**ARDUINO CODE**

#include <SPI.h>

#include <Ethernet.h>

#include <SD.h>

#include <Sensirion.h>

#define dataPin 8

#define clockPin 9

#define REQ\_BUF\_SZ 50

#define TF 2

#define EF 3

#define TB 5

#define EB 6

#define MOTOSPEED 2000 //Represents the delay

char HTTP\_req[REQ\_BUF\_SZ] = {0}; // buffered HTTP request stored as null terminated string

char req\_index = 0;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };// MAC address from Ethernet shield sticker under board

IPAddress ip(192, 168, 0, 25); // IP address, may need to change depending on network

EthernetServer server(80); // create a server at port 80

Sensirion tempSensor = Sensirion(dataPin, clockPin);

File webFile;

String Sensorvalue,Sht11\_val;

float tempC,humidity;

uint8\_t dangerFront=0; //WILL BE SET IF THERE IS SOMETHING IN FRONT

uint8\_t dangerBack=0; //WILL BE SET IF THERE IS SOMETHING IN BACK

const uint8\_t danger\_limit=30;

byte buf=0; //FOR STORING THE VALUES OF PORT WHILE USING THE MOTOR;

uint8\_t front,back,left,right; //CONTAINS THE GAP

void setup()

{

Ethernet.begin(mac, ip); // initialize Ethernet device

server.begin(); // start to listen for clients

Serial.begin(9600); // for debugging

// initialize SD card

Serial.println("Initializing SD card...");

if (!SD.begin(4)) {

Serial.println("ERROR - SD card initialization failed!");

return; // init failed

}

Serial.println("SUCCESS - SD card initialized.");

// check for index.htm file

if (!SD.exists("rishav.htm")) {

Serial.println("ERROR - Can't find rishav.htm file!");

return; // can't find index file

}

Serial.println("SUCCESS - Found rishav.htm file.");

pinMode(TF,OUTPUT);

pinMode(TB,OUTPUT);

pinMode(EF,INPUT);

pinMode(EB,INPUT);

DDRC=DDRC|B00111100;

Serial.println(DDRC);

}

void loop()

{

EthernetClient client = server.available(); // try to get client

front=getGap(TF,EF);

back=getGap(TB,EB);

checkdanger(front,back);

if (client) { // got client?

boolean currentLineIsBlank = true;

while (client.connected()) {

if (client.available()) { // client data available to read

char c = client.read(); // read 1 byte (character) from client

// last line of client request is blank and ends with \n

// respond to client only after last line received

if (req\_index < (REQ\_BUF\_SZ - 1)) {

HTTP\_req[req\_index] = c; // save HTTP request character

req\_index++;

}

if (c == '\n' && currentLineIsBlank) {

// send a standard http response header

client.println("HTTP/1.1 200 OK");

client.println("Content-Type: text/html");

client.println("Connection: keep-alive");

client.println();

// send web page

//Serial.println(HTTP\_req);

if (StrContains(HTTP\_req, "sensorval")) {

sendData(front,back,client);

front=getGap(TF,EF);

back=getGap(TB,EB);

checkdanger(front,back);

}

else if (StrContains(HTTP\_req, "forward")) {

client.println("FORWARD");

Forward();

}

else if (StrContains(HTTP\_req, "backward")) {

client.println("BACKWARD");

Backward();

}

else if (StrContains(HTTP\_req, "left")) {

client.println("LEFT");

Left();

}

else if (StrContains(HTTP\_req, "right")) {

client.println("RIGHT");

Right();

}

else if(StrContains(HTTP\_req, "temp\_humid"))

{

getHumid\_data(humidity,tempC,client);

}

else{

webFile = SD.open("rishav.htm"); // open web page file

if (webFile) {

while(webFile.available()) {

client.write(webFile.read()); // send web page to client

}

webFile.close();

}

}

req\_index = 0;

StrClear(HTTP\_req, REQ\_BUF\_SZ);

break;

}

// every line of text received from the client ends with \r\n

if (c == '\n') {

// last character on line of received text

// starting new line with next character read

currentLineIsBlank = true;

}

else if (c != '\r') {

// a text character was received from client

currentLineIsBlank = false;

}

} // end if (client.available())

} // end while (client.connected())

delay(1); // give the web browser time to receive the data

client.stop(); // close the connection

} // end if (client)

}

void StrClear(char \*str, char length)

{

for (int i = 0; i < length; i++) {

str[i] = 0;

}

}

char StrContains(char \*str, char \*sfind)

{

char found = 0;

char index = 0;

char len;

len = strlen(str);

if (strlen(sfind) > len) {

return 0;

}

while (index < len) {

if (str[index] == sfind[found]) {

found++;

if (strlen(sfind) == found) {

return 1;

}

}

else {

found = 0;

}

index++;

}

return 0;

}

long getGap(int T,int E)

{

long duration;

long distance;

digitalWrite(T, LOW); // Added this line

delayMicroseconds(2); // Added this line

digitalWrite(T, HIGH);

delayMicroseconds(10); // Added this line

digitalWrite(T, LOW);

duration = pulseIn(E, HIGH);

distance = (duration/2) / 29.1;

return distance;

}

void checkdanger(uint8\_t front,uint8\_t back)

{

if(front<=danger\_limit)

{dangerFront=1;}

else

{dangerFront=0;}

if(back<=danger\_limit)

{dangerBack=1;}

else

{dangerBack=0;}

}

void Forward()

{

Serial.println("FORWARD");

if(!dangerFront){Serial.println(PORTC);

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

buf=buf|B00010100; //SETTING THE MIDDLE FOUR BITS AS PER MOVEMENT REQUIREMENT

PORTC=buf; //LOADING THE DATA TO PORT

Serial.println(PORTC);

delay(MOTOSPEED);//WAIT FOR THE MOTOR TO ROTATE

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

PORTC=buf;

Serial.println(PORTC);

}

}

void Backward()

{

if(!dangerBack){

Serial.println("BACK");

Serial.println(PORTC);

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

buf=buf|B00101000; //SETTING THE MIDDLE FOUR BITS AS PER MOVEMENT REQUIREMENT

PORTC=buf; //LOADING THE DATA TO PORT

Serial.println(PORTC);

delay(MOTOSPEED);//WAIT FOR THE MOTOR TO ROTATE

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

PORTC=buf;

Serial.println(PORTC);

}

}

void Left() //MAKES IT MOVE LEFT CIRCLE

{

Serial.println("LEFT");

Serial.println(PORTC);

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

buf=buf|B00100100; //SETTING THE MIDDLE FOUR BITS AS PER MOVEMENT REQUIREMENT

PORTC=buf; //LOADING THE DATA TO PORT

Serial.println(PORTC);

delay(MOTOSPEED);//WAIT FOR THE MOTOR TO ROTATE

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

PORTC=buf;

Serial.println(PORTC);

}

void Right() //MAKES IT MOVE RIGHT CIRCLE

{ Serial.println("RIGHT");

Serial.println(PORTC);

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

buf=buf|B00011000; //SETTING THE MIDDLE FOUR BITS AS PER MOVEMENT REQUIREMENT

PORTC=buf; //LOADING THE DATA TO PORT

Serial.println(PORTC);

delay(MOTOSPEED);//WAIT FOR THE MOTOR TO ROTATE

buf=B11000011&PORTC; //MASK THE FIRST AND LAST TWO BITS AND CLEARING THE MIDDLE 4 BITS FOR NEW OUTPUT

PORTC=buf;

Serial.println(PORTC);

}

void sendData(uint8\_t front,uint8\_t back, EthernetClient cl)

{

Sensorvalue=front;

Sensorvalue+=",";

Sensorvalue+=back;

cl.println(Sensorvalue);

}

void getHumid\_data(float humid,float temp,EthernetClient cl)

{ float dewpoint;

tempSensor.measure(&temp, &humid, &dewpoint);

Sht11\_val=humid;

Sht11\_val+=",";

Sht11\_val+=temp;

cl.println(Sht11\_val);

}