THOMAS MORE

DOCUMENTATIE

Line Following Robot

Quentin & Zaid

2023 - 2024

Inhoud

Projectbeschrijving

Mercedes-Benz wil het bezoekersproces in hun museum in Stuttgart automatiseren door middel van zelfrijdende voertuigen. Bezoekers kunnen aan het begin van de rondleiding instappen in een autonoom wagentje, dat hen door het museum leidt en aan het einde van de tour weer laat uitstappen. Vervolgens rijdt het wagentje automatisch terug naar het startpunt voor de volgende bezoeker.

De voertuigen volgen een vooraf bepaalde lijn op de vloer en vermijden obstakels op hun route. Daarnaast kan een bezoeker op elk moment het voertuig tijdelijk stoppen met een drukknop. Optioneel willen ze de mogelijkheid toevoegen om de voertuigen op afstand te bedienen via een mobiele applicatie.

De voertuigen worden aangedreven door een batterij en sturen hun batterijstatus en afstandsmetingen **continu via MQTT** door naar een centraal monitoringsysteem. Dit systeem registreert de data in een database en visualiseert deze via een **webdashboard**.

Gebruikte Hardware

Microcontrolers

- Arduino Uno
- ESP32

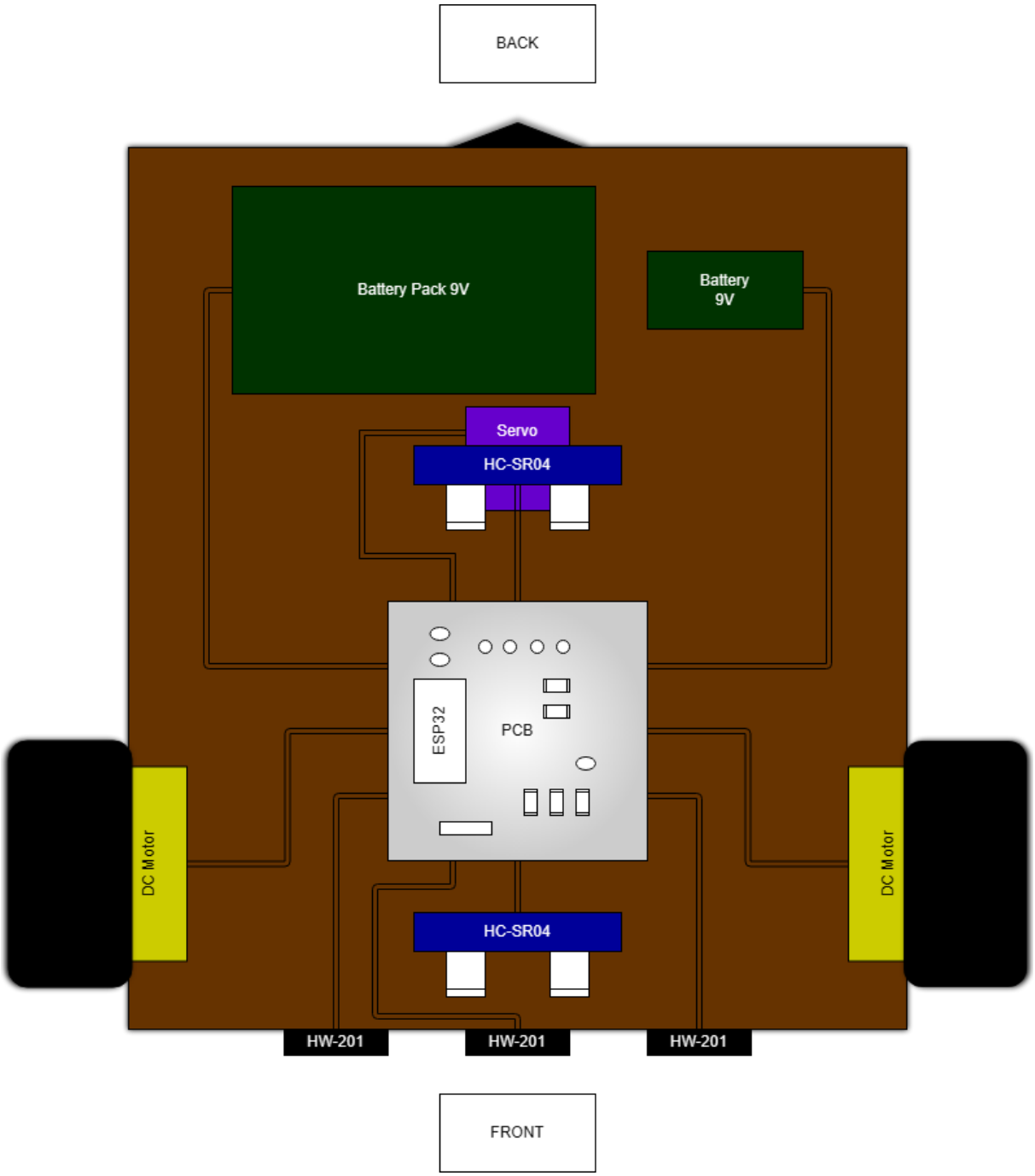
Sensoren

- 2 x HC-SR04
- 3 x HW-201

Overige componenten

- 2 x 9V batterij
- 1 x 1602A LCD
- 1 x Button
- 2 x DC Motor
- 1 x Ball wheel
- 25 x Connection cables
- # x Connectors (PCB)
- # x Connection Pins (PCB)
- 1 x L293D Motor Driver
- 1 x 9V DC Transformator
- 1 x 10µF Capacitor
- 1 x 1µF Capacitor
- 1 x Battery Header
- Wood (Chassis)
- Nails (Chassis)
- Screws (Chassis)
- Nuts & Bolts (M3, 12mm ←→)

Schema


BACK
Battery Pack 9V
Battery
9V
Servo
HC-SR04
ESP32
PCB
DC Motor
DC Motor
HC-SR04
HW-201
HW-201
HW-201
FRONT

Code

Arduino Uno + Motor shield

```
/*
  Auteur: Quentin Costermans & Zaid Ben
  Project: Line-following robot met obstakeldetectie & ontwijkingsmanoeuvres
  Voor: IoT 2025
  School: Thomas More - Graduaat IoT 2024-2025
*/

#include <Wire.h>
#include <AFMotor.h>
#include <Servo.h>

// IR sensoren voor lijnvolging
#define irLeft A0
#define irCenter A1
#define irRight A2

// Ultrasone sensoren voor afstandsmeting
#define trigPin 9
#define echoPin 10
#define trigPin2 A4
#define echoPin2 A5

// Knop voor pauzeren/starten
#define buttonPin A6

// Servo voor obstakel-scanning
#define servoPin A3

// Motoren
AF_DCMotor motor1(1, MOTOR12_1KHZ);
AF_DCMotor motor2(2, MOTOR12_1KHZ);
Servo scannerServo;

// Debounce logica & pauze status
bool paused = true;
unsigned long lastButtonPress = 0;
const unsigned long debounceDelay = 50;
bool lastButtonState = HIGH;
bool buttonState = HIGH;
unsigned long lastDebounceTime = 0;

// Setup: initialisatie van pinnen, motoren en sensoren
void setup() {
  Serial.begin(9600);

  pinMode(irLeft, INPUT);
  pinMode(irCenter, INPUT);
  pinMode(irRight, INPUT);
  pinMode(trigPin, OUTPUT);
```

```
  pinMode(echoPin, INPUT);
  pinMode(trigPin2, OUTPUT);
  pinMode(echoPin2, INPUT);
  pinMode(buttonPin, INPUT_PULLUP);

  scannerServo.attach(servoPin);
  scannerServo.write(90); // beginpositie (recht vooruit)

  Serial.println("Setup complete. Waiting for button to start...");
}

// Main loop: bepaalt gedrag op basis van sensorinput
void loop() {
  checkButton();
  if (paused) {
    stopMotors();
    return;
  }

  if (obstacleDetected()) {
    Serial.println("Obstacle detected. Starting avoidance maneuver...");
    avoidSmartObstacle();
  } else {
    followLine();
  }
}

// Meet afstand in cm via ultrasone sensor
long readDistance(int trig, int echo) {
  digitalWrite(trig, LOW); delayMicroseconds(2);
  digitalWrite(trig, HIGH); delayMicroseconds(10);
  digitalWrite(trig, LOW);
  long duration = pulseIn(echo, HIGH, 20000);
  long distance = duration * 0.034 / 2;
  Serial.print("Ultrasonic Distance (Trig: ");
  Serial.print(trig);
  Serial.print("): ");
  Serial.print(distance);
  Serial.println(" cm");
  return distance;
}

// Draaien servo en meten afstand in een specifieke richting
long readScannerAtAngle(int angle) {
  scannerServo.write(angle);
  delay(600);
  long d = readDistance(trigPin2, echoPin2);
  Serial.print("Scanner angle ");
  Serial.print(angle);
  Serial.print("° distance: ");
  Serial.print(d);
  Serial.println(" cm");
  return d;
}

// Detecteert of er een obstakel recht voor de robot staat
```

```
bool obstacleDetected() {
  long d = readDistance(trigPin, echoPin);
  return (d > 0 && d < 20);
}

// Obstacle ontwijking: slimme logica om obstakel te scannen en te omzeilen
void avoidSmartObstacle() {
  stopMotors(); delay(300);

  long leftDist = readScannerAtAngle(180);
  long rightDist = readScannerAtAngle(0);
  bool goLeft = leftDist > rightDist;
  readScannerAtAngle(90);

  Serial.print("Chosen path: ");
  Serial.println(goLeft ? "LEFT" : "RIGHT");

  if (goLeft) {
    Serial.println("Turning Left");
    turnLeft(); delay(500);
    scannerServo.write(0); delay(500);
  } else {
    Serial.println("Turning Right");
    turnRight(); delay(500);
    scannerServo.write(180); delay(500);
  }

  stopMotors(); delay(300);

  while (true) {
    long sideScan = readDistance(trigPin2, echoPin2);
    if (sideScan > 40 && readDistance(trigPin2, echoPin2) > 20) {
      Serial.println("Detected end of obstacle.");
      moveForward();
      delay(400);
      break;
    }
    if (obstacleDetected()) {
      stopMotors(); delay(300);
      return;
    }
    Serial.println("Moving along the obstacle...");
    moveForward();
    delay(100);
  }

  stopMotors(); delay(300);
  Serial.println("Moving forward to bypass obstacle...");
  moveForward(); delay(600);
  stopMotors(); delay(300);

  if (goLeft) {
    Serial.println("Turning to pass in front of obstacle (right turn)");
    turnRight(); delay(500);
  } else {
    Serial.println("Turning to pass in front of obstacle (left turn)");
```

```
    turnLeft(); delay(500);
  }

  Serial.println("Moving forward to bypass obstacle...");
  moveForward(); delay(600);
  stopMotors(); delay(300);

  while (true) {
    long sideScan = readDistance(trigPin2, echoPin2);
    if (sideScan > 40 && readDistance(trigPin2, echoPin2) > 20) {
      Serial.println("Detected end of obstacle.");
      moveForward();
      delay(200);
      break;
    }

    if (obstacleDetected()) {
      stopMotors(); delay(300);
      return;
    }

    Serial.println("Moving along the obstacle...");
    moveForward();
    delay(100);
  }

  Serial.println("Moving forward to bypass obstacle...");
  moveForward(); delay(600);
  stopMotors(); delay(300);

  if (goLeft) {
    Serial.println("Final right turn to realign with path");
    turnRight(); delay(500);
  } else {
    Serial.println("Final left turn to realign with path");
    turnLeft(); delay(500);
  }

  while (true) {
    if (digitalRead(irLeft) || digitalRead(irCenter) || digitalRead(irRight)) {
      break;
    }

    if (obstacleDetected()) {
      stopMotors(); delay(300);
      return;
    }

    Serial.println("Moving along the obstacle...");
    moveForward();
    delay(100);
  }

  scannerServo.write(90); delay(500);
  Serial.println("Returned to line-following position.");
}
```

```
// Volgt een zwarte lijn m.b.v. IR sensoren
void followLine() {
  int left = digitalRead(irLeft);
  int center = digitalRead(irCenter);
  int right = digitalRead(irRight);

  Serial.print("IR Readings - L:");
  Serial.print(left);
  Serial.print(" C:");
  Serial.print(center);
  Serial.print(" R:");
  Serial.println(right);

  if (center && !left && !right) {
    Serial.println("→ Moving Forward (center on line)");
    moveForward();
  } else if (left && !center && !right) {
    Serial.println("↖ Adjust Left");
    turnLeft();
  } else if (right && !center && !left) {
    Serial.println("↗ Adjust Right");
    turnRight();
  } else if (center && left && !right) {
    Serial.println("↖ Slight Left");
    turnSlightLeft();
  } else if (center && right && !left) {
    Serial.println("↗ Slight Right");
    turnSlightRight();
  } else {
    Serial.println("Lost Line - Stopping");
    stopMotors();
  }
}

// Bewegingen robot
void moveForward() {
  motor1.run(FORWARD); motor1.setSpeed(130);
  motor2.run(FORWARD); motor2.setSpeed(130);
}

void turnLeft() {
  motor1.run(FORWARD); motor1.setSpeed(150);
  motor2.run(BACKWARD); motor2.setSpeed(150);
}

void turnRight() {
  motor1.run(BACKWARD); motor1.setSpeed(150);
  motor2.run(FORWARD); motor2.setSpeed(150);
}

void turnSlightLeft() {
  motor1.run(FORWARD); motor1.setSpeed(120);
  motor2.run(BACKWARD); motor2.setSpeed(120);
}
```

```
void turnSlightRight() {
  motor1.run(BACKWARD); motor1.setSpeed(120);
  motor2.run(FORWARD); motor2.setSpeed(120);
}

void stopMotors() {
  Serial.println("Motors stopped.");
  motor1.run(RELEASE); motor1.setSpeed(0);
  motor2.run(RELEASE); motor2.setSpeed(0);
}

// Controleert of de start/pauze-knop werd ingedrukt
void checkButton() {
  int reading = digitalRead(buttonPin);
  if (reading != lastButtonState) lastDebounceTime = millis();
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == LOW) {
        paused = !paused;
        Serial.print("Robot is now ");
        Serial.println(paused ? "PAUSED" : "RUNNING");
        if (paused) stopMotors();
      }
    }
  }
  lastButtonState = reading;
}
```

Functieoverzicht en uitleg

setup()

Deze functie wordt éénmalig uitgevoerd bij het opstarten van de Arduino.
Ze initialiseert de seriële communicatie, configureert de pinnen als input/output, stelt de servo in de beginpositie (90° = vooruit), en drukt een statusbericht naar de seriële monitor.

loop()

De hoofdloop van het programma, die telkens opnieuw wordt uitgevoerd.

- Eerst wordt checkButton() aangeroepen om te kijken of de gebruiker op de start/pauze-knop heeft gedrukt.
- Als de robot gepauzeerd is, stopt hij de motoren.
- Indien niet gepauzeerd, controleert hij of er een obstakel is met obstacleDetected().
- Als er een obstakel is, voert hij avoidSmartObstacle() uit.
- Anders volgt hij de lijn via followLine().

readDistance(int trig, int echo)

Stuurt een puls uit via de trig pin van een ultrasone sensor en leest via de echo pin de tijd tot terugkeer.
Hiermee wordt de afstand tot een object berekend in centimeter.
Deze functie werkt voor beide ultrasone sensoren (vooraan en zijkant).

readScannerAtAngle(int angle)

Draait de servo naar een bepaalde hoek (0° = rechts, 180° = links) en roept dan readDistance() op met de tweede ultrasone sensor.
Wordt gebruikt om afstanden links en rechts van de robot te meten bij obstakeldetectie.

obstacleDetected()

Meet de afstand met de frontale ultrasone sensor en controleert of er een obstakel dichter dan 20 cm voor de robot staat.
Retourneert true als dat zo is, anders false.

avoidSmartObstacle()

Deze functie voert een volledige obstakel-ontwijking uit:

1. Scant links en rechts met de servo.
2. Bepaalt welke kant meer ruimte biedt.
3. Voert een draai uit in de gekozen richting.
4. Volgt langs het obstakel tot het einde wordt gedetecteerd.
5. Maakt een omleiding om voorbij het obstakel te geraken.
6. Lijnt zich terug uit met de oorspronkelijke lijn.

followLine()

Leest de waarden van de 3 IR-sensoren: links, midden en rechts.
Afhankelijk van de combinatie van detecties bepaalt de functie of de robot:

- Vooruit moet rijden
- Een bocht links of rechts moet maken
- Een lichte correctie moet uitvoeren
- Of gestopt moet worden (indien de lijn kwijt is)

De robot leest de IR-sensoren en handelt volgens deze logica:

Left	Center	Right	Action
0	1	0	Forward
1	0	0	Turn Left
0	0	1	Turn Right
1	1	0	Left Bias
0	1	1	Right Bias
1	1	1	Forward
0	0	0	Stop

moveForward()

Laat beide motoren vooruit draaien met een constante snelheid van 130 (PWM).
Wordt gebruikt voor rechtdoor rijden.

turnLeft()

Draait de linker motor vooruit en de rechter motor achteruit, beide op hogere snelheid.
Daardoor maakt de robot een bocht naar links.

turnRight()

Draait de rechter motor vooruit en de linker motor achteruit, voor een bocht naar rechts.

turnSlightLeft()

Voert een lichtere draai naar links uit met tragere snelheid.
Handig voor kleine aanpassingen bij lijnvolging.

turnSlightRight()

Idem als hierboven maar dan naar rechts.

stopMotors()

Stopt beide motoren volledig (snelheid op 0 en richting op RELEASE).
Drukt ook een melding naar de seriële monitor.

checkButton()

Debounce-functie voor de knop:
Controleert of er een stabiele wijziging is in knopstatus.
Indien op de knop wordt gedrukt, wordt paused omgeschakeld (start of pauzeer robot).
Indien gepauzeerd: motoren stoppen onmiddellijk.

ESP32 + MQTT (Simulation)

```
/*
  Auteur: Quentin Costermans & Zaid Ben
  Project: Simulatie & monitoring van ESP32 robot met MQTT & InfluxDB logging
  Voor: IoT 2025
  School: Thomas More - Graduaat IoT 2024-2025
*/

#include <WiFi.h>
#include <PubSubClient.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>

#define DEVICE "ESP32SimBot"
```

```
// WiFi-netwerkinstellingen
const char* ssid = "bbox4-71c2";
const char* password = "13620158";

// MQTT-serverinstellingen
const char* mqtt_server = "quecos.local";
const char* mqtt_user = "linepilot";
const char* mqtt_pass = "stayinline";
WiFiClient espClient;
PubSubClient client(espClient);

// InfluxDB-instellingen
#define INFLUXDB_URL "http://192.168.0.113:8086"
#define INFLUXDB_TOKEN
"EpSkGXMgIK67uM9Moe9TmuXjv_h5ZIqg57Pz4R_PSKna_LqKcnVVzrhsZ6Rw7nyR9mGn5AuJ-
mTN4d-Ee7v9Qw=="
#define INFLUXDB_ORG "102b7832fd343794"
#define INFLUXDB_BUCKET "linerobotbucket"

InfluxDBClient influxClient(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET,
INFLUXDB_TOKEN);
Point sensor("robot_data");

// Aangesloten pinnen voor motoren, IR, batterij, ultrasoon en knop
const int motorLeftFwd = 5;
const int motorLeftBwd = 18;
const int motorRightFwd = 19;
const int motorRightBwd = 21;
const int irLeft = 34;
const int irCenter = 32;
const int irRight = 35;
const int batteryPin = 36;
const int trigPin = 25;
const int echoPin = 26;
const int buttonPin = 27;

// Batterijspanningsmeting via spanningsdeler
const float R1 = 30000.0;
const float R2 = 7500.0;
const float ADC_MAX = 4095.0;
const float ADC_VREF = 3.3;

// Mogelijke bewegingstoestanden
const char* movements[] = {"forward", "turnLeft", "turnRight", "stopped"};

const long interval = 1000; // Log-interval in ms
unsigned long lastMsg = 0;
bool robotShutdown = false;

int button = 0;
bool buttonChanged = false;

// Verbindt met WiFi-netwerk
void setup_wifi() {
  Serial.print("Connecting to "); Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
}

// Verbindt met MQTT-server
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32SimBot", mqtt_user, mqtt_pass)) {
      client.subscribe("robot/control");
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(1000);
    }
  }
}

// Stuurt statusinformatie over WiFi- en MQTT-verbinding naar InfluxDB
void sendConnectivityStatus() {
  Point connStatus("robot_data");
  connStatus.addTag("device", DEVICE);
  connStatus.addTag("SSID", WiFi.SSID());
  connStatus.addField("wifi_state", (WiFi.status() == WL_CONNECTED) ? 1 : 0);
  connStatus.addField("mqtt_state", client.connected() ? 1 : 0);
  influxClient.writePoint(connStatus);
}

// Handler voor inkomende MQTT-berichten
void callback(char* topic, byte* payload, unsigned int length) {
  String msg;
  for (int i = 0; i < length; i++) msg += (char)payload[i];
  Serial.print("MQTT message: ");
  Serial.println(msg);

  if (msg == "STOP") {
    button = 0;
    buttonChanged = true;
  } else if (msg == "START") {
    button = 1;
    buttonChanged = true;
  } else if (msg == "R MQTT") {
    Serial.println("Reconnecting MQTT...");
    Point mqttDown("robot_data");
    mqttDown.addTag("device", DEVICE);
    mqttDown.addTag("SSID", WiFi.SSID());
    mqttDown.addField("mqtt_state", 0);
    influxClient.writePoint(mqttDown);
    client.disconnect();
    delay(2000);
    reconnect();
```

```
    sendConnectivityStatus();
  } else if (msg == "R WIFI") {
    Serial.println("Reconnecting WiFi...");
    Point wifiDown("robot_data");
    wifiDown.addTag("device", DEVICE);
    wifiDown.addTag("SSID", WiFi.SSID());
    wifiDown.addField("wifi_state", 0);
    influxClient.writePoint(wifiDown);
    WiFi.disconnect();
    delay(2000);
    setup_wifi();
    sendConnectivityStatus();
  }
}

// Leest de batterijspanning via een spanningsdeler
float readBatteryVoltage() {
  int adcValue = analogRead(batteryPin);
  float vOut = adcValue * ADC_VREF / ADC_MAX;
  float voltage = vOut / (R2 / (R1 + R2));
  return voltage;
}

// Initialisatie van communicatie en sensoren
void setup() {
  Serial.begin(115200);
  randomSeed(analogRead(0));

  setup_wifi();

  client.setServer(mqtt_server, 2222);
  client.setCallback(callback);

  sensor.addTag("device", DEVICE);
  sensor.addTag("SSID", WiFi.SSID());

  Serial.println("Simulation ready.");
}

// Main loop: behandelt herverbinding, statusupdates, batterijbewaking en logging
void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  unsigned long now = millis();

  if (now - lastMsg > interval || buttonChanged) {
    lastMsg = now;

    float voltage = readBatteryVoltage();

    // Lage batterijspanning loggen en shutdown status beheren
    if (voltage < 1.0) {
      if (!robotShutdown) {
        client.publish("robot/warning", "Battery < 10% - Shutting down robot");
        robotShutdown = true;
```

```
      }
    } else if (robotShutdown && voltage >= 1.0) {
      client.publish("robot/status", "Battery > 10% - Resuming robot operations");
      robotShutdown = false;
    }

    if (!robotShutdown) {
      int moveIndex = random(0, 4);
      float distance = random(0, 5000) / 100.0;

      int wifiState = (WiFi.status() == WL_CONNECTED) ? 1 : 0;
      int mqttState = client.connected() ? 1 : 0;

      char statusMsg[64];
      snprintf(statusMsg, sizeof(statusMsg), "Bat:%.2fV Dist:%.2fcm", voltage, distance);
      client.publish("robot/status", statusMsg);

      if (voltage < 2.8) client.publish("robot/warning", "Battery < 30%");

      // Gegevens toevoegen aan meetpunt voor logging
      sensor.clearFields();
      sensor.addField("voltage", voltage);
      sensor.addField("distance", distance);
      sensor.addField("button", button);
      sensor.addField("wifi_state", wifiState);
      sensor.addField("mqtt_state", mqttState);
      sensor.addField("movement_code", moveIndex);

      Serial.println(sensor.toLineProtocol());
      if (!influxClient.writePoint(sensor)) {
        Serial.print("InfluxDB write failed: ");
        Serial.println(influxClient.getLastErrorMessage());
      }
    }

    buttonChanged = false;
  }
}
```

Functieoverzicht en uitleg

setup_wifi()

Verbindt het ESP32-bord met een opgegeven WiFi-netwerk.
De functie probeert verbinding te maken met het SSID en wachtwoord dat werd opgegeven.
Tijdens de verbinding wordt een laadanimatie getoond via de seriële monitor (.).
Wanneer succesvol verbonden, drukt hij een bevestiging af.

WiFi & MQTT Configuratie:

- WiFi SSID: embed
- WiFi Password: weareincontrol
- MQTT Broker: quecos.local
- MQTT Username: linepilot
- MQTT Password: stayinline
- MQTT Port: 2222

reconnect()

Controleert of de MQTT-client verbonden is.
Indien niet, probeert de functie verbinding te maken met de MQTT-broker (client.connect).
Na succesvolle verbinding wordt het MQTT-kanaal "robot/control" geabonneerd.
Bij mislukking toont hij een foutcode en probeert het opnieuw na 1 seconde.

sendConnectivityStatus()

Logt de status van WiFi- en MQTT-verbinding naar InfluxDB.

- "wifi_state" = 1 als verbonden met WiFi
- "mqtt_state" = 1 als verbonden met MQTT
  De data wordt verzonden met behulp van het InfluxDB Point-object genaamd "robot_data".

callback(char* topic, byte* payload, unsigned int length)
Wordt automatisch aangeroepen wanneer een bericht binnenkomt op het MQTT-topic "robot/control".
Behandelt 4 mogelijke commando's:

- "STOP": zet de interne statusknop op 0
- "START": zet de statusknop op 1
- "R MQTT": forceert een herverbinding met de MQTT-server
- "R WIFI": forceert een herverbinding met WiFi
  Elke actie wordt gelogd in InfluxDB en zichtbaar in de seriële monitor.

readBatteryVoltage()

Leest de batterijspanning via een spanningsdeler aangesloten op een analoge pin.
De gemeten ADC-waarde wordt omgerekend naar een spanning in volt.
Deze functie compenseert voor de weerstandwaarden van de spanningsdeler.

setup()

Wordt één keer uitgevoerd bij opstarten.
Taken:

- Seriële verbinding starten op 115200 baud
- WiFi verbinden via setup_wifi()
- MQTT instellen (setServer, setCallback)
- Standaardtags instellen voor de InfluxDB sensor point
- Meldt via seriële poort dat simulatie klaar is

loop()

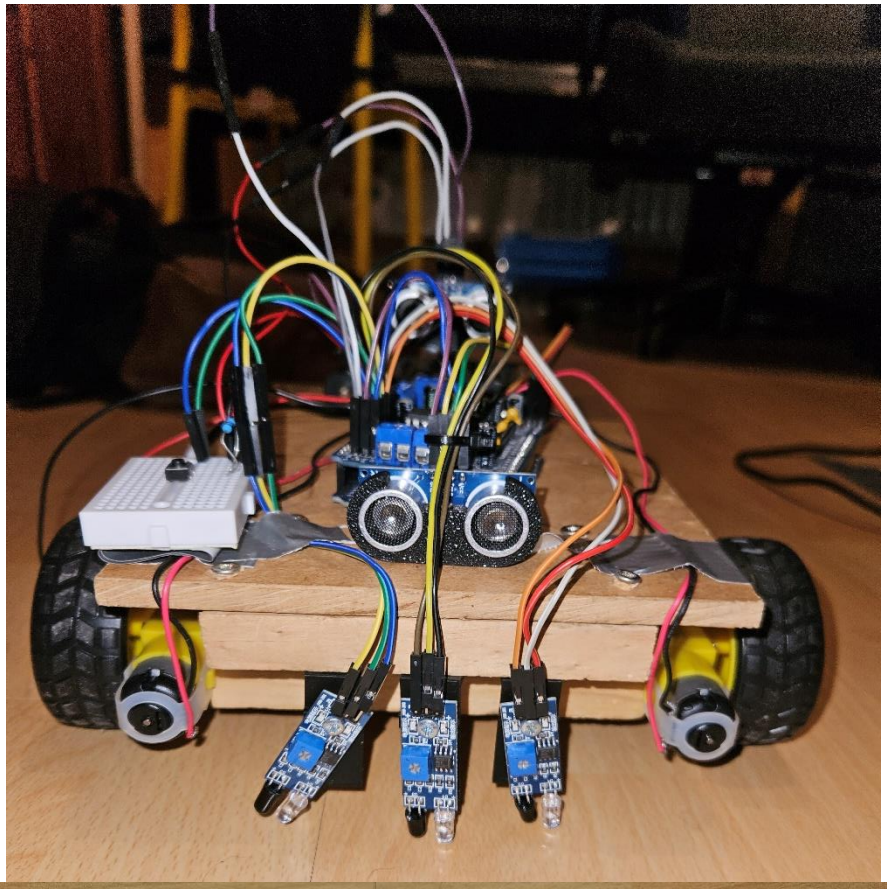
De hoofdloop van het programma, die voortdurend draait.
Taken:

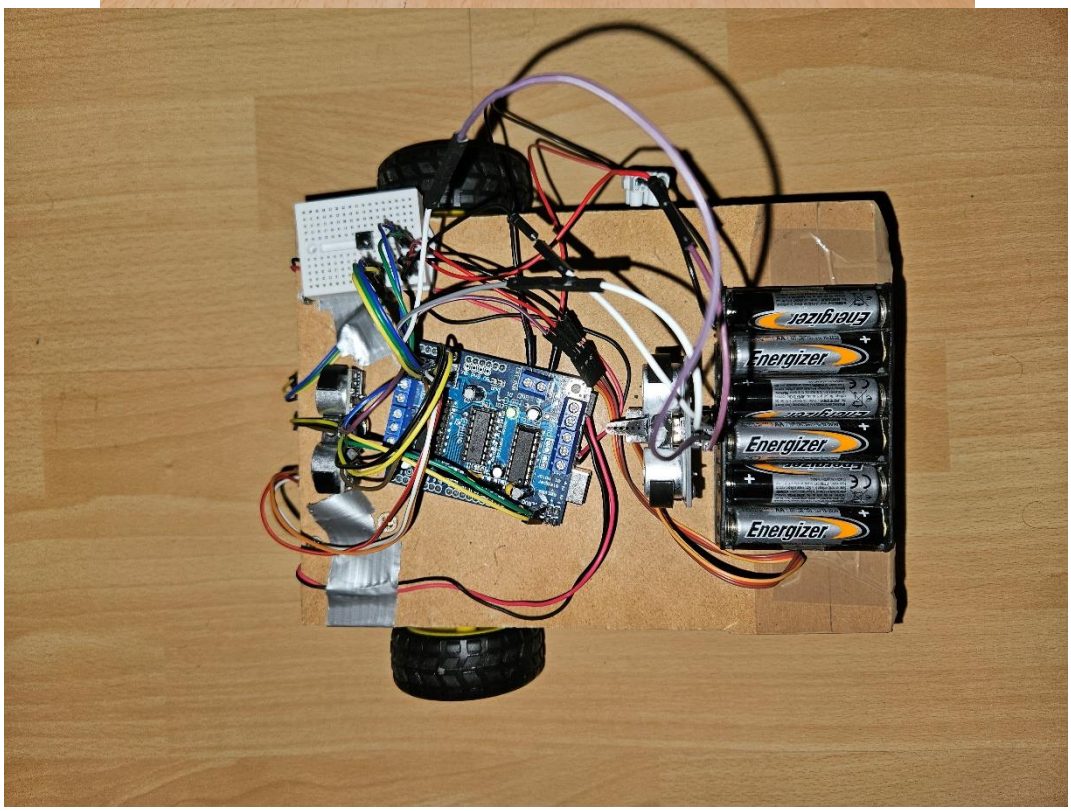
- Verbindt opnieuw met MQTT indien nodig (reconnect())
- Voert client.loop() uit om MQTT-berichten te verwerken
- Elke seconde of bij knopverandering worden:
    - Batterijspanning en willekeurige afstand gegenereerd
    - Verbindingsstatussen (WiFi en MQTT) geëvalueerd
    - Beweging gelogd als willekeurig gekozen actie (forward, turnLeft, turnRight, stopped)
    - InfluxDB sensorwaarden gelogd:
        - "voltage" van de batterij
        - "distance" (gesimuleerd)
        - "button" (status knop)
        - "wifi_state" en "mqtt_state"
        - "movement_code" (index uit movements[])
    - Indien batterij < 2.8V → waarschuwing via MQTT
    - Indien batterij < 1.0V → robot schakelt zichzelf uit (shutdown mode)

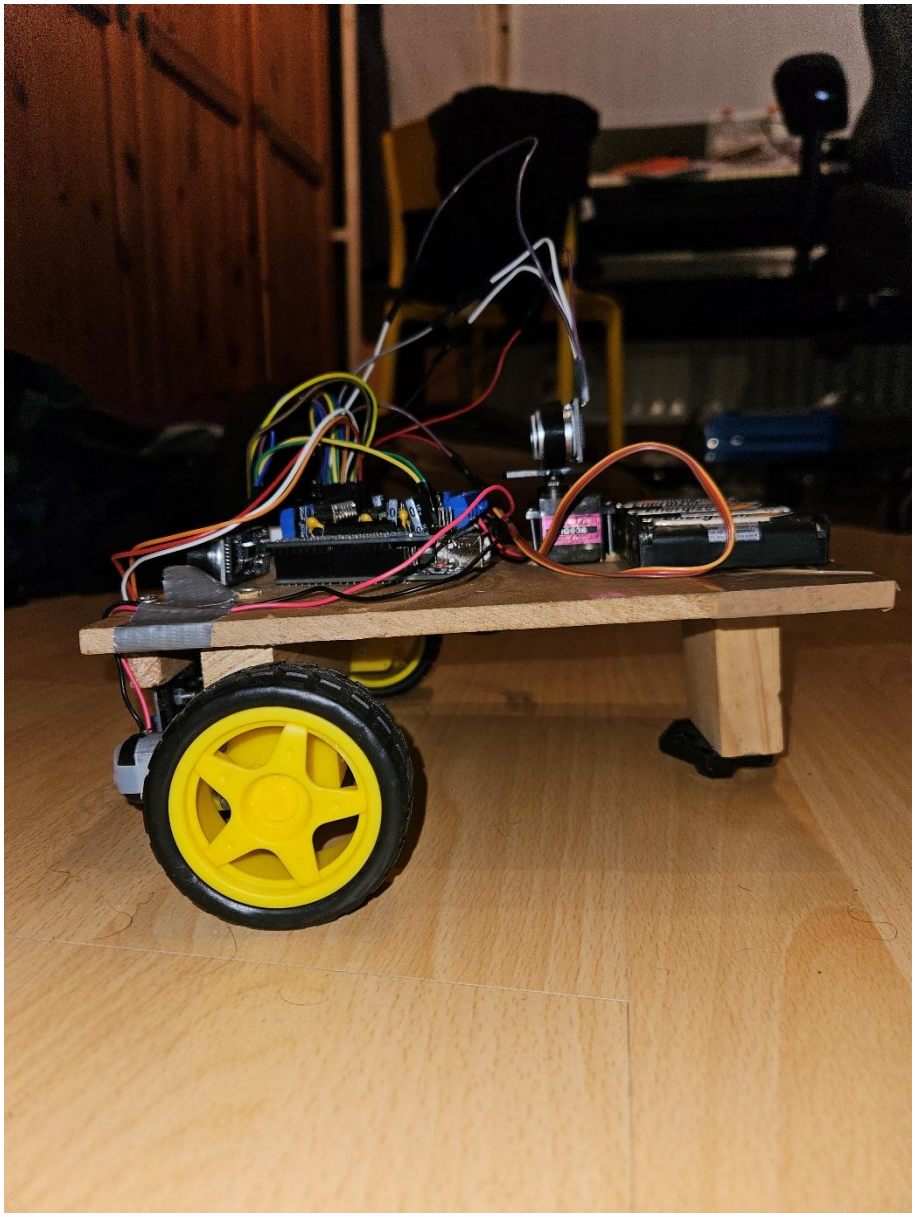
Pin Assignments:

Component	Pin Number	Description
Motor Left Fwd	D5	PWM forward voor left motor
Motor Left Bwd	D18	PWM backward voor left motor
Motor Right Fwd	D19	PWM forward voor right motor
Motor Right Bwd	D21	PWM backward voor right motor
IR Sensor Left	D34	IR line sensor (left)
IR Sensor Center	D32	IR line sensor (center)
IR Sensor Right	D35	IR line sensor (right)
Battery Voltage	D36 (ADC)	Leest battery level (analog)
LCD I2C Address	0x27	16x2 display over I2C

Foto's LijnRobot







MQTT Dashboard + Button Control

- Voor de Broker op te zetten volg je deze tutorial: https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/.
- Om je influxDB op te zetten volg je deze tutorial: https://randomnerdtutorials.com/esp32-influxdb/
- Om je data weer te kunnen geven in Grafana volg je deze tutorial: https://www.youtube.com/watch?v=Jszd7zrl-_U

```
quecos@quecos:~ $ sudo mosquitto_sub -d -t '#' -u linepilot -P stayinline -p 2222
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) received PUBLISH (d0, q0, r0, m0, 'robot/status', ... (17 bytes))
Obstacle Detected
Client (null) received PUBLISH (d0, q0, r0, m0, 'robot/status', ... (21 bytes))
Bat:8.22V Dist:0.00cm
Client (null) received PUBLISH (d0, q0, r0, m0, 'robot/status', ... (17 bytes))
Obstacle Detected
Client (null) received PUBLISH (d0, q0, r0, m0, 'robot/status', ... (21 bytes))
Bat:8.27V Dist:0.00cm
Client (null) received PUBLISH (d0, q0, r0, m0, 'robot/status', ... (17 bytes))
Obstacle Detected
```

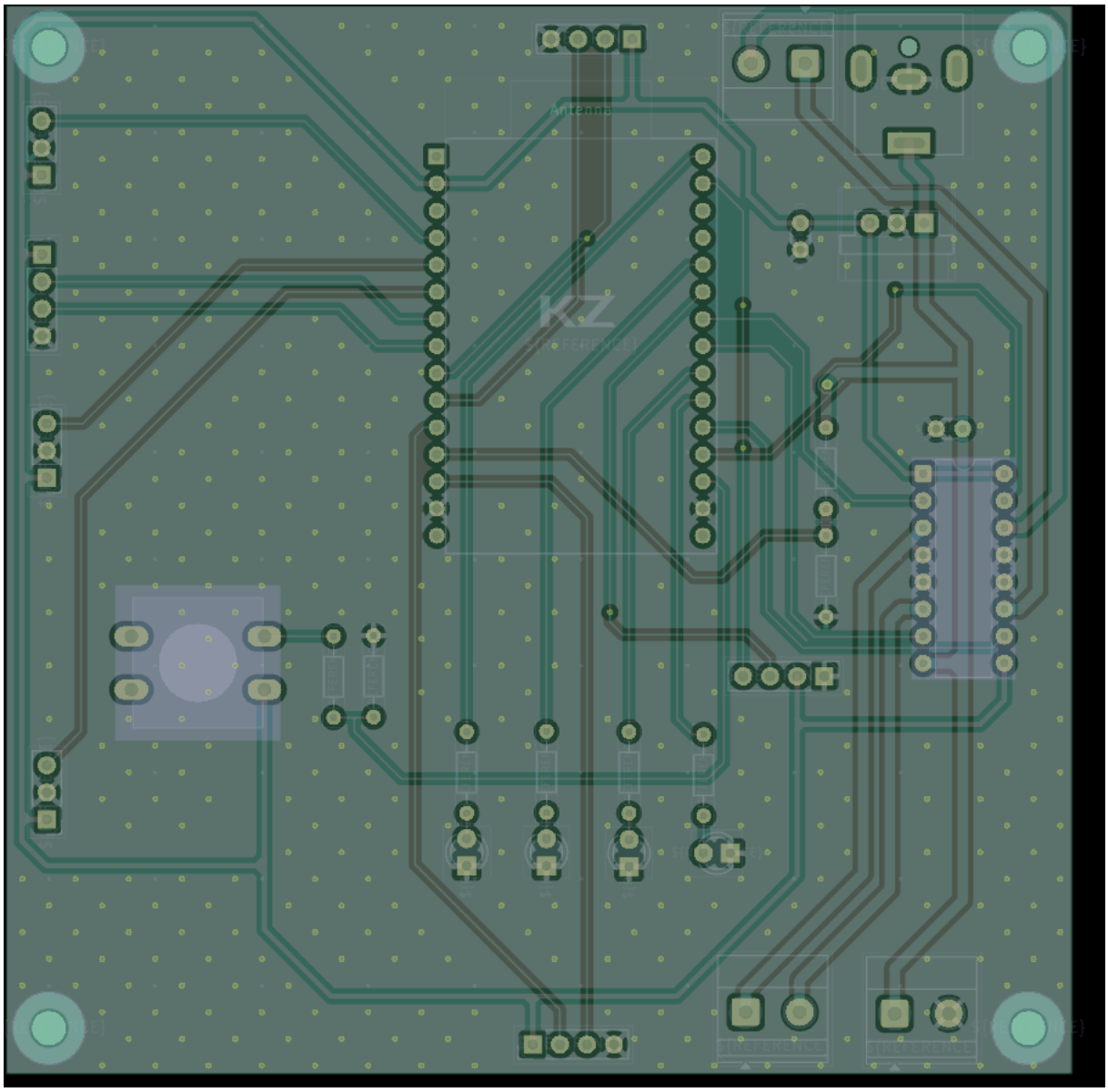
Bij het sturen van ‘START’ of ‘STOP’ over MQTT zal de knop van toestand veranderen waardoor de auto zal stoppen of starten. Dit wordt ook gelogged in influxDB en getoond op grafana.

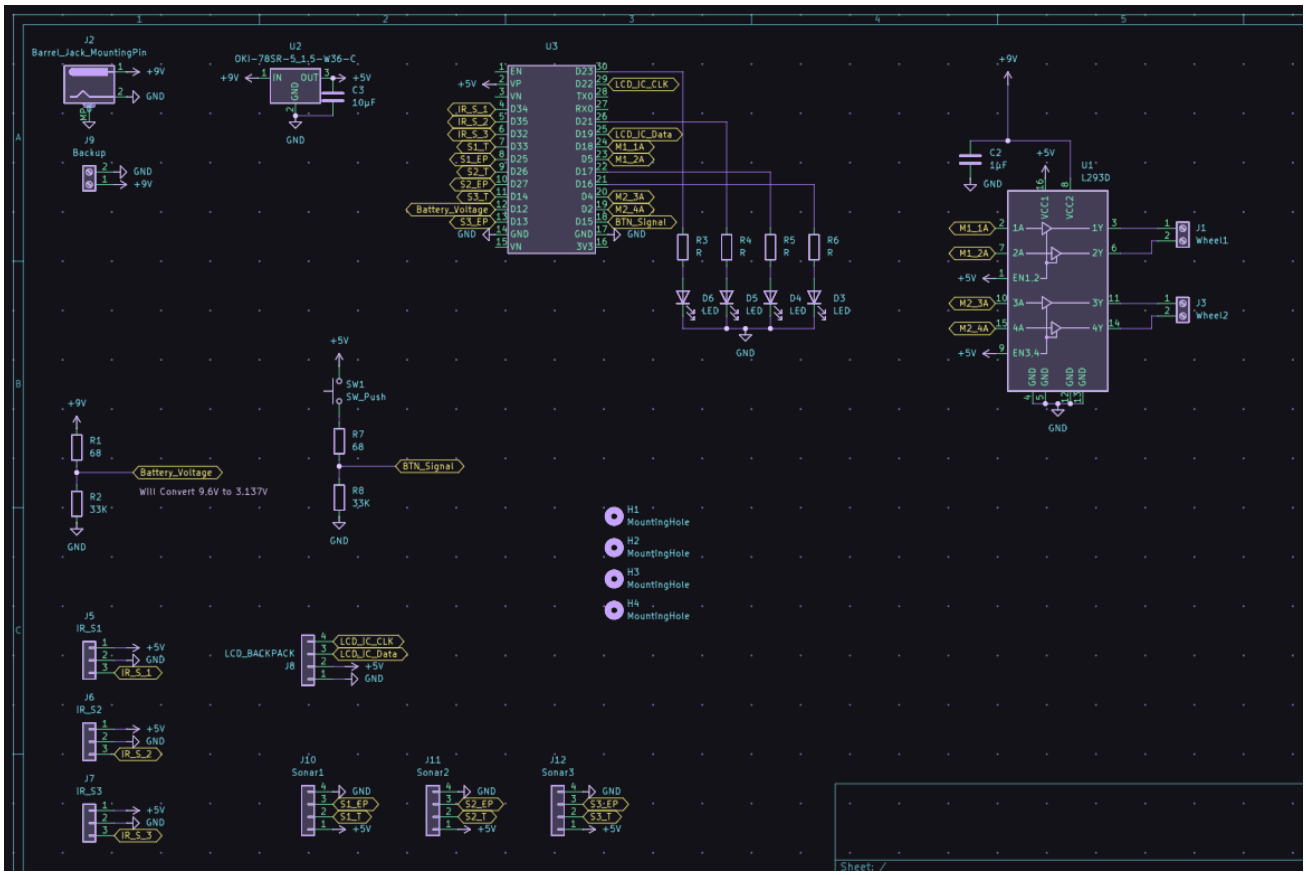
```
quecos@quecos:~ $ sudo mosquitto_pub -d -t 'robot/control' -u linepilot -P stayi
nline -p 2222 -m 'START'
```

```
quecos@quecos:~ $ sudo mosquitto_pub -d -t 'robot/control' -u linepilot -P stayi
nline -p 2222 -m 'STOP'
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'robot/control', ... (4 bytes))
Client (null) sending DISCONNECT
```

```
20:34:16.631 -> MQTT message: STOP
20:34:17.609 -> Writing: robot_data,device=ESP32Robot,SSID=bbox4-71c2 voltage=1.14,button=0i,wifi="connected",mqtt="connected",distance=0.00
20:34:18.721 -> Writing: robot_data,device=ESP32Robot,SSID=bbox4-71c2 voltage=1.14,button=0i,wifi="connected",mqtt="connected",distance=0.00
20:34:19.834 -> Writing: robot_data,device=ESP32Robot,SSID=bbox4-71c2 voltage=1.09,button=0i,wifi="connected",mqtt="connected",distance=0.00
20:34:21.131 -> Writing: robot_data,device=ESP32Robot,SSID=bbox4-71c2 voltage=1.04,button=0i,wifi="connected",mqtt="connected",distance=0.00
```

Kicad pcb





Stap 1: Voeding

Linksboven:

- **J2**: Dit is een aansluiting voor een barrel jack (meestal van een 9V adapter).
- **J9**: Alternatieve voeding (Backup 9V bron).
- **U2**: Een spanningsregelaar (OKI-78SR) die 9V omzet naar **5V**.
- **C5** (condensator): Helpt met spanningsstabilisatie.

Stap 2: ESP32

Middenboven:

- **U3**: De esp.
- Aansluitingen als `IR_S1`, `S1_EP`, `LCD_I2C_CLK`, enz. geven aan wat er verbonden is met elke pin.
- Voeding:
- **+5V naar VIN**
- **GND verbinding**
- Belangrijke signalen:
- `Battery_Voltage`, `BTN_Signal`, motor inputs (`M1_1A`, `M2_1A`, etc.), en I2C signalen (`LCD_I2C_CLK`, `LCD_I2C_Data`).

Stap 3: LEDs en Weergave

- **D3 t/m D6**: LEDs, elk in serie met een weerstand (R3 t/m R6).
- Verbonden met GPIO-pinnen van de microcontroller.
- Gaan waarschijnlijk aan of uit afhankelijk van software.

Stap 4: Drukknop

Middenonder:

- **SW1**: Een drukknop.
- Geconnecteerd via pull-down weerstand (R8) en serieweerstand (R7).
- Signaal wordt gelezen als `BTN_Signal`.

Stap 5: Batterijmonitor

- **R1 en R2**: Spanningsdeler om batterijspanning (9.6V) terug te brengen naar een veilige waarde (~3.3V) voor de microcontroller.
- Signaal wordt `Battery_Voltage` genoemd.

Stap 6: Motor Driver (L293D)

Rechtsboven:

- **U1**: Een L293D H-brug motordriver.
- Voedt 2 motoren (uitgangen: J1 en J3).
- Inputs zijn `M1_1A`, `M1_2A`, enz., vanuit de microcontroller.
- Voeding:
- **+9V naar Vcc2 (motorspanning)**
- **+5V naar Vcc1 (logica)**
- **GND is gedeeld**

Stap 7: Sensoren

Linksonder:

- **IR sensoren**:
- J5, J6, J7: Aansluitingen voor IR-sensoren (elke met +5V, GND, en signaal).
- **Sonar sensoren**:
- J10, J11, J12: Ultrasone afstandssensoren (zelfde structuur: GND, +5V, signaal).

Stap 8: I2C LCD

- **J8**: Aansluiting voor een LCD (via een I2C Backpack).
- Signalen: `LCD_I2C_CLK`, `LCD_I2C_Data`
- GND en +5V ook aanwezig.

Stap 9: Montagegaten

- **H1 t/m H4**: Mechanische gaten voor montage.

Samenvatting Signaalstromen:

- **Voeding**: 9V → 5V via U2 → naar microcontroller, sensoren, en LCD.
- **Sensoren** → microcontroller.
- **Knop** → microcontroller.
- **Microcontroller** → motorcontroller (L293D) → motoren.
- **Microcontroller** → LEDs & LCD voor output.

Bronnen

- https://www.instructables.com/Line-Follower-Robot-With-Arduino-Really-Fast-and-R/
- https://www.youtube.com/watch?v=QEEIPAAoZdA
- https://projecthub.arduino.cc/anova9347/line-follower-robot-with-pid-controller-01813f
- https://www.influxdata.com/get-influxdb-products/
- https://github.com/dhanuzch/L293D-Customized-Motor-Driver
- https://www.youtube.com/watch?v=nzZRsMbKrvo
- https://cults3d.com/en/3d-model/gadget/line-following-robot
- https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/
- https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/
- https://docs.influxdata.com/influxdb/v2/get-started/
- https://grafana.com/docs/
- https://randomnerdtutorials.com/esp32-influxdb/

CONTACT

Naam | Functie
xxx.xxx@thomasmore.be
Tel. + 32 xx xx xx xx

VOLG ONS

www.thomasmore.be
fb.com/ThomasMoreBE
#WeAreMore