THOMAS MORE

HAICO LUYCKX & JONATHAN DE GEYTER

Line following robot with esp32

2024 - 2025

Inhoud

1. Gebruikte hardware, uitleg en motivatie

Voor het bouwen van onze lijnvolgende robot hebben we zorgvuldig een combinatie van componenten gekozen die zowel betaalbaar als functioneel zijn. Hieronder volgt een overzicht van de gebruikte hardware, inclusief een uitleg van hun werking, rol binnen het systeem en waarom ze gekozen werden.

- 1x RC car chassis kit
  Link

  In deze kit zitten 2 panelen voor de auto, bevestigingsvijzen, 4 motoren, 4 wielen en een batterij pack maar deze gebruiken we niet doordat we onze eigen batterij hebben.

- 1x ESP-WROOM-32 Development Board
  Link

  Deze krachtige microcontroller beschikt over ingebouwde WiFi- en Bluetooth-functionaliteiten en is perfect geschikt voor real-time robotbesturing. Hij verwerkt gegevens van sensoren, stuurt motoren aan en verzorgt de communicatie met het dashboard via MQTT. We hebben voor de ESP32 gekozen omdat hij krachtiger is dan een klassieke Arduino en vlot integreert met moderne IoT-platformen, zoals Node-RED op de Raspberry Pi, maar dit was ook een van de requirements van het eindwerk.

- 1x Raspberry pi 5

  Voor de dataverwerking en dashboardfunctionaliteit gebruiken we een Raspberry Pi 5. Hierop draaien zowel de MQTT-broker (Mosquitto) als de visuele interface via Node-RED. Zo kunnen we alles van op afstand monitoren en besturen.

- 2x IR sensors
  Link

  De IR-sensoren die we gebruiken zijn reflectieve infraroodsensoren, specifiek bedoeld voor lijnvolgende toepassingen. Ze detecteren of het oppervlak onder de robot zwart of wit is. Met twee sensoren (links en rechts) bepalen we de positie ten opzichte van een zwarte lijn op een witte ondergrond. De sensoren geven een digitale waarde door aan de ESP32, waarmee we de koers van de robot automatisch kunnen corrigeren. Zo lezen we een 0 in als het wit ziet en een 1 bij zwart. Deze sensoren zijn gekozen vanwege hun lage kostprijs, eenvoud in gebruik en betrouwbaarheid bij juiste afstelling, maar ook deels aangezien onze eerste sensor niet afstelbaar was.

- 1x L298N dual h-bridge
  Link

  Voor het aansturen van de vier DC-motoren gebruiken we een L298N H-brug. Deze module laat toe om de motoren vooruit en achteruit te laten draaien én om de snelheid te regelen via PWM. Hij is geschikt voor spanningen rond 7V–12V en kan twee motoren onafhankelijk aansturen of je kan een paar aan elke kant koppelen. Dit maakt het een populaire keuze voor kleine robotica-projecten zoals de onze.

- 1x DC buck converter
  Link

  De ESP32 vereist een stabiele 5V-voeding. Omdat de batterij een hogere spanning levert, gebruiken we een buck converter om deze spanning veilig en efficiënt te verlagen. Dit zorgt voor een betrouwbare werking van de ESP32 zonder risico op beschadiging door overspanning.

- Servo motor

  De servomotor wordt gebruikt om de ultrasone sensor dynamisch te draaien. Hierdoor kan de robot niet alleen recht vooruit kijken, maar ook naar links en rechts om te beslissen waar het veilig is om te rijden. Door het aansturen van de servo via PWM-signalen kunnen we exact bepalen in welke richting gemeten moet worden. Deze opstelling verhoogt de betrouwbaarheid van de obstakelvermijding aanzienlijk en hierdoor hoeven we maar 1 ultrasoon te gebruiken.

- Ultrasone sensor (HC-SR04)

  Voor obstakeldetectie maken we gebruik van een ultrasone sensor. Deze sensor meet de afstand tot een object door ultrasoon geluid uit te zenden en de echo op te vangen. In combinatie met een servomotor kunnen we in verschillende richtingen "kijken", wat cruciaal is voor het vermijden van obstakels. We hebben deze sensor gekozen omdat hij relatief nauwkeurig is, een groot bereik heeft en zeer goed gedocumenteerd is voor gebruik met microcontrollers.

- 1x 7.2V NiMH-batterij
  Link

  De voeding van het volledige systeem gebeurt via een 7.2V NiMH-batterij. Deze batterij is veiliger en stabieler dan een LiPo en biedt voldoende capaciteit voor zowel de motoren als de elektronica. Dankzij de keuze voor NiMH is het systeem ook eenvoudiger op te laden met standaard laders.

- 1x drukknop

- Jumper wires
  Link

- Spanningsdeler (10kΩ weerstand & 5.1kΩ weerstand)

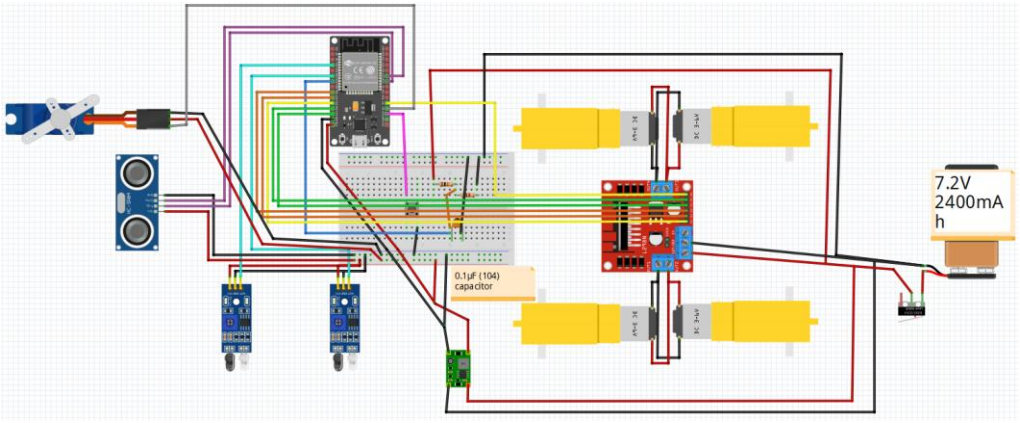
  Om de batterijspanning correct te meten via de analoge ingang van de ESP32, gebruiken we een spanningsdeler. Hiermee brengen we de spanning van de batterij terug binnen het bereik van de 3.3V-logica van de ESP32. De gemeten waarde wordt vervolgens omgerekend naar een percentage dat via MQTT op het dashboard verschijnt, zodat gebruikers steeds een zicht hebben op de resterende batterijduur.

- Fuse (optioneel, maar veilig!)

  We gebruiken een fuse om de batterij te beschermen indien er een slechte connectie plaatsvindt of zich een andere oorzaak voordoet.

2. Schema's en diagrammen

2.1. Aansluitschema



IR-L Sensor → GPIO 34

IR-R Sensor → GPIO 32

Servomotor →GPIO 2

Ultrasonic sensor: Echo = GPIO 19, Trig = GPIO 18

Motor A (L298N): in1 = GPIO 27, in2 = GPIO 26, enA = GPIO 14

Motor B (L298N): in3 = GPIO 13, in4 = GPIO 12, enB = GPIO 4

Startknop → GPIO 15 (INPUT_PULLUP)

Batterijmeting → GPIO 33 (via spanningsdeler: R1=10kΩ, R2=5.1kΩ)

3. Software uitleg

3.1. Lijnvolg- en IR-detectielogica

Onderstaande code zorgt ervoor dat de robot automatisch een zwarte lijn volgt met behulp van twee infraroodsensoren (links en rechts). Afhankelijk van de input van deze sensoren past de robot zijn rijrichting aan of voert hij een herstelprocedure uit.

3.1.1. Voorwaarde: Robot in RUNNING-status

```
if (robotState == RUNNING) {
```

Eerst kijken we naar de state van de auto, dit omdat de auto enkel actief mag zijn en de lijn volgen als hij op RUNNING staat.

3.1.2. Uitlezen van de IR-sensoren

```
int L = digitalRead(IR_L);
int R = digitalRead(IR_R);
Serial.printf("IR L=%d R=%d\n", L, R);
```

De linker (IR_L) en rechter (IR_R) IR-sensoren worden uitgelezen. Deze geven 1 als er zwart (lijn) wordt gedetecteerd en 0 bij wit (ondergrond).

3.1.3. Geen lijn gedetecteerd (L == 0 en R == 0)

```
if (L == 0 && R == 0) {
```

De robot bevindt zich buiten de lijn:

- Start timer wanneer hij begint met rechtdoor rijden:

```
        if (!goingStraight) {
          straightStart = now;
          goingStraight = true;
        }
```

- Herstelactie bij te lang geen lijn:

```
      // Te lang rechtdoor zonder lijn → lijn waarschijnlijk kwijt
      if (now - straightStart >= STRAIGHT_INTERVAL) {
        Stop();
        // Draai 180 graden om terug te keren
        turnLeft();
        delay(TURN_180_DURATION);

        // Rij een stukje terug en controleer opnieuw op lijn
        unsigned long driveStart = millis();
        while (millis() - driveStart < STRAIGHT_INTERVAL) {
          int L = digitalRead(IR_L);
          int R = digitalRead(IR_R);
          forward();

          // Lijn opnieuw gevonden tijdens terugrit → stop en herstel
          if (L == 1 || R == 1) {
            Serial.println("Lijn vroegtijdig gedetecteerd tijdens terugrit");
            Stop();
```

```
            delay(100);
            recoverParcours();  // herstel positie op parcours
            break;
          }

          delay(10);
        }

        // Na terugrit alsnog herstellen
        recoverParcours();
```

Als de robot te lang geen lijn ziet, maakt hij een 180° draai, rijdt een stukje terug, en probeert via recoverParcours() opnieuw de lijn te vinden. Voor meer gedetailleerde info kan je naar 3.5. RecoverParcours

3.1.4. Eén sensor ziet zwart – correctie

Als slechts één sensor zwart detecteert, wordt de robot bijgestuurd om terug op de lijn te komen:

- Linker sensor op zwart, rechter op wit → draai naar links

```
      if (L == 1 && R == 0) {
        turnLeft();
        delay(10);
      }
```

- Rechter sensor op zwart, linker op wit → draai naar rechts

```
      else if (R == 1 && L == 0) {
        turnRight();
        delay(10);
      }
```

3.1.5. Beide sensoren zien zwart (L == 1 en R == 1)

```
else if (L == 1 && R == 1) {
        // Beide sensoren detecteren lijn → horizontale markering
        Stop();
        servoToCenter();  // centreer servo indien nodig
        Serial.println("HORIZ-STREEP: wachten");
        delay(HORIZ_WAIT_MS);  // even wachten op kruispunt of markering

        Serial.println("DOORRIJDEN 600ms");
        forward();  // daarna kort vooruit
        delay(HORIZ_FORWARD_MS);
      }
```

De robot detecteert een horizontale streep:

- Hij stopt

- Centreert de servo indien nodig
- Wacht even
- Rijdt vervolgens een stukje vooruit om verder te gaan

3.1.6. Vertraging tussen metingen

Een korte vertraging zorgt voor stabiele metingen en voorkomt dat de robot te snel reageert op ruis of kleine variaties, dit doen we met een 10ms delay.

3.2. Ultrasonic sensor

We maken gebruik van een ultrasonic sensor die obstakels detecteert en door middel van een servomotor kan de ultrasonic sensor over een grotere zicht beschikken. Zo kunnen we niet alleen zien wat er recht voor zit, maar ook wat er schuin naast de robot is. De robot kiest dan de veiligste kant om te ontwijken.

De functie Ultrasonic_read() stuurt de ping en leest de echo.
Eerst maken we de trigger-pin even laag zodat we een schone pulse hebben.

```
long Ultrasonic_read() {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
```

Daarna gaan we kort hoog (10 µs) om de sensor te laten pingen.

```
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
```

Met pulseIn() meten we hoe lang de echo-pin hoog blijft.

```
  long duration = pulseIn(echoPin, HIGH);
```

Ten slotte converteren we de tijd naar centimeters aan de hand van deze formule:

$$afstand\ (cm) = \frac{duur\ \times\ 0.034}{2}$$

- Duur (seconden): De tijd gemeten door de Arduino.
- 0.034: Dit hebben we nodig om de afstand te kunnen berekenen aangezien dit de snelheid van geluid is.
- De deling door 2 komt omdat het geluid zowel naar het object als terug naar de sensor reist.

```
  return duration * 0.034 / 2;
```

3.3. Servomotor

De functie servoPulse(int angle) zet de servo op een bepaalde hoek.
map(angle, 0, 180, 500, 2400) zet de hoek (0–180°) om naar pulsduur (500–2400 µs)

```
void servoPulse(int angle) {
  int us = map(angle, 0, 180, 500, 2400);
```

Door de pin hoog te zetten voor die tijd en daarna laag, draait de servo naar de gewenste positie.

```
  digitalWrite(servoPin, HIGH);
  delayMicroseconds(us);
  digitalWrite(servoPin, LOW);
  delay(50);
```

Om de sensor te laten sweepen (heen en weer draaien) gebruiken we de functie servoSweepScan()
We beginnen in het midden (80°) en draaien stapsgewijs naar 140° om rechts te meten. De ultrasonic sensor meet de afstand aan de rechterkant.

```
SweepResult servoSweepScan() {
  SweepResult result;

  // Sweep naar rechts
  for (int a = 80; a <= 140; a += 5) {
    servoPulse(a);
    delay(30);
  }
  result.distanceRight = Ultrasonic_read();
  delay(100);
```

Daarna draaien we van 140° terug naar 0° om links te meten.
De ultrasonic sensor meet de afstand aan de linkerkant.

```
  // Sweep naar links
  for (int a = 140; a >= 0; a -= 5) {
    servoPulse(a);
    delay(30);
  }
  result.distanceLeft = Ultrasonic_read();
  Serial.printf("  L = %l cm\n", result.distanceLeft);
  delay(100);
```

Tussen elke stap wordt er gewacht met een kleine delay (30 ms) zodat de servo de tijd heeft om te bewegen.

We bewaren beide afstanden in SweepResult (zie ook bovenaan)

```
  return result;
```

3.4. Obstakelvermijding

In de mainloop() roepen we de ultrasonic functie om te checken of er een object vóór de robot staat.

OBSTACLE_DIST is een drempel (bijv. 15 cm), dit kan aangepast worden bovenaan de code.

```
  long dF = Ultrasonic_read();
  if (dF <= OBSTACLE_DIST) {
    avoidObstacle();
    return;
  }
```

Als iets dichterbij is, voeren we de obstakelvermijding uit.

We maken gebruik van avoidObstacle() om de robot te laten stoppen en de sensor te laten sweepen en beslissen welke kant vrij is om een manoeuvre te doen.

De robot stop en centreert de servomotor met de servoToCenter().

```
void avoidObstacle() {
  Stop();
   servoToCenter();
```

Voert een sweep uit met de servo en bewaart afstanden van links en rechts.

```
SweepResult scan = servoSweepScan();
```

Als beide kanten te dichtbij zijn (objecten in de weg), verhoogt de teller.

```
if (scan.distanceLeft <= OBSTACLE_DIST_LATERAL && scan.distanceRight <=
OBSTACLE_DIST_LATERAL) {
    obstacleBlockCount++;
```

Wanneer de tellerwaarde de limiet overschrijdt in OBSTACLE_BLOCK_LIMIT die je bovenaan kan aanpassen in de code, dan rijdt de robot even achteruit, maakt een U-turn en verwijst naar de functie recoverParcours() om het parcours terug te vinden.

```
if (obstacleBlockCount >= OBSTACLE_BLOCK_LIMIT) {
      // Achteruit
      digitalWrite(in1, HIGH);
      digitalWrite(in2, LOW);  // motor A achteruit
      digitalWrite(in3, LOW);
      digitalWrite(in4, HIGH);  // motor B achteruit
      analogWrite(enA, MOTOR_SPEED);
      analogWrite(enB, MOTOR_SPEED);
      delay(500);  // beetje achteruit
      Stop();

      // Nu draaien vóór recoverParcours
      turnLeft();
      delay(TURN_180_DURATION);
      Stop();
```

```
      // Herstel naar parcours
      recoverParcours();
```

De servomotor wacht een bepaalde tijd in BOTH_WAIT_MS die je ook bovenaan kan aanpassen in de code alvorens terug te scannen of er objecten in de weg zijn.
Indien er een object wegvalt dan wordt obstacleBlockCount gereset.

```
servoToCenter();
    delay(BOTH_WAIT_MS);
    return;
  } else {
    obstacleBlockCount = 0;  // reset als er wel ruimte is
  }
```

De robot kiest altijd de kant met de grootste afstand en voert een manoeuvre uit om soepel om het obstakel heen te gaan. Als er bijvoorbeeld links een object staat, dan start de robot zijn manoeuvre rechts.

```
if (scan.distanceLeft > scan.distanceRight) {
    turnRight();
    delay(500);
    forward();
    delay(800);
    turnLeft();
    delay(500);
    forward();
    delay(800);
    turnLeft();
    delay(600);
    forward();
    delay(200);
  } else {
    turnLeft();
    delay(500);
    forward();
    delay(800);
    turnRight();
    delay(500);
    forward();
    delay(800);
    turnRight();
    delay(600);
    forward();
    delay(200);
  }
```

3.5. RecoverParcours

Omdat de goedkope IR-sensoren erg gevoelig zijn en een bocht niet goed afgerond kan zijn, is het goed mogelijk dat de robot makkelijk van het parcours af kan rijden. Er is hier een deelse oplossing voor maar het is nog niet 100 procent foolproof.

Aan de hand van de sensors L == 0 && R == 0 weten we hoelang de robot rechtdoor rijdt.
goingStraight is een vlag die bijhoudt of de robot op dit moment rechtdoor rijdt.
straightStart slaat de tijd op van wanneer goingStraight aantoont dat de robot rechtdoor rijdt

```
    if (L == 0 && R == 0) {
      if (!goingStraight) {
        straightStart = now;
        goingStraight = true;
      }
```

STRAIGHT_INTERVAL is de limiet (ms) dat de robot rechtdoor mag blijven rijden, dit kan aangepast worden bovenaan de code. Als straightStart groter is dan STRAIGHT_INTERVAL wil dat zeggen dat de robot uit het parcours ligt.
De robot stopt en maakt een 180° draai en maakt de terugrit voorwaarts tot de limiet van STRAIGHT_INTERVAL is bereikt. Zolang het verschil tussen de huidige tijd en de starttijd kleiner is dan STRAIGHT_INTERVAL, blijft men in de lus. Hierdoor zal de robot dichter bij het parcours komen.

```
      if (now - straightStart >= STRAIGHT_INTERVAL) {
        Stop();
        // 180° + terugrijden
        turnLeft();
        delay(TURN_180_DURATION);
        unsigned long driveStart = millis();
        while (millis() - driveStart < STRAIGHT_INTERVAL) {
          int L = digitalRead(IR_L);
          int R = digitalRead(IR_R);
          forward();
```

De robot blijft detecteren of er in de terugrit tenminste 1 van de IR-sensoren zwart (=1) toont en zo de lijn raakt. De recoverParcours() functie wordt dan opgeroepen.

```
          // Als hij iets van de lijn detecteert → stop en herstel
          if (L == 1 || R == 1) {
            Stop();
            delay(100);
            recoverParcours();
            break;
          }
```

Indien de robot geen lijn detecteert tijdens de terugrit, wordt de functie recoverParcours() buiten de while lus eveneens opgeroepen.

In de functie recoverParcours(), zolang 1 sensor geen zwart (1) ziet dan blijft het langzaam doorrijden anders stopt de lus.

```
void recoverParcours() {

  int L, R;

  // Zoek naar detectie van iets
  while (true) {
    L = digitalRead(IR_L);
    R = digitalRead(IR_R);
    if (L == 1 || R == 1) break;
    forward();
    delay(10);
  }
```

De robot stopt bij het detecteren van zwart (1) en wanneer er maar 1 van de 2 sensors dit detecteert, betekent dat de robot netjes op het parcours staat en moet de robot geen turn maken om zijn plaatsing te corrigeren. Return, verlaat de functie en de robot volgt de normale lijnvolg logica terug.

```
 Stop();

  // Bepaal gedrag op basis van patroon
  if ((L == 1 && R == 0) || (L == 0 && R == 1)) {
    return;
  }
```

Als de 2 sensors zwart (1) zien, dan is de robot slecht gepositioneerd om correct het normale parcours te volgen. Met turnLeft() begint de robot te draaien tot er maar 1 van de sensors zwart (1) ziet. Robot stopt en kan de normale lijnvolg logica terug volgen.

```
  if (L == 1 && R == 1) {
    // Beide detecteren zwart → mogelijk midden van lijn of kruising

    // Begin draaien (bijvoorbeeld linksom)
    while (true) {
      turnLeft();
      delay(10);
      L = digitalRead(IR_L);
      R = digitalRead(IR_R);
      if ((L == 1 && R == 0) || (L == 0 && R == 1)) break;
    }

    Stop();
    delay(100);
  }
```

3.6. Kruising

Bij een kruising wanneer de sensors beiden zwart (1) detecteren, wordt dit beschouwt als een rustpauze. De robot stopt en met HORIZ_WAIT_MS kan je bovenaan de code instellen hoelang de robot pauze neemt. Na de pauze rijdt de robot met forward() enkele seconden vooruit. HORIZ_FORWARD_MS is het aantal seconden dat de robot nodig heeft om over de streep te gaan zodat beide sensors voor een kleine tijd niet reageren bij het detecteren van zwart. Dit kan bovenaan de code ook aangepast worden.

```
else if (L == 1 && R == 1) {
        // Horizontale streep
        Stop();
        delay(HORIZ_WAIT_MS);
        forward();
        delay(HORIZ_FORWARD_MS);
      }
```

3.7. Batterijmeting en statuspublicatie

De functie publishBatteryAndStatus() meet de batterijspanning van de robot, schat het batterijniveau in procenten, stuurt deze status via MQTT door, en schakelt automatisch de robot uit bij een te lage batterij. Daarnaast stuurt ze drie status-LED's aan om de batterijstatus visueel weer te geven.

3.7.1. Spanningsmeting via analoge pin

```
  const int NUM_SAMPLES = 10;
  long total = 0;

  for (int i = 0; i < NUM_SAMPLES; i++) {
    total += analogRead(analogPin);
    delay(5);  // kleine vertraging voor stabiliteit
  }
```

- Er worden 10 metingen gedaan aan de analoge pin die verbonden is met een spanningsdeler (batterij).
- Een korte vertraging tussen metingen zorgt voor een stabiel gemiddelde.
- Ruis wordt zo geminimaliseerd.

3.7.2. Omrekenen naar spanning

```
  float avgRaw = total / float(NUM_SAMPLES);
  float vPin = (avgRaw / 4095.0) * 3.3;
  float battV = vPin * ((r1 + r2) / r2);
```

- avgRaw is de gemiddelde ruwe waarde van 0 tot 4095 (12-bit ADC van de ESP32).
- Deze waarde wordt omgerekend naar een spanning aan de pin (vPin), veronderstellend dat de referentiespanning 3.3V is.
- Via de spanningsdelerformule wordt de echte batterijspanning (battV) berekend.

3.7.3. Batterijpercentage bepalen

```
  const float MIN_V = 6.0;
  const float MAX_V = 7.2;
  int pct = constrain(int((battV - MIN_V) / (MAX_V - MIN_V) * 100), 0, 100);
```

- Er wordt aangenomen dat 6.0V een lege batterij is en 7.2V volledig vol.
- Het percentage wordt berekend en begrensd tussen 0% en 100% met constrain().

3.7.4. LED-indicatie voor batterijstatus

```
  digitalWrite(LED_GREEN, pct > 30);
  digitalWrite(LED_YELLOW, pct > 10 && pct <= 30);
  digitalWrite(LED_RED, pct <= 10);
```

- Groene LED: voldoende batterij (> 30%)
- Gele LED: matige batterij (tussen 10% en 30%)
- Rode LED: bijna leeg (≤ 10%)

3.7.5. Automatisch stoppen bij lage batterij

```
  if (pct <= 10 && robotState == RUNNING) {
    Stop();
    robotState = STOPPED;
    stopStartTime = millis();
    currentStatus = "LAGE BATTERIJ";
    Serial.println("Batterij te laag: robot gestopt");
  }
```

- Als de batterij kritiek laag is (≤ 10%) en de robot rijdt nog, wordt deze automatisch gestopt.
- De status wordt aangepast naar "LAGE BATTERIJ".

3.7.6. Verzenden via MQTT

```
  char payload_batt[64];
  snprintf(payload_batt, sizeof(payload_batt),
"{\"voltage\":%.2f,\"pct\":%d}", battV, pct);
  client.publish(topic_battery, payload_batt);
  client.publish(topic_status, currentStatus.c_str());
```

- De batterijspanning en het percentage worden in JSON-formaat verzonden naar de MQTT-broker.
- Ook de status van de robot wordt meegestuurd.

3.8. Distance publicatie

```
void publishDistance(long distance) {
  char payload_dist[64];
  snprintf(payload_dist, sizeof(payload_dist), "{\"distance\": %ld, \"unit\":
\"cm\"}", distance);
  client.publish(topic_distance, payload_dist);
}
```

Hier sturen we de afstand die we meekrijgen van de metingen voor de afstand door naar MQTT in een JSON formaat.

3.9. Button

3.9.1. Robot statussen en button inlezing

Voor de knop hebben we 4 toestanden gemaakt in een enum en zetten hem standaard op IDLE om niets te doen.

```
enum RobotState { IDLE, RUNNING, STOPPED, WAIT_FOR_REPRESS };
RobotState robotState = IDLE;
```

```
bool currentButtonState = digitalRead(BUTTON_PIN);
bool buttonPressed = currentButtonState == LOW && lastButtonState == HIGH &&
now - lastButtonPress > DEBOUNCE_TIME;
lastButtonState = currentButtonState;
```

Dit zorgt ervoor dat:

- Enkel een **rising edge** telt (knop wordt **net** ingedrukt).
- Minimale tijd tussen drukken is 200 ms (debounce).

3.9.2. Gedrag per status

We gebruiken een switch case om makkelijk van status te veranderen.

```
switch (robotState) {
```

3.9.2.1. IDLE

Met een druk start hij met rijden.

```
case IDLE:
      if (buttonPressed) {
        robotState = RUNNING;
        currentStatus = "RIJDEND";
      }
      break;
```

3.9.2.2. RUNNING

Stopt onmiddellijk en onthoud het tijdstip dat hij gestopt is.

```
case RUNNING:
      if (buttonPressed) {
        robotState = STOPPED;
        stopStartTime = now;
        Stop();
        currentStatus = "GESTOPT";
      }
      break;
```

3.9.2.3. STOPPED

Hij wacht de timer af om verder te kunnen gaan, zelfs als je duwt tijdens de timer zal de auto niet bewegen.

```
case STOPPED:
      Stop();
      if (now - stopStartTime >= STOP_DURATION) {
        robotState = WAIT_FOR_REPRESS;
        buttonReleasedAfterStop = false;
        currentStatus = "WACHT OP KNOP";
      }
      break;
```

3.9.2.4. WAIT_FOR_PRESS

Enkel als de timer is afgelopen en erna de knop wordt ingeduwd hervat hij.

```
case WAIT_FOR_REPRESS:
      Stop();
      if (!buttonReleasedAfterStop && currentButtonState == HIGH) {
        buttonReleasedAfterStop = true;
      } else if (buttonReleasedAfterStop && buttonPressed) {
        robotState = RUNNING;
        currentStatus = "RIJDEND";
      }
      break;
```

3.9.3. Dashboard button

De dashboard button volgt dezelfde logica en werkt samen met de echte knop op de auto waardoor de status gesynchroniseerd zal zijn.

4. MQTT-instellingen

WiFi-instellingen:

```
const char* ssid        = "WiFi-2.4-6A9C_EXT";
const char* wifi_pass   = "w2p7k76kpcrfu";
```

MQTT-instellingen:

```
const char* mqtt_server = "192.168.1.39";
const int   mqtt_port   = 1883;
```

4.1. Topics

4.1.1. Batterij

```
const char* mqtt_topic  = "robot/battery";
```

- **Berichtfrequentie:** elke second
- **Payload-structuur (JSON):**

```
{
"voltage": 7.82,
"pct": 67
}
```

De spanning van de batterij wordt gemeten via een spanningsdeler en omgerekend naar een percentage. Deze waarde wordt via MQTT gepubliceerd zodat deze op een dashboard kan worden weergegeven.

4.1.2. Status

```
const char* topic_status  = "robot/status";
```

- **Berichtfrequentie:** elke second
- **Payload-strucuur:** String

Bij deze topic sturen we de momentele status van de auto, dit wil dus zeggen: IDLE, RIJDEND, GESTOPT of WACHT OP KNOP. Zo kunnen we dan vanop afstand zien wat de auto aan het doen is op dat exact moment.

4.1.3. RemoteButton

```
client.subscribe("robot/remoteButton");
```

We hebben ook de remoteButton knop waarmee we via ons dashboard de status van de auto kunnen veranderen.

5. Webdashboard

5.1. Hoe werkt het?

Voor de communicatie van het dashboard gebruiken we MQTT en Node-RED, beide draaien op een raspberry pi 5 dus via hetzelfde IP-adres.

Op het dashboard zelf kan je dan “nodes” gebruiken zoals de MQTT in node die je met het IP van de raspi kan verbinden en zeggen naar welke topics hij moet luisteren en welke user hiervoor te gebruiken.

5.2. Installatie

5.2.1. Node-RED

Om te beginnen updaten we de raspi met de commando’s

```
sudo apt update
sudo apt upgrade -y
```

Daarna installeren we de Node-RED met volgend commando en gaan we door de installatie.

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

We installeren node-red-dashboard om fouten bij het importeren van de .json file te voorkomen.

```
cd ~/.node-red
npm install node-red-dashboard
```

Dan gaan we ervoor zorgen dat Node-RED altijd opstart wanneer we onze raspi opstarten

```
sudo systemctl enable nodered.service
sudo systemctl start nodered.service
```

Tot slot kunnen we naar de webpagina gaan door het IP van de raspi in te geven zoals volgend voorbeeld:

http://192.168.1.123:1880

5.2.2. MQTT

Je kan de installatie volgen van deze website: https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/

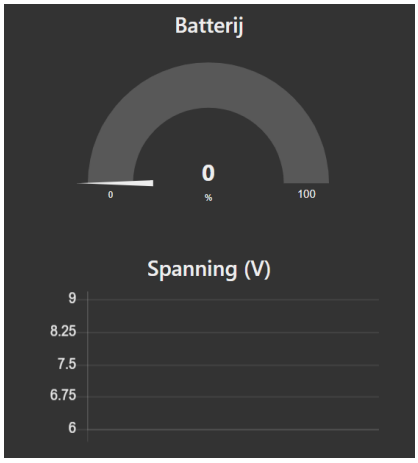
5.3. Welke data wordt weergegeven?

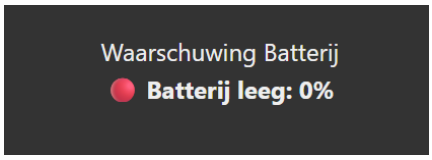
5.3.1. Button

We hebben een button waarmee we de auto op afstand kunnen stoppen en starten zoals met de echte knop op de auto zelf. Dit met een uitsturen commando via MQTT.

5.3.2. Batterij

Voor de batterij geven we zowel het percentage als de voltage als een waarschuwing weer over de status van de batterij op dat moment. Hij kijkt op Node-RED zelf naar de binnenkomende waardes om hier een waarschuwing voor te geven.

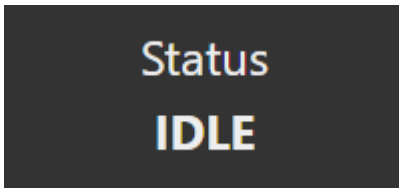




5.3.3. Status auto

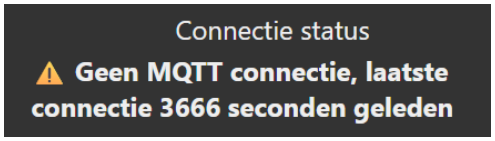
Hier geven we de momentele status van de auto weer er is dus keuze tussen:

- IDLE
- RIJDEND
- GESTOPT
- WACHT OP KNOP



5.3.4. MQTT status

We geven hier weer hoe de connectie met MQTT is, als deze weg valt komt er ook bij te staan wanneer de connectie voor het laatst actief was.



6. Installatie-instructies

6.1. Hardware opstellen

1. Verbind IR-sensoren met pinnen **34 en 32 (GND/VCC)**.
2. Sluit de motoren aan op de **L298N** en verbind deze met de pinnen in de code.
3. Sluit de spanningsdeler aan op analoge pin **33 en op de batterij**.
4. Verbind de startknop tussen pin **15 en GND**.
5. Voed de **ESP32** met **5V** via USB of buck converter.
6. Start de raspberry pi 5 op.

6.2. Software

1. Installeer Arduino IDE met de ESP32 board support.
2. Voeg de volgende libraries toe:
    - WiFi.h
    - PubSubClient.h
3. Vul je **WiFi-SSID en wachtwoord** in de code in.
4. Vul het **IP-adres van de MQTT-broker** in.
5. Open Node-RED via het IP van de raspi.
6. Upload de sketch naar de ESP32.

6.3. Testen

- Open de seriële monitor (115200 baud).
- Druk op de knop om de robot te activeren.
- Zet de robot op een zwart-wit lijnpad.
- Controleer of batterijstatus verschijnt in MQTT-database of -dashboard.
- Observeer of de robot correct bochten maakt en stopt bij verlies van lijn.

7. Stand-alone werking

Na programmeren:

- Verwijder USB
- Sluit een 7.2 V NiMH-batterij aan op de VIN-pin
- ESP32 start automatisch opnieuw, verbindt met Wi-Fi en MQTT en stuurt data

8. 3D prints

We maken gebruik van verschillende 3D prints. Het gebruikte platform is fusion 360.
Hier een korte uitleg over de verschillende files die je op github kan vinden.

Case (zelf ontworpen):

Base car.stl: het grootste deel van de case.
Lid car.stl: het bovenste gedeelte van de case, wordt gebruikt als deksel.
SliderBUTTON.stl: is de schuifdeur die toelaat om de opening tot de knop open of gesloten te houden.
SliderESP.stl: is de schuifdeur die toelaat om de opening tot de ESP open of gesloten te houden.

Deze 4 body's gebruik je om via je 3D print machine te printen.
Indien je wenst de case aan te passen, kan dit doen door Car complete.stl te openen.
Met Car_PCB.step kan je de PCB in je platform voor 3D prints plaatsen.

IR HOLDER (licht aangepast t.o.v. bron):

IR HOLDER modified.stl: deze print zorgt ervoor dat de IR sensors goed vastzitten met een 3D geprinte holder. Dit is zeer aangeraden aangezien de sensors zeer gevoelig zijn.

SONIC HOLDER (ongewijzigd t.o.v. bron):

Met SONIC HOLDER.stl wordt de ultrasonic sensor met de servomotor gecombineerd.

9. PCB

We gebruiken een PCB zodat we minder draden gebruiken en betere connecties maken. Het gebruikt platform is Altium Designer3
Hier een korte uitleg over de verschillende files die je op github kan vinden.

Car.ScbDoc: dit is het schema van de pcb die je eventueel zelf kan aanpassen.
Schematic Print PDF.zip: dit is de PDF file van het schema.

Car.PcbDoc: dit is de pcb zelf die je ook kan aanpassen.
3D PDF.zip: dit is de PDF file van de pcb.

GERBER+NC DRILL.zip: is het zip bestand die we doorgegeven hebben aan jlcpcb om de pcb te maken.

10. GitHub + Demo's + extra

Voor de demo's en alle files van het eindwerk bekijk de GitHub, hier kan je alles vinden om zelf dit werk na te maken.

Link: https://github.com/HaicoL/line-following-robot/tree/main

THOMAS MORE