THOMAS MORE

Documentatie Iot Project

Brent - Dawoed

2024 - 2025

1. Hardware:

1.1. Componenten

1.1.1. H-brug L298N

Een H-brug is een elektronische schakeling die de richting en snelheid van een DC-motor kan regelen. Het maakt het mogelijk om de motor vooruit en achteruit te laten draaien, evenals de snelheid te variëren via PWM per motor.

1.1.2. TCRT5000 (Infrarood sensor)

De sensor die we gebruiken om te zien of we boven een witte of zwarte lijn rijden. Wit = HIGH.

1.1.3. ESP32-WROOM-32

De ESP32-WROOM-32 om de code uit te voeren.

1.1.4. Raspberry Pi

De Raspberry pi gebruiken we om de mqtt berichten te ontvangen en af te printen in de terminal. Alsook om de data door te sturen via InfluxDB en weer te geven in Grafana.

1.1.5. Ultrasoonsensor (HC-SR04)

Een sensor gebaseerd op geluidsgolven om obstakels te detecteren.

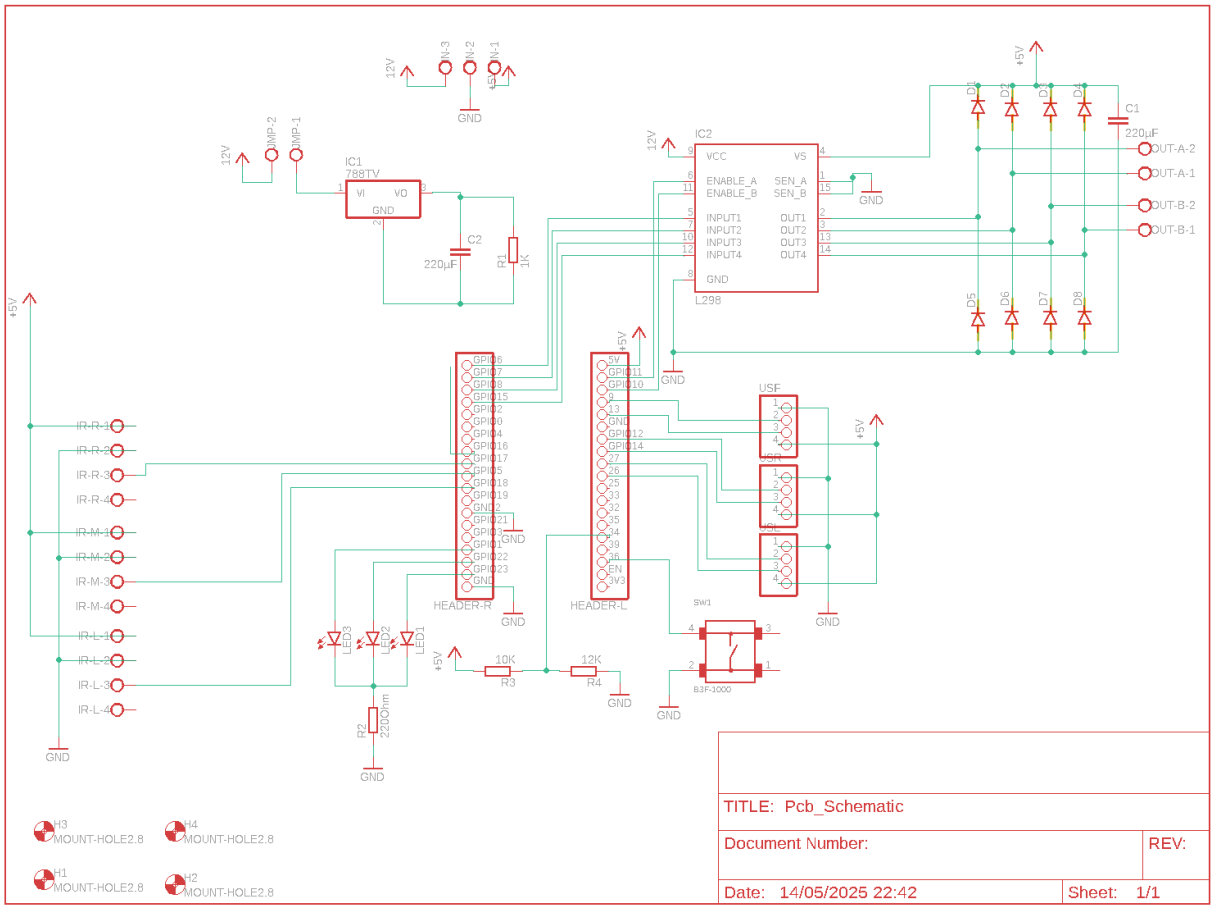
1.1.6. Drukknop

Een drukknop is een schakelaar die een elektrisch circuit tijdelijk sluit of opent.

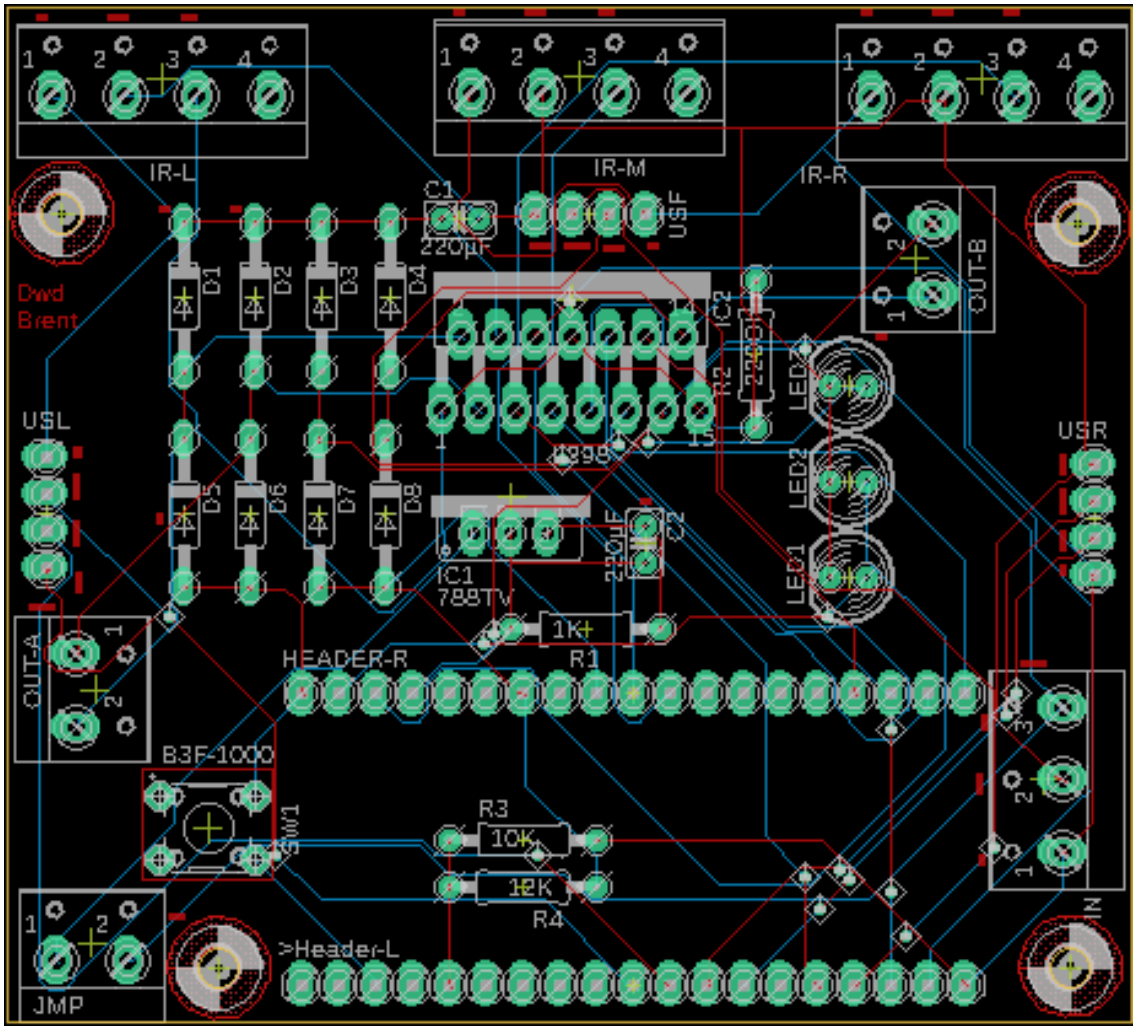
1.1.7. DC-motoren

De motoren die we gebruiken om de robot in beweging te brengen.

1.2. Schema hardware:



1.3. PCB Ontwerp



2. Softwarebeschrijving

2.1. Python

```
# ===== Bibliotheken Importeren =====
# Importeert de Paho MQTT-bibliotheek, een populaire client voor MQTT-communicatie in Python.
# We geven het de kortere naam 'mqtt' voor gebruiksgemak.
import paho.mqtt.client as mqtt
# Importeert de client-bibliotheek voor InfluxDB, een database die geoptimaliseerd is voor tijdgebonden data (zoals
sensormetingen).
from influxdb import InfluxDBClient
# Importeert de 'time' bibliotheek (hier niet direct gebruikt, maar vaak handig in dit soort scripts).
import time
# Importeert het 'datetime' object uit de 'datetime' bibliotheek om met timestamps te kunnen werken.
from datetime import datetime

# ===== Configuratie =====
# Hier worden de instellingen gedefinieerd die het script gebruikt om verbinding te maken.
MQTT_BROKER = "192.168.1.42"  # Het IP-adres van de MQTT-broker (de server die berichten ontvangt en doorstuurt).
MQTT_PORT = 1883              # De standaard netwerkpoort voor MQTT-communicatie.
INFLUX_HOST = "localhost"     # Het adres van de InfluxDB-server. 'localhost' betekent dat deze op dezelfde machine draait als
dit script.
INFLUX_DB = "test_mqtt"       # De naam van de database binnen InfluxDB waar de data wordt opgeslagen.

# ===== InfluxDB Setup =====
# Creëert een InfluxDB-client object. Dit object wordt gebruikt om verbinding te maken met de database en er data naartoe te
schrijven.
influx = InfluxDBClient(host=INFLUX_HOST, database=INFLUX_DB)

# ===== MQTT Callbacks =====
# Een callback-functie wordt automatisch aangeroepen wanneer een specifieke gebeurtenis plaatsvindt.

# Deze functie wordt aangeroepen zodra de client succesvol verbinding heeft gemaakt met de MQTT-broker.
def on_connect(client, userdata, flags, rc, properties):
    # Print een bevestiging en de resultaatcode (rc=0 betekent een succesvolle verbinding).
    print(f"Verbonden met MQTT (code: {rc})")
    # Abonneert de client op een lijst van topics. Vanaf nu zal de 'on_message' functie worden aangeroepen voor berichten op
deze topics.
    # De '0' staat voor Quality of Service level 0, wat betekent "verstuur maximaal één keer".
    client.subscribe([
        ("robot/status", 0),         # Statusberichten van de robot (bv. "Lijnvolgen").
        ("robot/sensors/voor", 0),   # Data van de voorste sensor.
        ("robot/sensors/rechts", 0), # Data van de rechter sensor.
        ("robot/drukknop", 0)        # Status van de fysieke drukknop op de robot.
    ])

# Deze functie wordt aangeroepen telkens wanneer een bericht binnenkomt op een van de geabonneerde topics.
def on_message(client, userdata, msg):
    # Een 'try...except' blok vangt eventuele fouten op tijdens het verwerken van een bericht, zodat het script niet crasht.
    try:
        # Maakt een timestamp in UTC-formaat. InfluxDB werkt het beste met UTC.
        timestamp = datetime.utcnow().isoformat()
        # Haalt het topic uit het binnenkomende bericht (bv. "robot/status").
        topic = msg.topic
        # Haalt de inhoud (payload) uit het bericht en decodeert deze van bytes naar een leesbare string.
        payload = msg.payload.decode()

        # Print het bericht op de console voor directe feedback, met de huidige tijd.
        print(f"[{datetime.now().strftime('%H:%M:%S')}] {topic} : {payload}")

        # Controleert van welk topic het bericht afkomstig is en voert de bijbehorende actie uit.
        if topic == "robot/status":
            # Dit is een slimme manier om de status op te slaan. We schrijven naar drie verschillende 'measurements' (tabellen).
```

```
            statuses = ["Stoppen", "Lijnvolgen", "Obstakel"] # Definieer de mogelijke statussen.

            # Loop door elke mogelijke status.
            for status in statuses:
                # Schrijf een datapunt naar InfluxDB.
                influx.write_points([{
                    "measurement": status.lower(), # De naam van de 'tabel' (bv. 'stoppen').
                    "tags": {"robot": "esp32"},    # Extra metadata om data te filteren.
                    "fields": {
                        # Het veld 'value' krijgt een 1 als de payload overeenkomt met de status, anders een 0.
                        # Hiermee kun je in Grafana makkelijk zien welke status actief was.
                        "value": 1 if payload == status else 0
                    },
                    "time": timestamp # De timestamp van de meting.
                }])

        elif topic == "robot/drukknop": # Als het bericht van de drukknop komt...
            # Converteer de payload ('1' of '0') naar een leesbare status voor de console.
            status = "ingedrukt" if payload == "1" else "losgelaten"
            # Schrijf de knopstatus naar een specifieke measurement in InfluxDB.
            influx.write_points([{
                "measurement": "drukknoprobotje",
                "tags": {"component": "button"},
                "fields": {"value": int(payload)}, # Sla de waarde op als een getal (0 of 1).
                "time": timestamp
            }])
            print(f"Knopstatus: {status}") # Print de leesbare status.

        elif "sensors" in topic: # Als het bericht van een sensor komt...
            # Controleer welke sensor het is.
            if "voor" in topic:
                # Schrijf de waarde van de voorste sensor naar InfluxDB.
                influx.write_points([{
                    "measurement": "voor_sensor",
                    "tags": {"type": "ultrasonic"},
                    # Converteer de payload naar een 'float' (getal met decimalen).
                    "fields": {"value": float(payload)},
                    "time": timestamp
                }])
            elif "rechts" in topic:
                # Schrijf de waarde van de rechter sensor naar InfluxDB.
                influx.write_points([{
                    "measurement": "rechts_sensor",
                    "tags": {"type": "ultrasonic"},
                    "fields": {"value": float(payload)},
                    "time": timestamp
                }])

    except Exception as e:
        # Als er een fout optreedt in het 'try'-blok, print dan een duidelijke foutmelding.
        print(f"Verwerkingsfout: {str(e)}")

# ===== Hoofdprogramma =====
# De 'main' functie bevat de logica om het script op te starten.
def main():
    print("=== Robot Monitoring Systeem ===")
    print("Configuratie:")
    print(f"- MQTT Broker: {MQTT_BROKER}")
    print(f"- InfluxDB Database: {INFLUX_DB}\n")

    # Maakt een MQTT-client object aan. `CallbackAPIVersion.VERSION2` is de moderne, aanbevolen versie.
    mqtt_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    # Koppelt onze 'on_connect' functie aan de 'on_connect' gebeurtenis van de client.
    mqtt_client.on_connect = on_connect
    # Koppelt onze 'on_message' functie aan de 'on_message' gebeurtenis van de client.
    mqtt_client.on_message = on_message
```

```
    try:
        # Probeert verbinding te maken met de MQTT-broker op het opgegeven adres en poort.
        mqtt_client.connect(MQTT_BROKER, MQTT_PORT)
        print("Start monitoring... (Ctrl+C om te stoppen)\n")
        # Start een oneindige lus die op de achtergrond luistert naar MQTT-berichten.
        # Deze functie blokkeert de uitvoering van de code hierna, totdat het script stopt.
        mqtt_client.loop_forever()

    except KeyboardInterrupt:
        # Deze code wordt uitgevoerd als de gebruiker op Ctrl+C drukt om het script te stoppen.
        print("\nAfsluiten...")
    finally:
        # Dit blok wordt altijd uitgevoerd, of het script nu crasht of netjes wordt afgesloten.
        # Het zorgt ervoor dat verbindingen netjes worden gesloten.
        mqtt_client.disconnect()
        influx.close()

# Dit is het startpunt van het script wanneer het wordt uitgevoerd vanuit de command-line.
if __name__ == "__main__":
    # Voordat we starten, zorgen we ervoor dat de database in InfluxDB bestaat.
    # Als deze al bestaat, doet dit commando niets.
    influx.create_database(INFLUX_DB)
    # Roep de hoofdfunctie aan om het monitoringproces te starten.
    main()
```

2.2. Esp32 - Code

```
// ===== Bibliotheken =====
#include <WiFi.h>          // Importeert de bibliotheek om WiFi-functionaliteit te gebruiken op de ESP32.
#include <PubSubClient.h>  // Importeert de bibliotheek voor MQTT (Message Queuing Telemetry Transport) communicatie.

// ===== WiFi en MQTT Instellingen =====
const char* ssid = "embed";                // Definieert de naam (SSID) van het WiFi-netwerk waarmee verbonden moet worden.
const char* password = "weareincontrol";   // Definieert het wachtwoord van het WiFi-netwerk.
const char* mqtt_server = "192.168.1.91";  // Definieert het IP-adres van de MQTT-broker (de server die de berichten verwerkt).

WiFiClient espClient;                  // Creëert een WiFi-client object. Dit is nodig voor de MQTT-bibliotheek om een
netwerkverbinding te maken.
PubSubClient client(espClient);        // Creëert een MQTT-client object en koppelt deze aan de hierboven gemaakte WiFi-
client.

// ===== Sensor Pinnen =====
// Hier worden de fysieke pinnen van de ESP32 gedefinieerd waar de sensoren op aangesloten zijn.
const int trigPinVoor = 4;     // Definieert de pin voor de Trigger van de voorste ultrasone sensor.
const int echoPinVoor = 2;     // Definieert de pin voor de Echo van de voorste ultrasone sensor.
const int SENSOR_MIDDEN = 5;   // Definieert de pin voor de middelste lijnsensor.
const int SENSOR_LINKS = 19;   // Definieert de pin voor de linker lijnsensor.
const int SENSOR_RECHTS = 16;  // Definieert de pin voor de rechter lijnsensor.

// ===== Motor Pinnen =====
// Hier worden de pinnen gedefinieerd voor de L298N motor driver.
const int ENA = 21, IN1 = 3, IN2 = 1;    // Pinnen voor de RECHTER motor: ENA (snelheid), IN1 en IN2 (richting).
const int ENB = 18, IN3 = 22, IN4 = 23;  // Pinnen voor de LINKER motor: ENB (snelheid), IN3 en IN4 (richting).

// ===== Snelheden =====
// Definieert de snelheden voor verschillende manoeuvres. Waardes zijn PWM-waardes (0-255).
const int Snelheid_Forward = 180; // De standaardsnelheid voor vooruit rijden.
const int Snelheid_Back = 180;    // De standaardsnelheid voor achteruit rijden.
const int Snelheid_Draai = 180;   // De standaardsnelheid bij het draaien.

// ===== Obstakel Detectie =====
const float FRONT_OBSTACLE_DISTANCE = 15.0; // De afstand in centimeters waarbij de robot een object als een te
vermijden obstakel beschouwt.
```

```
// ===== Robot Status =====
// Een 'enum' (opsomming) om de verschillende mogelijke statussen van de robot te benoemen. Dit maakt de code leesbaarder.
enum RobotStatus {
  GESTOPT,              // De robot staat stil (initiele staat of na een stop-commando).
  RIJDEND,              // Een algemene rij-status (momenteel minder gebruikt).
  OBSTAKEL_DETECTEERD,  // De robot is bezig met het ontwijken van een obstakel.
  LIJN_VOLGEN,          // De robot is de lijn aan het volgen.
  GESTOPT_DOOR_KNOP     // De robot is gepauzeerd door de fysieke knop.
};
RobotStatus huidigeStatus = GESTOPT; // Een variabele om de huidige status van de robot bij te houden. Begint als GESTOPT.

// ===== Timing Variabelen =====
unsigned long LaatstePublish = 0;   // Houdt de tijd bij van de laatste keer dat sensor data via MQTT is verstuurd.
const long Ultrasoon_Interval = 1000; // Het interval in milliseconden tussen het publiceren van de ultrasone sensorwaarden.

// ===== Motor PWM Kanalen =====
// De ESP32 gebruikt LEDC (LED Control) kanalen voor PWM-signalen. We wijzen elk een kanaal toe.
const int Motor_Left = 0;  // PWM-kanaal 0 voor de linkermotor.
const int Motor_Right = 1; // PWM-kanaal 1 voor de rechtermotor.

// ===== Drukknop Configuratie =====
const int Drukknop = 12;               // Definieert de pin waar de fysieke drukknop op aangesloten is.
bool pauzeActief = false;              // Houdt de 'gedebouncete' staat van de pauzeknop bij (waar = gepauzeerd, onwaar = actief).
bool commandPause = false;             // Houdt bij of de robot gepauzeerd is via een MQTT-commando ("stop").
const unsigned long debounceDelay = 50; // De wachttijd in milliseconden om 'denderen' (meerdere snelle signalen bij één
druk) van de knop te negeren.

// Variabelen voor het 'polling debounce' mechanisme.
bool laatsteKnopStatus = HIGH;         // Slaat de vorige gelezen status van de knop op. HIGH omdat we INPUT_PULLUP
gebruiken (niet ingedrukt = HIGH).
unsigned long lastDebounceTime = 0;    // Tijdstip van de laatste keer dat de knop van status veranderde.
bool knopStatus = HIGH;                // Slaat de stabiele, 'gedebouncete' status van de knop op.


// ===== WiFi en MQTT Setup Functies =====
void setup_wifi() {
  delay(10);                    // Korte vertraging om de hardware te stabiliseren bij opstarten.
  Serial.begin(115200);         // Start de seriële communicatie (voor debug-berichten via USB).
  Serial.println();             // Print een lege lijn voor leesbaarheid.
  Serial.print("Verbinden met WiFi ");
  Serial.println(ssid);         // Print met welk netwerk we proberen te verbinden.

  WiFi.begin(ssid, password);   // Start het WiFi-verbindingsproces met de opgegeven SSID en wachtwoord.

  while (WiFi.status() != WL_CONNECTED) { // Een 'while'-lus die blijft draaien zolang er geen WiFi-verbinding is.
    delay(500);                 // Wacht een halve seconde.
    Serial.print(".");          // Print een puntje om aan te geven dat het proces bezig is.
  }
  Serial.println("");           // Ga naar de volgende regel als de verbinding is gelukt.
  Serial.println("WiFi verbonden");
  Serial.print("IP-adres: ");
  Serial.println(WiFi.localIP()); // Print het verkregen IP-adres op de seriële monitor.
}

// Deze functie wordt automatisch aangeroepen wanneer een MQTT-bericht binnenkomt op een geabonneerd topic.
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Bericht ontvangen [");
  Serial.print(topic);          // Print het topic waar het bericht vandaan kwam.
  Serial.print("]: ");

  String message;               // Maak een lege String om de boodschap in op te slaan.
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];  // Voeg elke byte uit de payload (als karakter) toe aan de message String.
  }
  Serial.println(message);      // Print de volledige ontvangen boodschap.
```

```
  if (String(topic) == "robot/commands") { // Controleert of het bericht afkomstig is van het topic 'robot/commands'.
    if (message == "stop") {              // Als het bericht "stop" is...
      Serial.println("Stopcommando ontvangen!");
      StopMotor();                        // Roep de functie aan om de motoren te stoppen.
      updateStatus(GESTOPT);              // Werk de status van de robot bij naar GESTOPT.
      commandPause = true;                // Zet de vlag voor de MQTT-pauze op 'waar'.
    } else if (message == "start") {      // Als het bericht "start" is...
      Serial.println("Startcommando ontvangen!");
      commandPause = false;               // Zet de vlag voor de MQTT-pauze op 'onwaar'.
      if (!pauzeActief) {                 // Als de robot NIET door de fysieke knop is gepauzeerd...
        updateStatus(LIJN_VOLGEN);        // ...hervat dan met lijnvolgen.
      } else {                            // Anders...
        Serial.println("Robot is gepauzeerd door knop, startcommando genegeerd voor hervatten."); // ...geef een melding dat de knop eerst moet worden gebruikt.
      }
    }
  }
}

// Functie om opnieuw te verbinden met de MQTT-broker als de verbinding wegvalt.
void reconnect() {
  while (!client.connected()) { // Blijf proberen zolang de client niet verbonden is.
    Serial.print("Verbinden met MQTT-broker...");
    if (client.connect("ESP32Client")) { // Probeert te verbinden met een unieke client-ID "ESP32Client".
      Serial.println("verbonden");
      client.subscribe("robot/commands"); // Als de verbinding succesvol is, abonneer dan op het 'robot/commands' topic.
    } else {
      Serial.print("mislukt, rc=");
      Serial.print(client.state());       // Print de foutcode van de MQTT-client.
      Serial.println(" probeer opnieuw over 5 seconden");
      delay(5000);                        // Wacht 5 seconden voordat er opnieuw wordt geprobeerd.
    }
  }
}

// ===== Sensor Functies =====
// Meet de afstand met een ultrasone sensor.
float measureDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);   // Zorg ervoor dat de trigger-pin laag is om te beginnen.
  delayMicroseconds(20);        // Wacht even voor stabiliteit.
  digitalWrite(trigPin, HIGH);  // Zet de trigger-pin 10 microseconden hoog om een geluidspuls te versturen.
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);   // Zet de trigger-pin weer laag.

  // Meet de tijd (in microseconden) dat de echo-pin hoog is. Dit is de tijd die de puls nodig had om heen en terug te reizen.
  // De 30000 is een timeout in microseconden om niet oneindig te wachten als er geen echo terugkomt.
  long duration = pulseIn(echoPin, HIGH, 30000);
  // Berekent de afstand in cm. De duur wordt vermenigvuldigd met de geluidssnelheid (0.034 cm/µs) en gedeeld door 2 (voor de enkele reis).
  // Als de duur 0 is (timeout), geef dan een hoge waarde (999.0) terug om een foute meting aan te duiden.
  return (duration == 0) ? 999.0 : duration * 0.034 / 2;
}

// ===== Motor Functies =====
// Deze functies sturen de motoren aan.
void StopMotor() {
  digitalWrite(IN1, LOW);     // Zet alle richtingspinnen van de motordriver laag.
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  ledcWrite(Motor_Left, 0);   // Zet de snelheid van de linkermotor op 0 via PWM.
  ledcWrite(Motor_Right, 0);  // Zet de snelheid van de rechtermotor op 0 via PWM.
}

void MotorLForward() {
  digitalWrite(IN1, HIGH);    // Stelt de pinnen voor de linkermotor in om vooruit te draaien.
  digitalWrite(IN2, LOW);
```

```
}

void MotorLBack() {
  digitalWrite(IN1, LOW);     // Stelt de pinnen voor de linkermotor in om achteruit te draaien.
  digitalWrite(IN2, HIGH);
}

void MotorRForward() {
  digitalWrite(IN3, HIGH);    // Stelt de pinnen voor de rechtermotor in om vooruit te draaien.
  digitalWrite(IN4, LOW);
}

void MotorRBack() {
  digitalWrite(IN3, LOW);     // Stelt de pinnen voor de rechtermotor in om achteruit te draaien.
  digitalWrite(IN4, HIGH);
}

// ===== Bewegingsfuncties =====
void drivestraight() {
  MotorRForward();                    // Zet de rechtermotor op vooruit.
  MotorLForward();                    // Zet de linkermotor op vooruit.
  ledcWrite(Motor_Right, Snelheid_Forward); // Stel de snelheid voor de rechtermotor in.
  ledcWrite(Motor_Left, Snelheid_Forward);  // Stel de snelheid voor de linkermotor in.
}

void driveleft() {
  MotorRForward();                  // Zet de rechtermotor op vooruit.
  MotorLBack();                     // Zet de linkermotor op achteruit (om ter plekke te draaien).
  ledcWrite(Motor_Right, Snelheid_Draai); // Stel de draaisnelheid in.
  ledcWrite(Motor_Left, Snelheid_Draai);
}

void driveright() {
  MotorLForward();                  // Zet de linkermotor op vooruit.
  MotorRBack();                     // Zet de rechtermotor op achteruit (om ter plekke te draaien).
  ledcWrite(Motor_Left, Snelheid_Draai);  // Stel de draaisnelheid in.
  ledcWrite(Motor_Right, Snelheid_Draai);
}

void driveback() {
  MotorLBack();                     // Zet de linkermotor op achteruit.
  MotorRBack();                     // Zet de rechtermotor op achteruit.
  ledcWrite(Motor_Left, Snelheid_Back);   // Stel de snelheid voor achteruit rijden in.
  ledcWrite(Motor_Right, Snelheid_Back);
}

// ===== Robot Logica =====
// Voert een vooraf gedefinieerde reeks bewegingen uit om een obstakel te ontwijken.
void vermijdObstakelLinks() {
  updateStatus(OBSTAKEL_DETECTEERD); // Werk de status bij naar 'obstakel detecteerd'.
  StopMotor();                      // Stop eerst.
  delay(500);                       // Wacht een halve seconde.
  driveleft();                      // Draai naar links.
  delay(500);
  drivestraight();                  // Rijd een stukje vooruit.
  delay(1100);
  driveright();                     // Draai naar rechts om weer parallel aan de originele route te komen.
  delay(500);
  drivestraight();                  // Rijd voorbij het obstakel.
  delay(500);
  driveright();                     // Draai weer naar rechts om terug te keren naar de lijn.
  delay(250);
}

// Logica voor het volgen van een lijn.
void volgLijn() {
  bool Midden = digitalRead(SENSOR_MIDDEN); // Leest de status van de middensensor.
```

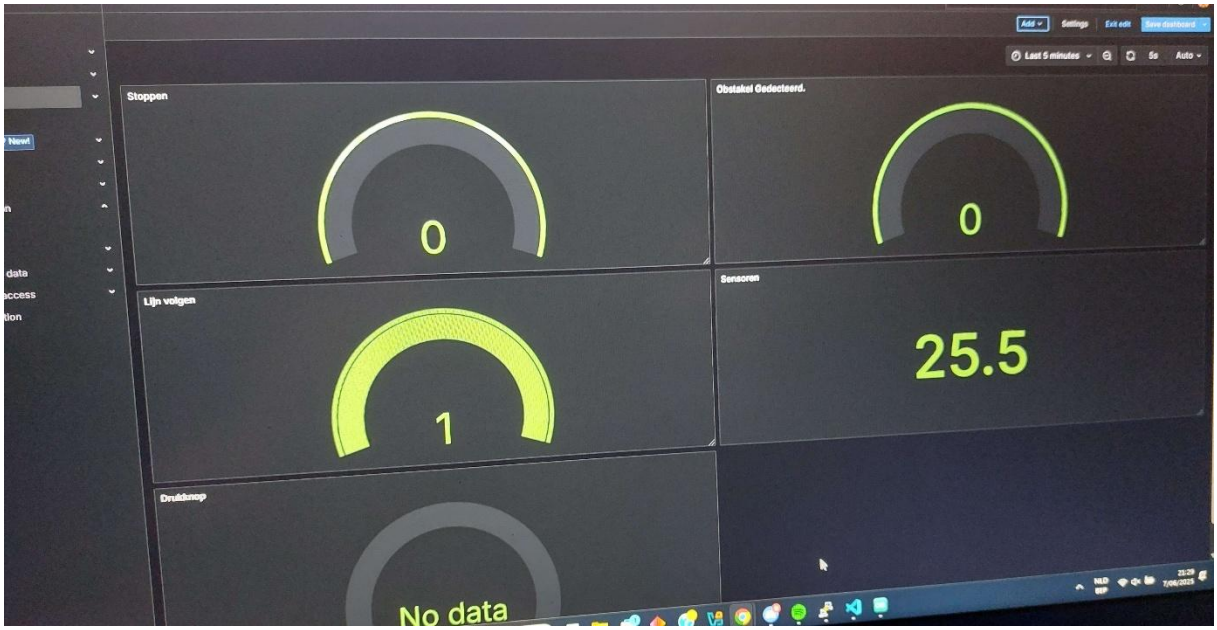
```
  bool Links = digitalRead(SENSOR_LINKS);   // Leest de status van de linkersensor.
  bool Rechts = digitalRead(SENSOR_RECHTS); // Leest de status van de rechtersensor.

  if (Midden && !Links && !Rechts) {        // Als alleen de middensensor de lijn ziet (is HIGH)...
    drivestraight();                        // ...rijd dan rechtdoor.
  } else if (!Midden && !Rechts && Links) { // Als de linkersensor de lijn ziet...
    driveleft();                            // ...stuur dan naar links om te corrigeren.
  } else if (!Midden && !Links && Rechts) { // Als de rechtersensor de lijn ziet...
    driveright();                           // ...stuur dan naar rechts om te corrigeren.
  } else if (Midden && Links && Rechts) {   // Als alle sensoren de lijn zien (bijv. een T-splitsing of eindpunt)...
    StopMotor();                            // ...stop de motor.
    updateStatus(GESTOPT);                  // ...update de status.
    delay(2000);                            // ...wacht 2 seconden.
    drivestraight();                        // ...rijd een klein stukje door (om over de kruising te komen).
    updateStatus(LIJN_VOLGEN);              // ...update de status weer naar lijnvolgen.
    delay(500);
  } else if (!Midden && !Links && !Rechts) { // Als geen enkele sensor de lijn ziet (er vanaf gereden)...
    drivestraight();                        // ...ga rechtdoor in de hoop de lijn weer op te pikken.
  }
}

// Functie om de status van de robot bij te werken en te publiceren via MQTT.
void updateStatus(RobotStatus newStatus) {
  if (huidigeStatus != newStatus) { // Voer de code alleen uit als de status daadwerkelijk verandert.
    huidigeStatus = newStatus;      // Werk de globale statusvariabele bij.

    const char* statusStr = "";     // Maak een pointer voor de status-tekst.
    switch (huidigeStatus) {        // Selecteer de juiste tekstuele status gebaseerd op de 'enum' waarde.
      case GESTOPT: statusStr = "Stoppen"; break;
      case RIJDEND: statusStr = "Rijdend"; break;
      case OBSTAKEL_DETECTEERD: statusStr = "Obstakel"; break;
      case LIJN_VOLGEN: statusStr = "Lijnvolgen"; break;
      case GESTOPT_DOOR_KNOP: statusStr = "Gestopt_knop"; break;
    }

    if (client.connected()) { // Als er een MQTT-verbinding is...
      client.publish("robot/status", statusStr); // ...publiceer de nieuwe status naar het 'robot/status' topic.
    }
    Serial.print("Nieuwe status: "); Serial.println(statusStr); // Print de nieuwe status ook op de seriële monitor.
  }
}

// Publiceert de gemeten afstand van de ultrasone sensor via MQTT op een vast interval.
void printultrasonic(float afstand) {
  unsigned long currentMillis = millis(); // Haal de huidige tijd op.

  // Controleer of er genoeg tijd verstreken is sinds de laatste publicatie.
  if (currentMillis - LaatstePublish >= Ultrasoon_Interval) {
    LaatstePublish = currentMillis; // Reset de timer voor de volgende interval.
    if (client.connected()) {       // Als er een MQTT-verbinding is...
      char voorMsg[8];              // Maak een character array om de float-waarde in op te slaan.
      // Converteer de 'float' (getal met decimalen) afstand naar een 'char array' (tekst).
      dtostrf(afstand, 6, 2, voorMsg);
      client.publish("robot/sensors/voor", voorMsg); // Publiceer de afstand.
    }
  }
}

// ===== Drukknop Functie (met Debounce) =====
// Verwerkt het indrukken van de knop met een debounce-mechanisme om valse detecties te voorkomen.
void handleButtonPress() {
  int reading = digitalRead(Drukknop); // Leest de huidige, ruwe status van de knop (HIGH of LOW).

  // Als de knopstatus is veranderd (mogelijk door denderen of een echte druk)...
  if (reading != laatsteKnopStatus) {
    lastDebounceTime = millis(); // ...reset dan de debounce timer.
  }
```

```
  // Als de status al langer dan 'debounceDelay' stabiel is...
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // ...dan is dit de nieuwe, betrouwbare status.

    // Als de nieuwe betrouwbare status verschilt van de vorige betrouwbare status...
    if (reading != knopStatus) {
      knopStatus = reading; // ...werk de stabiele status bij.

      // Voer de actie alleen uit als de nieuwe stabiele status LOW is (knop is ingedrukt).
      if (knopStatus == LOW) {
        pauzeActief = !pauzeActief; // Wissel de pauzestatus (van waar naar onwaar, of andersom).

        if (pauzeActief) { // Als de robot NU gepauzeerd wordt...
          StopMotor();     // Stop de motoren onmiddellijk.
          updateStatus(GESTOPT_DOOR_KNOP); // Werk de status bij.
          if (client.connected()) client.publish("robot/drukknop", "1"); // Publiceer knopstatus via MQTT.
          Serial.println("Knop ingedrukt - robot gestopt");
        } else { // Als de robot NU hervat wordt...
          // Controleer of de robot ook via MQTT gepauzeerd is.
          if (!commandPause) {
              updateStatus(LIJN_VOLGEN); // Zo niet, hervat lijnvolgen.
          } else {
              updateStatus(GESTOPT); // Zo ja, blijf gestopt (MQTT heeft voorrang).
          }
          if (client.connected()) client.publish("robot/drukknop", "0"); // Publiceer knopstatus.
          Serial.println("Knop losgelaten - robot hervat/status aangepast");
        }
      }
    }
  }
  // Sla de huidige lezing op voor de volgende keer. Zo wordt dit de 'laatsteKnopStatus'.
  laatsteKnopStatus = reading;
}


// ===== Main Setup Functie =====
// Deze functie wordt één keer uitgevoerd bij het opstarten of resetten van de ESP32.
void setup() {
  // Configureer alle sensorpinnen als INPUT. PULLUP betekent dat er een interne weerstand wordt gebruikt,
  // dus de pin is HIGH tenzij hij naar de grond wordt getrokken (door de sensor of knop).
  pinMode(trigPinVoor, OUTPUT);
  pinMode(echoPinVoor, INPUT);
  pinMode(SENSOR_MIDDEN, INPUT_PULLUP);
  pinMode(SENSOR_LINKS, INPUT_PULLUP);
  pinMode(SENSOR_RECHTS, INPUT_PULLUP);
  pinMode(Drukknop, INPUT_PULLUP);

  // Configureer alle motorpinnen als OUTPUT.
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  // Configureer de PWM-kanalen voor de motoren (frequentie 1000Hz, 8-bit resolutie).
  ledcSetup(Motor_Left, 1000, 8);
  // Koppel het PWM-kanaal aan de fysieke pin (ENA).
  ledcAttachPin(ENA, Motor_Left);
  ledcSetup(Motor_Right, 1000, 8);
  // Koppel het PWM-kanaal aan de fysieke pin (ENB).
  ledcAttachPin(ENB, Motor_Right);

  setup_wifi(); // Roep de functie aan om met WiFi te verbinden.
  client.setServer(mqtt_server, 1883); // Configureer de MQTT-client met het serveradres en de standaardpoort (1883).
```

```
  client.setCallback(callback); // Vertelt de MQTT-client welke functie ('callback') hij moet uitvoeren als er een bericht
binnenkomt.
}

// ===== Main Loop Functie =====
// Deze functie wordt continu herhaald zolang de ESP32 stroom heeft.
void loop() {
  if (!client.connected()) { // Controleer bij elke lus of de MQTT-verbinding nog actief is.
    reconnect();             // Zo niet, probeer opnieuw te verbinden.
  }
  client.loop(); // Houd de MQTT-verbinding in stand en verwerkt inkomende berichten (roept 'callback' aan indien nodig).

  handleButtonPress(); // Roep de functie aan die de status van de drukknop controleert.

  float voorAfstand = measureDistance(trigPinVoor, echoPinVoor); // Meet de afstand met de voorste sensor.
  printultrasonic(voorAfstand); // Roep de functie aan die de afstand periodiek via MQTT publiceert.

  // Belangrijke controle: als de robot gepauzeerd is (via de fysieke knop OF een MQTT-commando)...
  if (pauzeActief || commandPause) {
    StopMotor(); // Zorg ervoor dat de motoren gestopt zijn (extra veiligheid).
    return;      // Sla de rest van de lus over. Er wordt geen bewegingslogica uitgevoerd.
  }

  // De bewegingslogica wordt alleen uitgevoerd als de robot niet gepauzeerd is.
  // Controleer eerst op obstakels. De afstand > 0 controle is om foute metingen te negeren.
  if (voorAfstand < FRONT_OBSTACLE_DISTANCE && voorAfstand > 0) {
    vermijdObstakelLinks(); // Als er een obstakel is, roep de ontwijkingsfunctie aan.
  } else {
    // Als de vorige staat niet 'obstakel ontwijken' was, zet de staat dan (terug) op 'lijn volgen'.
    if(huidigeStatus != OBSTAKEL_DETECTEERD) {
        updateStatus(LIJN_VOLGEN);
    }
    volgLijn(); // Als er geen obstakel is, voer de lijnvolg-logica uit.
  }
}
```

3. Webdashboard uitleg:

Om naar Grafana te gaan doe je je IP van je Raspberry pi en dan :3000 ernaast. Vul dan de gerbuikersnaam admin in en het paswoord is paswoord.
Dan zie je moet je verbinden met influxdb en de database van uw keuze. Maak dan een dashboard aan en pas het aan zodat je je gegevens binnenkrijgt en kunt laten tonen.



De status wordt verdeeld in 3. Stoppen, lijnvolgen en obstakel. Dan hebben we nog ultrasonic die de afstanden van de ultrasoon doorstuured. Dan hebben we ook nog de drukknop. Al die 5 staan in een database die test_mqtt noemt met die 5 measurements. Grafana connecteerd met de database via influx en haalt die data op.

Dit toont wanneer hij aan het rijden is, Hij heeft dus geen obstakel gezien daarom een 0. Hij is niet gestop dus daarom ook een 0 en de drukknop is niet ingedrukt dus even geen data. De ultrasoon ziet een afstand van 25.5 cm dus geeft die dat weer.

4. Handleiding voor gebruik

Stap 1: Zorg dat alles goed verbonden is en je batterijen opgeladen zijn.
Stap 2: upload je code naar je ESP32. Pas wifi en IP-gegevens aan.
Stap 3: Connecteer met je Raspberry pi en pas je python code aan met het nieuwe IP-adres.
Stap 4: Open Grafana, verbind met influxdb en zoek naar je database.
Stap 5: Pas Grafana zo aan zodat het eruit ziet zoals je wil.
Stap 6: Run je python code, schakel de schakelaar om en laat het karretje rijden.
Stap 7: Open Grafana en kijk of je gegevens binnenkrijgt.

5. Fouten

Op de pcb hebben we pinnen 6, 7, 8, 9 en 10 gebruikt om te verbinden met de H-brug.
Blijkbaar zijn deze pinnen ook de interne flash pinnen, wat er voor zorgt dat de esp32 in een bootloop terecht komt zodra de motor begint te draaien.
In het vervolg beter opletten welke pinnen er gebruikt worden, om dit te vermijden.
De afstand tussen de twee headers te klein gemaakt, waardoor de pinnen gebogen moesten worden om erin te passen.

6. Github-Link

https://github.com/SkeletonBoszz/Iot-Eindproject

CONTACT

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

VOLG ONS

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

THOMAS MORE