with the laptop using a Explorer Board.

https://circuitdigest.com/microcontroller-projects/arduino-xbee-module-interfacing-tutorial

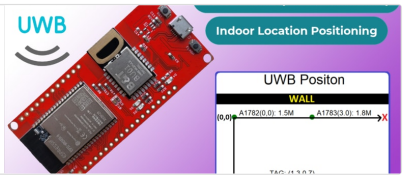
Exploring XBees and XCTU - SparkFun Learn

https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu/all

ESP32 DW1000 UWB Indoor Location Positioning System

ESP32 DW1000 UWB based Indoor Location Positioning System consists of two Anchors & a tag with Data visualization with Python & Arduino Code

https://how2electronics.com/esp32-dw1000-uwb-indoor-location-positioning-system/



python socket programming:

Python Programming Tutorials

Python Programming tutorials from beginner to advanced on a massive variety of topics. All video and text tutorials are free.

https://pythonprogramming.net/sockets-tutorial-python-3/

will help in making raspberry pi connect to the tag.

How to Turn a Raspberry Pi Into a Wi-Fi Access Point

Reuse your spare Raspberry Pi

https://www.tomshardware.com/how-to/raspberry-pi-access-point

for raspi as hotspot

and there are some ways to increase the wifi range

Alfa Long-Range Dual-Band AC1200 Wireless USB 3.0 Wi-Fi Adapter w/2x 5dBi External Antennas - 2.4GHz 300Mbps / 5Ghz 867Mbps - 802.11ac

Amazon.in: Buy Alfa Long-Range Dual-Band AC1200 Wireless USB 3.0 Wi-Fi Adapter w/2x 5dBi External Antennas - 2.4GHz 300Mbps / 5Ghz 867Mbps - 802.11ac & A online at low price in India on Amazon.in. Check out Alfa Long-Range Dual-Band AC1200 Wireless USB 3.0 Wi-Fi Adapter w/2x 5dBi External Antennas - 2.4GHz 300Mbps / 5Ghz 867Mbps - 802.11ac & A, B, G, N reviews, ratings, features, specifications and more at Amazon.in

https://www.amazon.in/Alfa-Long-Range-Dual-Band-Wireless-External/dp/B00MX57AO4

yes wifi dongles are a solution. i dont have proof, but intuition says they might increase hotspot range.

Why raspi should have hotspot on? multiple tags and reliable dongle range extension, single point of failure, easy and resuable code. no hassles basically.

what about the other choice? raspi connecting to tag, its not feasible for mulitple tags. only one device at a time, is there tech to connect to mutilple networks at once?

Raspberry Pi Documentation - Configuration

The official documentation for Raspberry Pi computers and microcontrollers

https://www.raspberrypi.com/documentation/computers/configuration.html#configuring-networking33

[Solved]How to increase WiFi range? - Raspberry Pi Forums

https://forums.raspberrypi.com/viewtopic.php?t=323215

esp32 must calculate distance and send to intermediate node = task

json to csv conversion

locally available esp32 with uwb module

tag —> nearest node —> node —> Rpi

tag measures distance through uwb, from uwb data it checks which device to connect to. esp now to communicate between two esp 32 and finally Rpi has hotspot on…. neartes esp32 will connect and send data continosly too

so, output:

```
{"1782": ["range": 1.25,"dbm": -66.35],"1783": ["range": 4.14,"dbm": -90.39],"temperature": -127.00}
```

comparing distances approach:

```
fresh_link(uwb_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), DW1000Ranging.
```

assign variables for address, range beforehand and then enter into fresh_link function and compare it.

Problems as of today (26th May 2023):

- No sign of data being sent over ESP NOW protocol after comparing weights (distances between two nodes).
- Code complexity 💀
- Tag code is too huge, might panick with even more additions ig

ok it worked. (29th May 2023)

```
13:21:21.835 -> Sent with success to A2
13:21:21.835 -> {"1783": ["range": 4.23,"dbm": -88.56],"1782": ["range": 1.75,"dbm": -76.73],"temperature": 34.63}
13:21:21.835 ->
13:21:21.835 -> Last Packet Send Status:        Delivery Success
13:21:22.945 -> Delete inactive device: 1782
13:21:23.127 -> From: 6019      Range: 3.90 m   RX power: -86.73 dBm
```

now all that is left is to transmit the data from anchor node to raspi.

```
#include <SPI.h>
#include <DW1000Ranging.h>
#include <WiFi.h>
#include "link.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <esp_now.h>


#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4
#define PIN_RST 27
#define PIN_IRQ 34
#define ONE_WIRE_BUS 14
float d1 = 100; // dist between tag and A1
float d2 = 100; //distance between tag and A2
int tempadd;
int a1 = 6019;  //address of A1
int a2 = 6018; // address of A2

uint8_t broadcastAddress1[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xFC};
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0};


struct MyLink *uwb_data;
int index_num = 0;
long runtime = 0;
String all_json = "";

// DS18B20 sensor setup
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{
    Serial.begin(115200);

    // Connect to WiFi
    WiFi.mode(WIFI_STA);

   if (esp_now_init() != ESP_OK){
     Serial.println("Error Initializing ESP-NOW");
     return;
   }

  esp_now_register_send_cb(OnDataSent);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
```

```
  delay(1000);

    // Initialize DW1000 module
    SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
    DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
    DW1000Ranging.attachNewRange(newRange);
    DW1000Ranging.attachNewDevice(newDevice);
    DW1000Ranging.attachInactiveDevice(inactiveDevice);

    // Start the module as a tag
    DW1000Ranging.startAsTag("7D:00:22:EA:82:60:3B:9C", DW1000.MODE_LONGDATA_RANGE_LOWPOWER);

    // Initialize the link data structure
    uwb_data = init_link();

    // Initialize the DS18B20 sensor
    sensors.begin();
}

void loop()
{

    DW1000Ranging.loop();
    if ((millis() - runtime) > 1000)
    {
        // Read temperature from the DS18B20 sensor
        sensors.requestTemperatures();
        float temperature = sensors.getTempCByIndex(0);

        // Create JSON string with range and temperature data
        make_link_json(uwb_data, &all_json);
        addTemperatureToJSON(&all_json, temperature);

        // Send data via UDP
        //sendUDP(&all_json);
      if (d1<d2){ //to A1
          esp_err_t result1 = esp_now_send(broadcastAddress1, (uint8_t *) &all_json, sizeof(all_json));
          if (result1==ESP_OK){
            Serial.println("Sent with success to A1");
          }
          else{
          Serial.println("Error sending data");
          }
      }
      else if (d2<d1){ // to A2
          esp_err_t result2 = esp_now_send(broadcastAddress2, (uint8_t *) &all_json, sizeof(all_json));
          if (result2==ESP_OK){
            Serial.println("Sent with success to A2");
          }
          else{
            Serial.println("Error sending data");
          }
      }
        Serial.println(all_json);
        delay(1000);
        runtime = millis();
    }
}

void newRange()
{
    Serial.print("From: ");
    tempadd=(DW1000Ranging.getDistantDevice()->getShortAddress());
    Serial.print(tempadd);
    Serial.print("\tRange: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRange());
    Serial.print(" m");
    Serial.print("\tRX power: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
    Serial.println(" dBm");
    float temperature = sensors.getTempCByIndex(0);
    Serial.println("\tTemperature : ");
    Serial.print(temperature);
    Serial.println(" C");
    fresh_link(uwb_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), DW1000Rang
    if (tempadd==a1){
      d1=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d1);
    }
    else if (tempadd==a2){
      d2=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d2);
    }
}

void newDevice(DW1000Device *device)
```

```
{
    Serial.print("Ranging init; 1 device added! -> ");
    Serial.print("Short: ");
    Serial.println(device->getShortAddress(), HEX);

    add_link(uwb_data, device->getShortAddress());
}

void inactiveDevice(DW1000Device *device)
{
    Serial.print("Delete inactive device: ");
    Serial.println(device->getShortAddress(), HEX);

    delete_link(uwb_data, device->getShortAddress());
}




void addTemperatureToJSON(String *json, float temperature)
{
    *json = json->substring(0, json->length() - 1); // Remove the closing brace
    *json += ",\"temperature\": " + String(temperature);
    *json += "}";
}
```

New problem (29th May 2023):

ESP NOW and raspi communication. ESP NOW and WiFi cant exist at the same time apparently. I've read some sources and they suggested to have a different MAC address? Attached link as reference.

https://github.com/jonasbystrom/ESP-Now-Sensor-system-with-WiFi

But is it possible to send from ESP NOW to Raspi without any of these intermediaries? Found one such question raised in stack exchange…. check out more about it.

Anyone get direct esp-now to raspberry pi (or other linux wifi device) working?

It would be nice to be able to use raspberry wifi to receive ESP-NOW messages directly and eliminate another ESP-to-PI device to manage.

https://raspberrypi.stackexchange.com/questions/83129/anyone-get-direct-esp-now-to-raspberry-pi-or-other-linux-wifi-device-working

Lots of ppl suggest MQTT, and consider ESP NOW useless since it's not modifiable. mhmmm. Attached another link as reference:

Is there something similar to ESP-NOW to use with RPi? - Raspberry Pi Forums

https://forums.raspberrypi.com/viewtopic.php?t=342570

and this seems cool too:

I used ESP-NOW to create a networking system for sensors on my vegetable farm without using WiFi

https://www.reddit.com/r/esp32/comments/o98x8p/i_used_espnow_to_create_a_networking_system_for/

and this guys claims to have solved this issue:

ESP32: WiFi and ESP-Now simultaneously - ElectroSoftCloud

In this guide I will teach you how to use the ESP-Now protocol together with the WiFi Station, in your ESP32 and ESP8266 microcontrollers.

https://www.electrosoftcloud.com/en/esp32-wifi-and-esp-now-simultaneously/

try the above solution tmrw (30th May 2023). It’s comment sections though * chef’s kiss *. Assuming they’re all real, its gold.

guido
14 de September de 2021 at 15:01

thanks for this article, it helped me get to the right solution

but, there is an easier solution, use “esp_wifi_set_ps(WIFI_PS_NONE);” to disable modem sleep

see https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#station-sleep

Reply

apparently even easier ohohoho

```
#include <stdio.h>
#include <math.h>

typedef struct {
    char node;
    float distance;
} Distance;

typedef struct {
    char current;
    Distance neighbors[3];
} Node;

Node nodes[] = {
    {'A', {{'D', 380.0}, {'\0', 0.0}, {'\0', 0.0}}},
    {'B', {{'C', 380.0}, {'\0', 0.0}, {'\0', 0.0}}},
    {'C', {{'B', 380.0}, {'D', 380.0}, {'E', 268.7}}},
    {'D', {{'A', 380.0}, {'C', 380.0}, {'E', 269.1}}},
    {'E', {{'C', 268.7}, {'D', 269.1}, {'\0', 0.0}}}
};

#define NUM_NODES sizeof(nodes) / sizeof(nodes[0])

void dijkstra(char current, Node nodes[], int numNodes, char start, char goal) {
    char unvisited[numNodes];
    char visited[numNodes];
    float currentDistance = 0.0;
    int unvisitedCount = numNodes;
    int i, j;

    for (i = 0; i < numNodes; i++) {
        unvisited[i] = 1;
        visited[i] = 0;
    }

    unvisited[start - 'A'] = 0;

    while (unvisitedCount > 0) {
        for (i = 0; i < numNodes; i++) {
            if (!unvisited[i]) continue;

            char currentNode = nodes[i].current;

            for (j = 0; j < 3; j++) {
                char neighbor = nodes[i].neighbors[j].node;
                float distance = nodes[i].neighbors[j].distance;

                if (neighbor == '\0') continue;
                if (!unvisited[neighbor - 'A']) continue;
```

```
                float newDistance = currentDistance + distance;

                int neighborIndex = neighbor - 'A';
                if (unvisited[neighborIndex] || newDistance < nodes[neighborIndex].neighbors[0].distance) {
                    nodes[neighborIndex].neighbors[0].distance = newDistance;
                }
            }
        }

        visited[current - 'A'] = 1;
        unvisited[current - 'A'] = 0;
        unvisitedCount--;

        if (current == goal) break;

        float minDistance = INFINITY;
        char nextNode;

        for (i = 0; i < numNodes; i++) {
            if (!unvisited[i]) continue;

            float distance = nodes[i].neighbors[0].distance;
            if (distance < minDistance) {
                minDistance = distance;
                nextNode = nodes[i].current;
            }
        }

        current = nextNode;
        currentDistance = minDistance;
    }
}

void func(float x, float y){
    float ad = sqrt(pow((x - 100), 2) + pow((y - 100), 2));
    ad = round(ad * 10) / 10;
    float bd = sqrt(pow((x - 500), 2) + pow((y - 100), 2));
    bd = round(bd * 10) / 10;
    float cd = sqrt(pow((x - 500), 2) + pow((y - 500), 2));
    cd = round(cd * 10) / 10;
    float dd = sqrt(pow((x - 100), 2) + pow((y - 500), 2));
    dd = round(dd * 10) / 10;

    char s;
    if (ad <= bd && ad <= cd && ad <= dd) {
        s = 'A';
    } else if (bd <= ad && bd <= cd && bd <= dd) {
        s = 'B';
    } else if (cd <= ad && cd <= bd && cd <= dd) {
        s = 'C';
    } else {
        s = 'D';
    }

    dijkstra(s, nodes, NUM_NODES, s, 'E');

    printf("Shortest distance to E: %.1f\n", nodes['E' - 'A'].neighbors[0].distance);
}

int main() {
    func(130, 720);
    return 0;
}
```

updated output:

```
{"1783": {"range": 1.54,"dbm": -74.56},"temperature": 34.31}
```

```
11:18:07.750 -> Received data:
11:18:07.750 -> ???
11:18:07.750 ->
```

data is garbled….. why is it happening? how to fix it?

is unsigned int a problem?

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void reverse(char* str, int length) {
    int start = 0;
    int end = length - 1;
    while (start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
}

void intToString(unsigned int num, char* str) {
    int i = 0;
    while (num > 0) {
        str[i++] = (num % 10) + '0';
        num /= 10;
    }
    str[i] = '\0';
    reverse(str, i);
}

unsigned int stringToInt(const char* str) {
    unsigned int result = 0;
    int i = 0;
    while (str[i] != '\0') {
        result = result * 10 + (str[i] - '0');
        i++;
    }
    return result;
}

int main() {
    unsigned int num = 15;
    char str[20];
    intToString(num, str);
    printf("Unsigned Integer: %u\n", num);
    printf("String: %s\n", str);

    unsigned int convertedNum = stringToInt(str);
    printf("Converted Integer: %u\n", convertedNum);

    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct Node {
    int x;
    int y;
    struct Node* parent;
    double g;
    double h;
    double f;
} Node;

Node* createNode(int x, int y) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->x = x;
    node->y = y;
    node->parent = NULL;
    node->g = 0.0;
    node->h = 0.0;
    node->f = 0.0;
    return node;
}

double heuristic(Node* current, Node* goal) {
    // Calculate Euclidean distance as the heuristic
    int dx = abs(current->x - goal->x);
    int dy = abs(current->y - goal->y);
    return sqrt(dx * dx + dy * dy);
}
```

```
void astar(int startX, int startY, int goalX, int goalY, int** grid, int width, int height) {
    // Create start and goal nodes
    Node* startNode = createNode(startX, startY);
    Node* goalNode = createNode(goalX, goalY);

    // Initialize open and closed lists
    Node** openList = (Node*)malloc(width * height * sizeof(Node));
    Node** closedList = (Node*)malloc(width * height * sizeof(Node));
    int openCount = 0;
    int closedCount = 0;

    // Add start node to open list
    openList[openCount++] = startNode;

    while (openCount > 0) {
        // Find node with the lowest f score in the open list
        Node* current = openList[0];
        int currentIndex = 0;
        for (int i = 1; i < openCount; i++) {
            if (openList[i]->f < current->f) {
                current = openList[i];
                currentIndex = i;
            }
        }

        // Move current node from open to closed list
        openCount--;
        for (int i = currentIndex; i < openCount; i++) {
            openList[i] = openList[i + 1];
        }
        closedList[closedCount++] = current;

        // Check if goal node is reached
        if (current->x == goalNode->x && current->y == goalNode->y) {
            // Path found, backtrack from goal to start
            Node* pathNode = current;
            printf("Path: ");
            while (pathNode != NULL) {
                printf("(%d, %d) ", pathNode->x, pathNode->y);
                pathNode = pathNode->parent;
            }
            printf("\n");
            break;
        }

        // Generate neighboring nodes
        int dx[8] = {-1, 0, 1, -1, 1, -1, 0, 1};
        int dy[8] = {-1, -1, -1, 0, 0, 1, 1, 1};

        for (int i = 0; i < 8; i++) {
            int newX = current->x + dx[i];
            int newY = current->y + dy[i];

            // Check if new position is within the grid
            if (newX >= 0 && newX < width && newY >= 0 && newY < height) {
                // Check if new position is traversable (0 in the grid represents obstacles)
                if (grid[newY][newX] == 0) {
                    // Create new node
                    Node* neighbor = createNode(newX, newY);
                    neighbor->parent = current;
                    neighbor->g = current->g + 1.0;
                    neighbor->h = heuristic(neighbor, goalNode);
                    neighbor->f = neighbor->g + neighbor->h;

                    // Check if neighbor is already in the closed list
                    int foundInClosed = 0;
                    for (int j = 0; j < closedCount; j++) {
                        if (closedList[j]->x == neighbor->x && closedList[j]->y == neighbor->y) {
                            foundInClosed = 1;
                            break;
                        }
                    }
                    if (foundInClosed)
                        continue;

                    // Check if neighbor is already in the open list
                    int foundInOpen = 0;
                    for (int j = 0; j < openCount; j++) {
                        if (openList[j]->x == neighbor->x && openList[j]->y == neighbor->y) {
                            foundInOpen = 1;
                            break;
                        }
                    }
                    if (!foundInOpen) {
                        // Add neighbor to open list
                        openList[openCount++] = neighbor;
```

```
                    } else {
                        // Update neighbor's g score if it's a better path
                        if (neighbor->g < current->g) {
                            neighbor->parent = current;
                            neighbor->g = current->g + 1.0;
                            neighbor->f = neighbor->g + neighbor->h;
                        }
                    }
                }
            }
        }
    }

    // Clean up
    free(openList);
    free(closedList);
}

int main() {
    int width = 5;
    int height = 5;

    // Example grid representing obstacles (1) and free spaces (0)
    int** grid = (int*)malloc(height * sizeof(int));
    for (int i = 0; i < height; i++) {
        grid[i] = (int*)malloc(width * sizeof(int));
        for (int j = 0; j < width; j++) {
            grid[i][j] = 0;
        }
    }
    grid[1][1] = 1;
    grid[1][2] = 1;
    grid[2][2] = 1;
    grid[3][2] = 1;

    // Start and goal positions
    int startX = 0;
    int startY = 0;
    int goalX = 4;
    int goalY = 4;

    // Run A* algorithm
    astar(startX, startY, goalX, goalY, grid, width, height);

    // Clean up
    for (int i = 0; i < height; i++) {
        free(grid[i]);
    }
    free(grid);

    return 0;
}
```

```
#include <SPI.h>
#include <DW1000Ranging.h>
#include <WiFi.h>
#include "link.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <esp_now.h>


#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4
#define PIN_RST 27
#define PIN_IRQ 34
#define ONE_WIRE_BUS 14
float d1 = 100; // dist between tag and A1
float d2 = 100; //distance between tag and A2
int tempadd;
int a1 = 6019;  //address of A1
int a2 = 6018; // address of A2

uint8_t broadcastAddress1[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xFC};   // to A1
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0};   // to A2


struct MyLink *uwb_data;
```

```
int index_num = 0;
long runtime = 0;
String all_json = "";

// DS18B20 sensor setup
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{
    Serial.begin(115200);

    // Connect to WiFi
    WiFi.mode(WIFI_STA);

   if (esp_now_init() != ESP_OK){
     Serial.println("Error Initializing ESP-NOW");
     return;
   }

  esp_now_register_send_cb(OnDataSent);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  delay(1000);

    // Initialize DW1000 module
    SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
    DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
    DW1000Ranging.attachNewRange(newRange);
    DW1000Ranging.attachNewDevice(newDevice);
    DW1000Ranging.attachInactiveDevice(inactiveDevice);

    // Start the module as a tag
    DW1000Ranging.startAsTag("7D:00:22:EA:82:60:3B:9C", DW1000.MODE_LONGDATA_RANGE_LOWPOWER);

    // Initialize the link data structure
    uwb_data = init_link();

    // Initialize the DS18B20 sensor
    sensors.begin();

}

void loop()
{

    DW1000Ranging.loop();
    if ((millis() - runtime) > 1000)
    {
        // Read temperature from the DS18B20 sensor
        sensors.requestTemperatures();
        float temperature = sensors.getTempCByIndex(0);

        // Create JSON string with range and temperature data
        make_link_json(uwb_data, &all_json);
        addTemperatureToJSON(&all_json, temperature);

        // Convert all_json to a byte array

        /*uint8_t all_json_bytes[all_json.length()];
        all_json.getBytes(all_json_bytes, all_json.length());*/
        uint8_t all_json_bytes[all_json.length() + 1];  // Add 1 for null terminator
        strcpy((char*)all_json_bytes, all_json.c_str());  // Convert string to byte array




      if (d1<d2){ //to A1
```

```
          esp_err_t result1 = esp_now_send(broadcastAddress1, (uint8_t *) &all_json, sizeof(all_json));

          if (result1==ESP_OK){
            Serial.println("Sent with success to A1");
          }
          else{
          Serial.println("Error sending data");
          }
      }
      else if (d2<d1){ // to A2
          esp_err_t result2 = esp_now_send(broadcastAddress2, (uint8_t *) &all_json, sizeof(all_json));

          if (result2==ESP_OK){
            Serial.println("Sent with success to A2");
          }
          else{
            Serial.println("Error sending data");
          }
      }
        Serial.println(all_json);
        /*for(int i=0; i<sizeof(all_json_bytes); i++){
          Serial.println(*ptr+i);
        }*/
        delay(1000);
        runtime = millis();
    }
}

void newRange()
{
    Serial.print("From: ");
    tempadd=(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);
    Serial.print(tempadd);
    Serial.print("\tRange: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRange());
    Serial.print(" m");
    Serial.print("\tRX power: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
    Serial.println(" dBm");
    float temperature = sensors.getTempCByIndex(0);
    Serial.println("\tTemperature : ");
    Serial.print(temperature);
    Serial.println(" C");
    fresh_link(uwb_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), DW1000Rang
    if (tempadd==a1){
      d1=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d1);
    }
    else if (tempadd==a2){
      d2=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d2);
    }
}

void newDevice(DW1000Device *device)
{
    Serial.print("Ranging init; 1 device added! -> ");
    Serial.print("Short: ");
    Serial.println(device->getShortAddress(), HEX);

    add_link(uwb_data, device->getShortAddress());
}

void inactiveDevice(DW1000Device *device)
{
    Serial.print("Delete inactive device: ");
    Serial.println(device->getShortAddress(), HEX);

    delete_link(uwb_data, device->getShortAddress());
}




void addTemperatureToJSON(String *json, float temperature)
{
    *json = json->substring(0, json->length() - 1); // Remove the closing brace
    *json += ",\"temperature\": " + String(temperature);
    *json += "}";
}
```

```
#include <SPI.h>
#include <DW1000Ranging.h>
#include <WiFi.h>
#include "link.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <esp_now.h>


#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4
#define PIN_RST 27
#define PIN_IRQ 34
#define ONE_WIRE_BUS 14
float d1 = 100; // dist between tag and A1
float d2 = 100; //distance between tag and A2
int tempadd;
int a1 = 6019;  //address of A1
int a2 = 6018; // address of A2

uint8_t broadcastAddress1[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xFC};   // to A1
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0};   // to A2


struct MyLink *uwb_data;
int index_num = 0;
long runtime = 0;
String all_json = "";

// DS18B20 sensor setup
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{
    Serial.begin(115200);

    // Connect to WiFi
    WiFi.mode(WIFI_STA);

   if (esp_now_init() != ESP_OK){
     Serial.println("Error Initializing ESP-NOW");
     return;
   }

  esp_now_register_send_cb(OnDataSent);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  delay(1000);

    // Initialize DW1000 module
    SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
    DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
    DW1000Ranging.attachNewRange(newRange);
    DW1000Ranging.attachNewDevice(newDevice);
    DW1000Ranging.attachInactiveDevice(inactiveDevice);

    // Start the module as a tag
    DW1000Ranging.startAsTag("7D:00:22:EA:82:60:3B:9C", DW1000.MODE_LONGDATA_RANGE_LOWPOWER);

    // Initialize the link data structure
    uwb_data = init_link();

    // Initialize the DS18B20 sensor
```

```
    sensors.begin();

}

void loop()
{

    DW1000Ranging.loop();
    if ((millis() - runtime) > 1000)
    {
        // Read temperature from the DS18B20 sensor
        sensors.requestTemperatures();
        float temperature = sensors.getTempCByIndex(0);

        // Create JSON string with range and temperature data
        make_link_json(uwb_data, &all_json);
        addTemperatureToJSON(&all_json, temperature);

        // Convert all_json to a byte array

        /*uint8_t all_json_bytes[all_json.length()];
        all_json.getBytes(all_json_bytes, all_json.length());*/
        uint8_t all_json_bytes[all_json.length() + 1];  // Add 1 for null terminator
        strcpy((char*)all_json_bytes, all_json.c_str());  // Convert string to byte array




      if (d1<d2){ //to A1
          esp_err_t result1 = esp_now_send(broadcastAddress1, (uint8_t *) &all_json_bytes, sizeof(all_json_bytes));

          if (result1==ESP_OK){
            Serial.println("Sent with success to A1");
          }
          else{
          Serial.println("Error sending data");
          }
      }
      else if (d2<d1){ // to A2
          esp_err_t result2 = esp_now_send(broadcastAddress2, (uint8_t *) &all_json_bytes, sizeof(all_json_bytes));

          if (result2==ESP_OK){
            Serial.println("Sent with success to A2");
          }
          else{
            Serial.println("Error sending data");
          }
      }
        Serial.println(all_json);
        /*for(int i=0; i<sizeof(all_json_bytes); i++){
          Serial.println(*ptr+i);
        }*/
        delay(1000);
        runtime = millis();
    }
}

void newRange()
{
    Serial.print("From: ");
    tempadd=(DW1000Ranging.getDistantDevice()->getShortAddress());
    Serial.print(tempadd);
    Serial.print("\tRange: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRange());
    Serial.print(" m");
    Serial.print("\tRX power: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
    Serial.println(" dBm");
    float temperature = sensors.getTempCByIndex(0);
    Serial.println("\tTemperature : ");
    Serial.print(temperature);
    Serial.println(" C");
    fresh_link(uwb_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), DW1000Rang
    if (tempadd==a1){
      d1=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d1);
    }
    else if (tempadd==a2){
      d2=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d2);
    }
}

void newDevice(DW1000Device *device)
{
    Serial.print("Ranging init; 1 device added! -> ");
    Serial.print("Short: ");
```

```
    Serial.println(device->getShortAddress(), HEX);

    add_link(uwb_data, device->getShortAddress());
}

void inactiveDevice(DW1000Device *device)
{
    Serial.print("Delete inactive device: ");
    Serial.println(device->getShortAddress(), HEX);

    delete_link(uwb_data, device->getShortAddress());
}




void addTemperatureToJSON(String *json, float temperature)
{
    *json = json->substring(0, json->length() - 1); // Remove the closing brace
    *json += ",\"temperature\": " + String(temperature);
    *json += "}";
}
```

```
#include <SPI.h>
#include "DW1000Ranging.h"
#include <esp_now.h>
#include <WiFi.h>
#include <ArduinoJson.h>

#define ANCHOR_ADD "83:17:5B:D5:A9:9A:E2:9C"

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4

typedef struct struct_message {
  char a[200];
} struct_message;

struct_message receivedData;

esp_now_peer_info_t peerInfo;

void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  memcpy(&receivedData, data, sizeof(struct_message));
  Serial.print("Bytes Received: ");
  Serial.println(data_len);
  Serial.print("Received data: ");

  for (int i = 0; i < data_len; i++) {
    Serial.print((char)data[i]);
  }

  Serial.println();
}


// Connection pins
const uint8_t PIN_RST = 27; // Reset pin
const uint8_t PIN_IRQ = 34; // IRQ pin
const uint8_t PIN_SS = 4;   // SPI select pin

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);

  // Set receiver MAC address
  uint8_t receiverMacAddress[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFB, 0x20};
  memcpy(peerInfo.peer_addr, receiverMacAddress, 6);

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
```

```
  }

  delay(1000);

  // Initialize DW1000 module
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
  DW1000Ranging.attachNewRange(newRange);
  DW1000Ranging.attachBlinkDevice(newBlink);
  DW1000Ranging.attachInactiveDevice(inactiveDevice);

  // Start the module as an anchor
  DW1000Ranging.startAsAnchor(ANCHOR_ADD, DW1000.MODE_LONGDATA_RANGE_LOWPOWER, false);
}

void loop() {
  DW1000Ranging.loop();
}

void newRange() {
  Serial.print("From: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);
  Serial.print("\tRange: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRange());
  Serial.print(" m");
  Serial.print("\tRX power: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
  Serial.println(" dBm");
}

void newBlink(DW1000Device *device) {
  Serial.print("Blink; 1 device added! -> ");
  Serial.print("Short: ");
  Serial.println(device->getShortAddress(), HEX);
}

void inactiveDevice(DW1000Device *device) {
  Serial.print("Delete inactive device: ");
  Serial.println(device->getShortAddress(), HEX);
}
```

How is tag5.ino different you ask? It sends the array as an whole in the form of a byte array of type uint8_t. Why wasnt it sending in between? It's cuz the address was taken out in HEX values, and it wasn't able to execute that if-else conditions. I spent a week on it cuz I already resolved this issue in the past, but the new code was taken from the past… and I didnt take into account about this correction. So basically, the best solution is achieved by sending the data in the form of a byte array (less space is consumed) and sending it as a whole. The anchor receives this byte array as a whole and converts into a string. Thus no more garbled output.

Localization Techniques in Wireless Sensor Networks - Nabil Ali Alrajeh, Maryam Bashir, Bilal Shams, 2013

The important function of a sensor network is to collect and forward data to destination. It is very important to know about the location of collected data. Thi...

https://journals.sagepub.com/doi/10.1155/2013/304628

A reference for different possible wireless localization methods.

ESP32 com protocolo ESP-Now

Uma rede muito especial, de alta velocidade e, sendo assim, perfeita para a automação residencial e industrial, e que se trata de mais...

https://www.fernandok.com/2018/03/esp32-com-protocolo-esp-now.html

This spanish guy's code helped me explore the possibilities of sending an array through esp_now_send() function!

Setting up a Raspberry Pi 3 as an Access Point - SparkFun Learn

https://learn.sparkfun.com/tutorials/setting-up-a-raspberry-pi-3-as-an-access-point/all

setting up raspi as hotspot tutorial, this one worked… follow in future if not working again.

```
#include <SPI.h>
#include "DW1000Ranging.h"
#include <esp_now.h>
#include <WiFi.h>
#include <WiFiUdp.h>   // Library for UDP communication
#include <ArduinoJson.h>

#define ANCHOR_ADD "83:17:5B:D5:A9:9A:E2:9C" //for anchor A1

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4

typedef struct struct_message {
  char a[200];
} struct_message;

struct_message receivedData;

esp_now_peer_info_t peerInfo;



void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  memcpy(&receivedData, data, sizeof(struct_message));
  Serial.print("Bytes Received: ");
  Serial.println(data_len);
  Serial.print("Received data: ");

  for (int i = 0; i < data_len; i++) {
    Serial.print((char)data[i]);
  }

  Serial.println();

  // Send data via UDP
  sendUDPData(receivedData.a);
}

// Connection pins
const uint8_t PIN_RST = 27; // Reset pin
const uint8_t PIN_IRQ = 34; // IRQ pin
const uint8_t PIN_SS = 4;   // SPI select pin

// UDP communication variables
const char* ssid = "MyPiAP";
const char* password = "raspberry";
const IPAddress udpServerIP(192, 168, 5, 1);  // IP address of the Raspberry Pi
const unsigned int udpServerPort = 5001;        // UDP port number on the Raspberry Pi
WiFiUDP udp;


void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP_STA);

  // Connect to Wi-Fi
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);

  // Set receiver MAC address
  uint8_t receiverMacAddress[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFB, 0x20};
  memcpy(peerInfo.peer_addr, receiverMacAddress, 6);

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
```

```
  delay(1000);

  // Initialize DW1000 module
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
  DW1000Ranging.attachNewRange(newRange);
  DW1000Ranging.attachBlinkDevice(newBlink);
  DW1000Ranging.attachInactiveDevice(inactiveDevice);

  // Start the module as an anchor
  DW1000Ranging.startAsAnchor(ANCHOR_ADD, DW1000.MODE_LONGDATA_RANGE_LOWPOWER, false);
}

void loop() {
  DW1000Ranging.loop();
}

void newRange() {
  Serial.print("From: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);
  Serial.print("\tRange: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRange());
  Serial.print(" m");
  Serial.print("\tRX power: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
  Serial.println(" dBm");
}

void newBlink(DW1000Device *device) {
  Serial.print("Blink; 1 device added! -> ");
  Serial.print("Short: ");
  Serial.println(device->getShortAddress(), HEX);
}

void inactiveDevice(DW1000Device *device) {
  Serial.print("Delete inactive device: ");
  Serial.println(device->getShortAddress(), HEX);
}

void sendUDPData(const char* data) {
  udp.beginPacket(udpServerIP, udpServerPort);
  udp.print(data);
  udp.endPacket();
  Serial.println("Data sent via UDP");
}
```

https://www.youtube.com/watch?v=iNWH1FuM320

https://stackoverflow.com/questions/74957548/esp-32-with-esp-now-and-mqtt-connection

https://stackoverflow.com/questions/74957548/esp-32-with-esp-now-and-mqtt-connection

impossible to have esp now and wifi at the same time.

```
#include <SPI.h>
#include "DW1000Ranging.h"
#include <esp_now.h>
#include <WiFi.h>
#include <ArduinoJson.h>
#include <HardwareSerial.h>

#define ANCHOR_ADD "83:17:5B:D5:A9:9A:E2:9C" //for anchor A1

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
```

```
#define DW_CS 4

HardwareSerial SerialUSB(2);

typedef struct struct_message {
  char a[200];
} struct_message;

struct_message receivedData;

esp_now_peer_info_t peerInfo;

void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  memcpy(&receivedData, data, sizeof(struct_message));
  //Serial.print("Bytes Received: ");
  //Serial.println(data_len);
  Serial.print("Received data: ");

  for (int i = 0; i < data_len; i++) {
    Serial.print((char)data[i]);
    ///SerialUSB.print((char)data[i]);
  }

  Serial.println();
}


// Connection pins
const uint8_t PIN_RST = 27; // Reset pin
const uint8_t PIN_IRQ = 34; // IRQ pin
const uint8_t PIN_SS = 4;   // SPI select pin

void setup() {
  Serial.begin(115200);
  SerialUSB.begin(115200);
  while(!SerialUSB){
    ;
  }
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    //Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);

  // Set receiver MAC address
  uint8_t receiverMacAddress[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFB, 0x20};
  memcpy(peerInfo.peer_addr, receiverMacAddress, 6);

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    //Serial.println("Failed to add peer");
    return;
  }

  delay(1000);

  // Initialize DW1000 module
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
  DW1000Ranging.attachNewRange(newRange);
  DW1000Ranging.attachBlinkDevice(newBlink);
  DW1000Ranging.attachInactiveDevice(inactiveDevice);

  // Start the module as an anchor
  DW1000Ranging.startAsAnchor(ANCHOR_ADD, DW1000.MODE_LONGDATA_RANGE_LOWPOWER, false);
}

void loop() {
  DW1000Ranging.loop();
}

void newRange() {
  /*Serial.print("From: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);
  Serial.print("\tRange: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRange());
  Serial.print(" m");
  Serial.print("\tRX power: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
  Serial.println(" dBm");*/
}

void newBlink(DW1000Device *device) {
  /*Serial.print("Blink; 1 device added! -> ");
```

```
  Serial.print("Short: ");
  Serial.println(device->getShortAddress(), HEX);*/
}

void inactiveDevice(DW1000Device *device) {
  /*Serial.print("Delete inactive device: ");
  Serial.println(device->getShortAddress(), HEX);*/
}
```

python serial code:

```
import serial
import time
ser = serial.Serial('/dev/ttyUSB0', 115200)

while True:
    if ser.in_waiting > 0:
        data = ser.readline().decode().strip()
        # print(type(data)) <class 'str'>
        if (data[ :15] == "Received data: "):
            print(f"{data[15:]")
        else:
            print("wrong") #debugging purposes

        # for future data processing tasks
    ser.flush()
    #time.sleep(1) removed this cuz its not with sync with received data
ser.close() #you can remove this in future updates if you want, try for sth similar to spin
```

```
#include <stdio.h>
#include <math.h>

int main() {
    double x1 = 3, y1 = 5;
    double x2 = 7, y2 = 5;
    double c = 4;
    double a = 12;
    double b = 14;
    double cos_alpha = (b*b+c*c-a*a)/(2*b*c);
    double sin_alpha = sqrt(1-(cos_alpha)*(cos_alpha));
    double x3 = b*cos_alpha, y3 = b*sin_alpha;
    printf("%f %f", x3, y3);
    return 0;
}
```

```
#include <stdio.h>
#include <math.h>

char nodes[] = {'A', 'B', 'C', 'D', 'E'};
float distances[5][5] = {
    {0, 0, 0, 380.0, 0},
    {0, 0, 380.0, 0, 0},
    {0, 380.0, 0, 380.0, 268.7},
    {380.0, 0, 380.0, 0, 269.1},
    {0, 0, 268.7, 269.1, 0}
};

void dijkstra(char current, char nodes[], float distances[5][5], float *visited, char *prevNode) {
    float unvisited[5];
    float currentDistance = 0;
    int i, j;

    for (i = 0; i < 5; i++) {
        unvisited[i] = INFINITY;
        visited[i] = INFINITY;
        prevNode[i] = '\0';  // Initialize previous node as null character
    }

    unvisited[current - 'A'] = 0;

    while (1) {
        for (j = 0; j < 5; j++) {
```

```
            char neighbour = nodes[j];
            float distance = distances[current - 'A'][j];
            if (distance == 0 || visited[j] <= currentDistance + distance)
                continue;

            float newDistance = currentDistance + distance;
            int neighbourIndex = neighbour - 'A';
            if (unvisited[neighbourIndex] == INFINITY || unvisited[neighbourIndex] > newDistance) {
                unvisited[neighbourIndex] = newDistance;
                prevNode[neighbourIndex] = current;  // Set previous node for the neighbor
            }
        }

        visited[current - 'A'] = currentDistance;
        unvisited[current - 'A'] = INFINITY;

        int done = 1;
        float minDistance = INFINITY;
        char nextNode;
        for (i = 0; i < 5; i++) {
            if (unvisited[i] < INFINITY) {
                done = 0;
                if (unvisited[i] < minDistance) {
                    minDistance = unvisited[i];
                    nextNode = nodes[i];
                }
            }
        }

        if (done)
            break;
        current = nextNode;
        currentDistance = minDistance;
    }
}

void printPath(char start, char end, char *prevNode) {
    if (start == end) {
        printf("%c ", start);
        return;
    }
    printPath(start, prevNode[end - 'A'], prevNode);
    printf("%c ", end);
}

void func(float x, float y) {
    float ad = sqrt(pow((x - 100), 2) + pow((y - 100), 2));
    ad = round(ad * 10) / 10;
    float bd = sqrt(pow((x - 500), 2) + pow((y - 100), 2));
    bd = round(bd * 10) / 10;
    float cd = sqrt(pow((x - 500), 2) + pow((y - 500), 2));
    cd = round(cd * 10) / 10;
    float dd = sqrt(pow((x - 100), 2) + pow((y - 500), 2));
    dd = round(dd * 10) / 10;

    char start_node;
    if (ad <= bd && ad <= cd && ad <= dd)
        start_node = 'A';
    else if (bd <= ad && bd <= cd && bd <= dd)
        start_node = 'B';
    else if (cd <= ad && cd <= bd && cd <= dd)
        start_node = 'C';
    else
        start_node = 'D';

    float visited[5];
    char prevNode[5];
    dijkstra(start_node, nodes, distances, visited, prevNode);

    printf("Shortest distance to E: %.1f\n", visited['E' - 'A']);
    printf("Path to E: ");
    printPath(start_node, 'E', prevNode);
    printf("\n");
}

int main() {
    func(10, 190);
    return 0;
}
```

A3 mac address: `54:43:B2:7F:56:BC`

A4 mac address: `D4:D4:DA:46:70:58`

A5 mac address: `D4:D4:DA:46:72:74`

`D4:D4:DA:46:72:74`

A6 mac address: `D4:D4:DA:46:6C:E4`

A7 mac address: `D4:D4:DA:46:68:C4`

```
#include <SPI.h>
#include <DW1000Ranging.h>
#include <WiFi.h>
#include "link.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <esp_now.h>
#include <math.h>


#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4
#define PIN_RST 27
#define PIN_IRQ 34
#define ONE_WIRE_BUS 14
float d1 = 100; // dist between tag and A1
float d2 = 100; //distance between tag and A2

float x; //coordinates for tag
float y;

int tempadd;
int a1 = 6019;  //address of A1
int a2 = 6018; // address of A2

char nodes[] = {'A', 'B', 'C', 'D', 'E'};
float distances[5][5] = {
    {0, 0, 0, 380.0, 0},
    {0, 0, 380.0, 0, 0},
    {0, 380.0, 0, 380.0, 268.7},
    {380.0, 0, 380.0, 0, 269.1},
    {0, 0, 268.7, 269.1, 0}
};

uint8_t broadcastAddress1[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xFC};   // to A1
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0};   // to A2


struct MyLink *uwb_data;
int index_num = 0;
long runtime = 0;
String all_json = "";

typedef struct algdata {
  uint8_t disdata[200];
  char path[10];
  int count;
  int length;
} algdata;

algdata data;

// DS18B20 sensor setup
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
esp_now_peer_info_t peerInfo;


void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{
    Serial.begin(115200);

    // Connect to WiFi
    WiFi.mode(WIFI_STA);
```

```
   if (esp_now_init() != ESP_OK){
     Serial.println("Error Initializing ESP-NOW");
     return;
   }

  esp_now_register_send_cb(OnDataSent);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  delay(1000);

    // Initialize DW1000 module
    SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
    DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
    DW1000Ranging.attachNewRange(newRange);
    DW1000Ranging.attachNewDevice(newDevice);
    DW1000Ranging.attachInactiveDevice(inactiveDevice);

    // Start the module as a tag
    DW1000Ranging.startAsTag("7D:00:22:EA:82:60:3B:9C", DW1000.MODE_LONGDATA_RANGE_LOWPOWER);

    // Initialize the link data structure
    uwb_data = init_link();

    // Initialize the DS18B20 sensor
    sensors.begin();

    /*data.path[0]=6019;
    data.path[1]=6018;
    data.path[2]=6017;*/
    data.count=0;

}

void loop()
{

    DW1000Ranging.loop();
    if ((millis() - runtime) > 1000)
    {
        // Read temperature from the DS18B20 sensor
        sensors.requestTemperatures();
        float temperature = sensors.getTempCByIndex(0);

        // Create JSON string with range and temperature data
        make_link_json(uwb_data, &all_json);
        addTemperatureToJSON(&all_json, temperature);

        // Convert all_json to a byte array

        /*uint8_t all_json_bytes[all_json.length()];
        all_json.getBytes(all_json_bytes, all_json.length());*/
        uint8_t all_json_bytes[all_json.length() + 1];  // Add 1 for null terminator
        strcpy((char*)all_json_bytes, all_json.c_str());  // Convert string to byte array
        int length_of_arr = sizeof(all_json_bytes);
        memcpy(data.disdata, all_json_bytes, sizeof(all_json_bytes));
        data.length=length_of_arr;
        Serial.print("Length of all_json_bytes is : ");
        Serial.println(data.length);

      if (d1<d2){ //to A1
          esp_err_t result1 = esp_now_send(broadcastAddress1, (uint8_t *) &data, sizeof(data));

          if (result1==ESP_OK){
            Serial.println("Sent with success to A1");
          }
          else{
          Serial.println("Error sending data");
          }
      }
      else if (d2<d1){ // to A2
          esp_err_t result2 = esp_now_send(broadcastAddress2, (uint8_t *) &data, sizeof(data));

          if (result2==ESP_OK){
            Serial.println("Sent with success to A2");
```

```
          }
          else{
            Serial.println("Error sending data");  // in case if failure send data to A1, potential for recursive function?? try until
          }
      }
        Serial.println(all_json);
        /*for(int i=0; i<sizeof(all_json_bytes); i++){
          Serial.println(*ptr+i);
        }*/
        delay(1000);
        runtime = millis();
    }
}

void newRange()
{
    Serial.print("From: ");
    tempadd=(DW1000Ranging.getDistantDevice()->getShortAddress());
    Serial.print(tempadd);
    Serial.print("\tRange: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRange());
    Serial.print(" m");
    Serial.print("\tRX power: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
    Serial.println(" dBm");
    float temperature = sensors.getTempCByIndex(0);
    Serial.println("\tTemperature : ");
    Serial.print(temperature);
    Serial.println(" C");
    fresh_link(uwb_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), temperatur
    if (tempadd==a1){
      d1=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d1);
    }
    else if (tempadd==a2){
      d2=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d2);
    }
    x=locationx(d1,d2);
    y=locationy(d1,d2);

    func(x,y);
}

void newDevice(DW1000Device *device)
{
    Serial.print("Ranging init; 1 device added! -> ");
    Serial.print("Short: ");
    Serial.println(device->getShortAddress(), HEX);

    add_link(uwb_data, device->getShortAddress());
}

void inactiveDevice(DW1000Device *device)
{
    Serial.print("Delete inactive device: ");
    Serial.println(device->getShortAddress(), HEX);

    delete_link(uwb_data, device->getShortAddress());
}

float locationx(float d1, float d2)
{
    float x1 = 3, y1 = 5;
    float x2 = 7, y2 = 5;
    float c = 4;
    float cos_alpha = (d2*d2+c*c-d1*d1)/(2*d2*c);
    float sin_alpha = sqrt(1-(cos_alpha)*(cos_alpha));
    float x3 = d2*cos_alpha, y3 = d2*sin_alpha;
    return(x3);
}

float locationy(float d1, float d2)
{
    float x1 = 3, y1 = 5;
    float x2 = 7, y2 = 5;
    float c = 4;
    float cos_alpha = (d2*d2+c*c-d1*d1)/(2*d2*c);
    float sin_alpha = sqrt(1-(cos_alpha)*(cos_alpha));
    float x3 = d2*cos_alpha, y3 = d2*sin_alpha;
    return(y3);
}

void dijkstra(char current, char nodes[], float distances[5][5], float *visited, char *prevNode) {
    float unvisited[5];
    float currentDistance = 0;
    int i, j;
```

```
    for (i = 0; i < 5; i++) {
        unvisited[i] = INFINITY;
        visited[i] = INFINITY;
        prevNode[i] = '\0';  // Initialize previous node as null character
    }

    unvisited[current - 'A'] = 0;

    while (1) {
        for (j = 0; j < 5; j++) {
            char neighbour = nodes[j];
            float distance = distances[current - 'A'][j];
            if (distance == 0 || visited[j] <= currentDistance + distance)
                continue;

            float newDistance = currentDistance + distance;
            int neighbourIndex = neighbour - 'A';
            if (unvisited[neighbourIndex] == INFINITY || unvisited[neighbourIndex] > newDistance) {
                unvisited[neighbourIndex] = newDistance;
                prevNode[neighbourIndex] = current;  // Set previous node for the neighbor
            }
        }

        visited[current - 'A'] = currentDistance;
        unvisited[current - 'A'] = INFINITY;

        int done = 1;
        float minDistance = INFINITY;
        char nextNode;
        for (i = 0; i < 5; i++) {
            if (unvisited[i] < INFINITY) {
                done = 0;
                if (unvisited[i] < minDistance) {
                    minDistance = unvisited[i];
                    nextNode = nodes[i];
                }
            }
        }

        if (done)
            break;
        current = nextNode;
        currentDistance = minDistance;
    }
}

void func(float x, float y) {
    float ad = sqrt(pow((x - 100), 2) + pow((y - 100), 2));
    ad = round(ad * 10) / 10;
    float bd = sqrt(pow((x - 500), 2) + pow((y - 100), 2));
    bd = round(bd * 10) / 10;
    float cd = sqrt(pow((x - 500), 2) + pow((y - 500), 2));
    cd = round(cd * 10) / 10;
    float dd = sqrt(pow((x - 100), 2) + pow((y - 500), 2));
    dd = round(dd * 10) / 10;

    char start_node;
    if (ad <= bd && ad <= cd && ad <= dd)
        start_node = 'A';
    else if (bd <= ad && bd <= cd && bd <= dd)
        start_node = 'B';
    else if (cd <= ad && cd <= bd && cd <= dd)
        start_node = 'C';
    else
        start_node = 'D';

    float visited[5];
    char prevNode[5];
    dijkstra(start_node, nodes, distances, visited, prevNode);

    Serial.print("Shortest distance to E: ");
    Serial.println(visited['E' - 'A']);
    Serial.print("Path to E: ");
    printPath(start_node, 'E', prevNode);
    Serial.print("\n");
}

void printPath(char start, char end, char *prevNode) {
    int counter=0;
    if (start == end) {
        Serial.print(start);
        data.path[0]=start;
        return;
    }
    printPath(start, prevNode[end - 'A'], prevNode);
    data.path[counter]=end;
```

```
    Serial.print(end);
    counter++;
    return;
}


void addTemperatureToJSON(String *json, float temperature)
{
    *json = json->substring(0, json->length() - 1); // Remove the closing brace
    *json += ",\"temperature\": " + String(temperature);
    *json += "}";
}
```

```
#include <SPI.h>
#include "DW1000Ranging.h"
#include <esp_now.h>
#include <WiFi.h>
#include <ArduinoJson.h>

#define ANCHOR_ADD "83:17:5B:D5:A9:9A:E2:9C" //for anchor A1
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0}; //b
uint8_t broadcastAddress3[] = {0x54, 0x43, 0xB2, 0x7F, 0x56, 0xBC}; //c
uint8_t broadcastAddress4[] = {0xD4, 0xD4, 0xDA, 0x46, 0x70, 0x58}; //d
uint8_t broadcastAddress5[] = {0xD4, 0xD4, 0xDA, 0x46, 0x68, 0xC4}; //e

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4

typedef struct struct_message {
  uint8_t disdata[200];
  char path[10];
  int count;
  int length;
} struct_message;


struct_message receivedData;

esp_now_peer_info_t peerInfo;

void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  memcpy(&receivedData, data, sizeof(struct_message));
  Serial.print("Bytes Received: ");
  Serial.println(data_len);

  //int length = sizeof(receivedData.disdata) / sizeof(uint8_t);

  // Print the received data
  Serial.print("Received disdata: ");
  for(int i=0; i <receivedData.length;i++){
    Serial.print((char)receivedData.disdata[i]);
  }
  Serial.println();

  for(int j = 0; j<10; j++){
    Serial.print(receivedData.path[j]);
  }
  Serial.println();
  Serial.println(receivedData.count);

  receivedData.count++;
  Serial.println(receivedData.count);
  if ((char)receivedData.disdata[receivedData.count]=='B'){
      esp_err_t result1 = esp_now_send(broadcastAddress2, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result1==ESP_OK){
            Serial.println("Sent with success to A2");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else if (receivedData.disdata[receivedData.count]=='C'){
      esp_err_t result2 = esp_now_send(broadcastAddress2, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result2==ESP_OK){
            Serial.println("Sent with success to A3");
      }
      else{
          Serial.println("Error sending data");
      }
  }
```

```
  else if (receivedData.disdata[receivedData.count]=='D'){
      esp_err_t result3 = esp_now_send(broadcastAddress2, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result3==ESP_OK){
            Serial.println("Sent with success to A4");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else if (receivedData.disdata[receivedData.count]=='E'){
      esp_err_t result4 = esp_now_send(broadcastAddress2, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result4==ESP_OK){
            Serial.println("Sent with success to A5");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else{
    Serial.println("Invalid conditions!");
    Serial.print(receivedData.path[0]);
  }
}


// Connection pins
const uint8_t PIN_RST = 27; // Reset pin
const uint8_t PIN_IRQ = 34; // IRQ pin
const uint8_t PIN_SS = 4;   // SPI select pin

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);


  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress4, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress5, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  delay(1000);

  // Initialize DW1000 module
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
  DW1000Ranging.attachNewRange(newRange);
  DW1000Ranging.attachBlinkDevice(newBlink);
  DW1000Ranging.attachInactiveDevice(inactiveDevice);

  // Start the module as an anchor
  DW1000Ranging.startAsAnchor(ANCHOR_ADD, DW1000.MODE_LONGDATA_RANGE_LOWPOWER, false);
}

void loop() {
  DW1000Ranging.loop();
}

void newRange() {
  Serial.print("From: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);
  Serial.print("\tRange: ");
```

```
  Serial.print(DW1000Ranging.getDistantDevice()->getRange());
  Serial.print(" m");
  Serial.print("\tRX power: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
  Serial.println(" dBm");
}

void newBlink(DW1000Device *device) {
  Serial.print("Blink; 1 device added! -> ");
  Serial.print("Short: ");
  Serial.println(device->getShortAddress(), HEX);
}

void inactiveDevice(DW1000Device *device) {
  Serial.print("Delete inactive device: ");
  Serial.println(device->getShortAddress(), HEX);
}
```

```
#include <SPI.h>
#include <DW1000Ranging.h>
#include <WiFi.h>
#include "link.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <esp_now.h>
#include <math.h>
#include <string.h>
#include <float.h>

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4
#define PIN_RST 27
#define PIN_IRQ 34
#define ONE_WIRE_BUS 14
float d1 = 100; // dist between tag and A1
float d2 = 100; //distance between tag and A2
float d3 = 100; //distance between tag and A3
float d4 = 100; //distance between tag and A4

float x; //coordinates for tag
float y;

int tempadd;

int a1 = 6019;  // address of A1 1783
int a2 = 6018;  // address of A2 1782
int a3 = 6017;  // address of A3 1781
int a4 = 6016;  // address of A4 1780

float smallest1 = FLT_MAX;
float smallest2 = FLT_MAX;

float ax=0, ay=0, bx=1, by=0, cx=0, cy=1, dx=1, dy=1;


char nodes[] = {'A', 'B', 'C', 'D', 'E'};
float distances[5][5] = {
    {0, 1, 1, 1.42, 2.06},
    {1, 0, 1.42, 1, 2.06},
    {1, 1.42, 0, 1, 1.12},
    {1.42, 1, 1, 0, 1.12},
    {2.06, 2.06, 1.12, 1.12, 0}
};

uint8_t broadcastAddress1[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xFC};   // to A1
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0};   // to A2
uint8_t broadcastAddress3[] = {0x54, 0x43, 0xB2, 0x7F, 0x56, 0xBC};   // to A3
uint8_t broadcastAddress4[] = {0xD4, 0xD4, 0xDA, 0x46, 0x70, 0x58};   // to A4
uint8_t broadcastAddress5[] = {0xD4, 0xD4, 0xDA, 0x46, 0x68, 0xC4};   // to A5

struct MyLink *uwb_data;
int index_num = 0;
long runtime = 0;
String all_json = "";

typedef struct algdata {
  uint8_t disdata[200];
  char path[10];
  int count;
  int length;
} algdata;
```

```
algdata data;

// DS18B20 sensor setup
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
esp_now_peer_info_t peerInfo;

char first_node[1];

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{
    Serial.begin(115200);

    // Connect to WiFi
    WiFi.mode(WIFI_STA);

   if (esp_now_init() != ESP_OK){
     Serial.println("Error Initializing ESP-NOW");
     return;
   }

  esp_now_register_send_cb(OnDataSent);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }


  delay(1000);

    // Initialize DW1000 module
    SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
    DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
    DW1000Ranging.attachNewRange(newRange);
    DW1000Ranging.attachNewDevice(newDevice);
    DW1000Ranging.attachInactiveDevice(inactiveDevice);

    // Start the module as a tag
    DW1000Ranging.startAsTag("7D:00:22:EA:82:60:3B:9C", DW1000.MODE_LONGDATA_RANGE_LOWPOWER);

    // Initialize the link data structure
    uwb_data = init_link();

    // Initialize the DS18B20 sensor
    sensors.begin();

    /*data.path[0]=6019;
    data.path[1]=6018;
    data.path[2]=6017;*/
    data.count=-1;

}

void loop()
{

    DW1000Ranging.loop();
    if ((millis() - runtime) > 1000)
    {
        // Read temperature from the DS18B20 sensor
        sensors.requestTemperatures();
        float temperature = sensors.getTempCByIndex(0);

        // Create JSON string with range and temperature data
        make_link_json(uwb_data, &all_json);
        addTemperatureToJSON(&all_json, temperature);
```

```
        // Convert all_json to a byte array

        /*uint8_t all_json_bytes[all_json.length()];
        all_json.getBytes(all_json_bytes, all_json.length());*/
        uint8_t all_json_bytes[all_json.length() + 1];  // Add 1 for null terminator
        strcpy((char*)all_json_bytes, all_json.c_str());  // Convert string to byte array
        int length_of_arr = sizeof(all_json_bytes);
        memcpy(data.disdata, all_json_bytes, sizeof(all_json_bytes));
        data.length=length_of_arr;
        Serial.print("Length of all_json_bytes is : ");
        Serial.println(data.length);
        Serial.print("The starting node is:");
        Serial.println(first_node);

      if (d1<d2 && d1<d3){ //to A1
          esp_err_t result1 = esp_now_send(broadcastAddress1, (uint8_t *) &data, sizeof(data));

          if (result1==ESP_OK){
            Serial.println("Sent with success to A1");
          }
          else{
          Serial.println("Error sending data");
          }
      }
      else if (d2<d1 && d2<d3){ // to A2
          esp_err_t result2 = esp_now_send(broadcastAddress2, (uint8_t *) &data, sizeof(data));

          if (result2==ESP_OK){
            Serial.println("Sent with success to A2");
          }
          else{
            Serial.println("Error sending data");  // in case if failure send data to A1, potential for recursive function?? try until
          }
      }

      else if (d3<d1 && d3<d2){ // to A3
          esp_err_t result3 = esp_now_send(broadcastAddress3, (uint8_t *) &data, sizeof(data));

          if (result3==ESP_OK){
            Serial.println("Sent with success to A3");
          }
          else{
            Serial.println("Error sending data");  // in case if failure send data to A1, potential for recursive function?? try until
            Serial.println(result3);
          }
      }
        Serial.println(all_json);
        /*for(int i=0; i<sizeof(all_json_bytes); i++){
          Serial.println(*ptr+i);
        }*/
        delay(1000);
        runtime = millis();
    }
}

void newRange()
{
    Serial.print("From: ");
    tempadd=(DW1000Ranging.getDistantDevice()->getShortAddress());
    Serial.print(tempadd);
    Serial.print("\tRange: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRange());
    Serial.print(" m");
    Serial.print("\tRX power: ");
    Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
    Serial.println(" dBm");
    float temperature = sensors.getTempCByIndex(0);
    Serial.println("\tTemperature : ");
    Serial.print(temperature);
    Serial.println(" C");
    fresh_link(uwb_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), temperatur
    if (tempadd==a1){
      d1=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d1);
    }
    else if (tempadd==a2){
      d2=DW1000Ranging.getDistantDevice()->getRange();
      //Serial.println(d2);
    }
    else if (tempadd==a3){
      d3=DW1000Ranging.getDistantDevice()->getRange();
    }
    else if (tempadd==a4){
      d4=DW1000Ranging.getDistantDevice()->getRange();
    }

    findPoint(ax,ay,d1,bx,by,d2,cx,cy,d3);
```

```
    func(x,y);
}

void newDevice(DW1000Device *device)
{
    Serial.print("Ranging init; 1 device added! -> ");
    Serial.print("Short: ");
    Serial.println(device->getShortAddress(), HEX);

    add_link(uwb_data, device->getShortAddress());
}

void inactiveDevice(DW1000Device *device)
{
    Serial.print("Delete inactive device: ");
    Serial.println(device->getShortAddress(), HEX);

    delete_link(uwb_data, device->getShortAddress());
}

void findPoint(float x1, float y1, float d1, float x2, float y2, float d2, float x3, float y3, float d3) {


    float A = 2 * (x2 - x1);
    float B = 2 * (y2 - y1);
    float C = pow(d1, 2) - pow(d2, 2) - pow(x1, 2) + pow(x2, 2) - pow(y1, 2) + pow(y2, 2);

    float D = 2 * (x3 - x2);
    float E = 2 * (y3 - y2);
    float F = pow(d2, 2) - pow(d3, 2) - pow(x2, 2) + pow(x3, 2) - pow(y2, 2) + pow(y3, 2);

    x = (C*E - F*B) / (E*A - B*D);
    y = (C*D - A*F) / (B*D - A*E);

    //printf("The point is (%lf, %lf)\n", x, y);
}


void dijkstra(char current, char nodes[], float distances[5][5], float *visited, char *prevNode) {
    float unvisited[5];
    float currentDistance = 0;
    int i, j;

    for (i = 0; i < 5; i++) {
        unvisited[i] = INFINITY;
        visited[i] = INFINITY;
        prevNode[i] = '\0';  // Initialize previous node as null character
    }

    unvisited[current - 'A'] = 0;

    while (1) {
        for (j = 0; j < 5; j++) {
            char neighbour = nodes[j];
            float distance = distances[current - 'A'][j];
            if (distance == 0 || visited[j] <= currentDistance + distance)
                continue;

            float newDistance = currentDistance + distance;
            int neighbourIndex = neighbour - 'A';
            if (unvisited[neighbourIndex] == INFINITY || unvisited[neighbourIndex] > newDistance) {
                unvisited[neighbourIndex] = newDistance;
                prevNode[neighbourIndex] = current;  // Set previous node for the neighbor
            }
        }

        visited[current - 'A'] = currentDistance;
        unvisited[current - 'A'] = INFINITY;

        int done = 1;
        float minDistance = INFINITY;
        char nextNode;
        for (i = 0; i < 5; i++) {
            if (unvisited[i] < INFINITY) {
                done = 0;
                if (unvisited[i] < minDistance) {
                    minDistance = unvisited[i];
                    nextNode = nodes[i];
                }
            }
        }

        if (done)
            break;
        current = nextNode;
        currentDistance = minDistance;
```

```
    }
}

void func(float x, float y) {
    float ad = sqrt(pow((x - 100), 2) + pow((y - 100), 2));
    //ad = round(ad * 10) / 10;
    float bd = sqrt(pow((x - 500), 2) + pow((y - 100), 2));
    //bd = round(bd * 10) / 10;
    float cd = sqrt(pow((x - 500), 2) + pow((y - 500), 2));
    //cd = round(cd * 10) / 10;
    float dd = sqrt(pow((x - 100), 2) + pow((y - 500), 2));
    //dd = round(dd * 10) / 10;
Serial.print(ad);
Serial.print(bd);
Serial.print(cd);
Serial.print(dd);
    char start_node;
    if (ad <= bd && ad <= cd && ad <= dd)
        start_node = 'A';
    else if (bd <= ad && bd <= cd && bd <= dd)
        start_node = 'B';
    else if (cd <= ad && cd <= bd && cd <= dd)
        start_node = 'C';
    else
        start_node = 'D';

    const char* start_nodeptr = &start_node;

    float visited[5];
    char prevNode[5];
    dijkstra(start_node, nodes, distances, visited, prevNode);
    int counter = 0;
    Serial.print("Shortest distance to E: ");
    Serial.println(visited['E' - 'A']);
    Serial.print("Path to E: ");
    counter=printPath(start_node, 'E', prevNode, counter);
    Serial.print("\n");
    Serial.print(start_node);
}

int printPath(char start, char end, char *prevNode, int counter) {
    if (start == end) {
        Serial.print(start);
        data.path[0]=start;
        return 0;
    }
    counter=printPath(start, prevNode[end - 'A'], prevNode, counter);
    data.path[counter+1]=end;
    Serial.print(end);
    counter++;
    return counter;
}


void addTemperatureToJSON(String *json, float temperature)
{
    *json = json->substring(0, json->length() - 1); // Remove the closing brace
    *json += ",\"temperature\": " + String(temperature);
    *json += "}";
}
```

```
#include <SPI.h>
#include "DW1000Ranging.h"
#include <esp_now.h>
#include <WiFi.h>
#include <ArduinoJson.h>

#define ANCHOR_ADD "80:17:5B:D5:A9:9A:E2:9C" //for anchor A4
uint8_t broadcastAddress1[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xFC};   // to A1
uint8_t broadcastAddress2[] = {0xC8, 0xF0, 0x9E, 0xBE, 0xFA, 0xB0};   // to A2
uint8_t broadcastAddress3[] = {0x54, 0x43, 0xB2, 0x7F, 0x56, 0xBC};   // to A3
uint8_t broadcastAddress5[] = {0xD4, 0xD4, 0xDA, 0x46, 0x72, 0x74}   // to A5

#define SPI_SCK 18
#define SPI_MISO 19
#define SPI_MOSI 23
#define DW_CS 4

typedef struct struct_message {
  uint8_t disdata[200];
```

```
  char path[10];
  int count;
  int length;
  char s_node;
} struct_message;


struct_message receivedData;

esp_now_peer_info_t peerInfo;

void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  memcpy(&receivedData, data, sizeof(struct_message));
  Serial.print("Bytes Received: ");
  Serial.println(data_len);

  //int length = sizeof(receivedData.disdata) / sizeof(uint8_t);

  // Print the received data
  Serial.print("Received disdata: ");
  for(int i=0; i <receivedData.length;i++){
    Serial.print((char)receivedData.disdata[i]);
  }
  Serial.println();

  for(int j = 0; j<10; j++){
    Serial.print(receivedData.path[j]);
  }
  Serial.println();
  Serial.println(receivedData.count);

  receivedData.count++;
  Serial.println(receivedData.count);
  if ((char)receivedData.path[receivedData.count]=='B'){
      esp_err_t result1 = esp_now_send(broadcastAddress2, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result1==ESP_OK){
            Serial.println("Sent with success to A2");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else if (receivedData.path[receivedData.count]=='A'){
      esp_err_t result2 = esp_now_send(broadcastAddress1, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result2==ESP_OK){
            Serial.println("Sent with success to A1");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else if (receivedData.path[receivedData.count]=='C'){
      esp_err_t result3 = esp_now_send(broadcastAddress3, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result3==ESP_OK){
            Serial.println("Sent with success to A3");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else if (receivedData.path[receivedData.count]=='E'){
      esp_err_t result4 = esp_now_send(broadcastAddress5, (uint8_t *) &receivedData, sizeof(receivedData));
      if (result4==ESP_OK){
            Serial.println("Sent with success to A5");
      }
      else{
          Serial.println("Error sending data");
      }
  }
  else{
    Serial.println("Invalid conditions!");
    Serial.print(receivedData.path[0]);
  }
}


// Connection pins
const uint8_t PIN_RST = 27; // Reset pin
const uint8_t PIN_IRQ = 34; // IRQ pin
const uint8_t PIN_SS = 4;   // SPI select pin

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
```

```
    return;
  }

  esp_now_register_recv_cb(OnDataRecv);


  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress5, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  delay(1000);

  // Initialize DW1000 module
  SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);
  DW1000Ranging.initCommunication(PIN_RST, DW_CS, PIN_IRQ);
  DW1000Ranging.attachNewRange(newRange);
  DW1000Ranging.attachBlinkDevice(newBlink);
  DW1000Ranging.attachInactiveDevice(inactiveDevice);

  // Start the module as an anchor
  DW1000Ranging.startAsAnchor(ANCHOR_ADD, DW1000.MODE_LONGDATA_RANGE_LOWPOWER, false);
}

void loop() {
  DW1000Ranging.loop();
}

void newRange() {
  Serial.print("From: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);
  Serial.print("\tRange: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRange());
  Serial.print(" m");
  Serial.print("\tRX power: ");
  Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());
  Serial.println(" dBm");
}

void newBlink(DW1000Device *device) {
  Serial.print("Blink; 1 device added! -> ");
  Serial.print("Short: ");
  Serial.println(device->getShortAddress(), HEX);
}

void inactiveDevice(DW1000Device *device) {
  Serial.print("Delete inactive device: ");
  Serial.println(device->getShortAddress(), HEX);
}
```