THOMAS MORE

Robot grasmaaier documentatie

Dylan Geldhof
&
Mathias Vermeulen

2024 - 2025

Inhoudstafel

1. Inleiding

Deze case study behandeld de volgende stappen van een zeflgemaakte, 3D-geprinte robotmaaier.
Hierin wordt een bestaand prototype met al geïnstalleerde hardware en software, verder uitgebreid zodat deze autonoom kan rijden. Via toegevoegde software zal deze robotmaaier een eigen maaiproces hebben om zelf veilig te kunnen rijden, als ook problemen met obstakels op te kunnen lossen.

Extra info op Github: https://github.com/DylanGhf/TM_Robot_Mower
Daar kan je alle code en schema’s terugvinden

2. Bestaand prototype

Motor Driver

Voor dit project is een zelfgemaakte H-brug gebruikt om de motoren aan te sturen. Deze H-brug is opgebouwd uit MOSFETs en wordt aangestuurd via een Attiny die op een I²C-bus is aangesloten.

Bij het ontwerpen van de H-brug is echter de verkeerde soort MOSFETs gekozen voor de high-side switches. In plaats van PNP-MOSFETs zijn NPN-MOSFETs gebruikt. Het ontwerp functioneerde, maar liet niet genoeg spanning door, omdat de gate-spanning op de high-side MOSFETs niet hoog genoeg was (5V in plaats van meer dan 12V). Daarom is uiteindelijk een pre-made H-brug gebruikt. Het zou echter handig zijn geweest om een eigen module te gebruiken. Op de motor driver bevinden zich ook twee IC's om het stroomverbruik en de spanning per motor te meten.

Power-board

Het power-board zet de batterijspanning om naar 5V en 3.3V met behulp van LM1117T IC's. Het bord is ontworpen om gevoed te worden door een 3S LIPO-batterij.

Om de individuele cellen van de batterij te meten, is een JST-poort voorzien. Deze kan echter nog niet worden gebruikt, omdat de spanning te hoog is om direct uit te lezen met de Attiny. Een mogelijke oplossing is een tussenstuk met spanningsdelers om de spanning binnen het bereik van de Attiny te brengen. De Attiny op het power-board communiceert met de ESP32 via de I²C-bus.

MCU

Het MCU-board vormt het brein van de Robot Mower. Hier worden alle signalen van de sensoren ontvangen en worden de actuatoren aangestuurd. De microcontroller op dit board is de ESP32S3 Wroom-1-N8R8, dezelfde chip die wordt gebruikt op het ESP32S3 development board.

Oorspronkelijk was het plan om een custom H-brug via I²C te verbinden, waardoor er geen aansluitingen waren voorzien voor de standaard H-brug. Er zijn echter extra IO-pins voor uitbreidingen beschikbaar, die gebruikt konden worden om de H-brug rechtstreeks met de ESP aan te sturen.

Alle communicatie tussen de MCU en andere componenten, zoals het power-board en de HC-SR04 ultrasone sensoren, verloopt via een I²C-bus. Deze bus maakt het mogelijk berichten te verzenden en ontvangen tussen aangesloten apparaten en voorziet externe modules van stroom via de 5V en 3.3V pinnen. De SCL- en SDA-lijnen hebben een pull-up naar 3.3V en mogen nooit met 5V worden verbonden, om kortsluiting van de ESP32 te voorkomen. Omdat I²C werkt met een pulldown om een digitale 1 te verzenden, blijft dit binnen veilige waarden zolang de aangesloten apparaten een digitale 1 herkennen bij een spanning lager dan 3.3V.

Dit board bevat ook een USB-B connector, die wordt gebruikt om nieuwe programma’s naar de ESP32 te uploaden en om de seriële monitor te gebruiken.

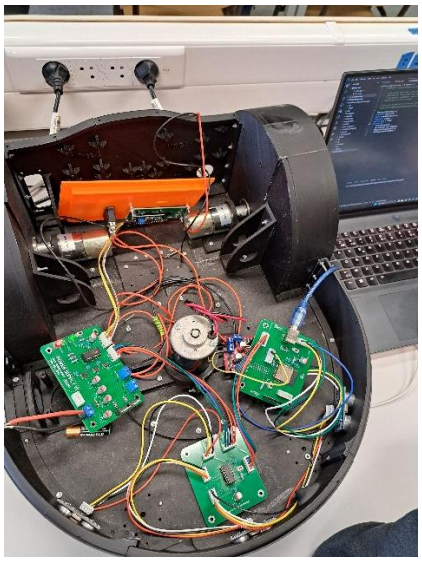
Veel sensoren en IC’s werken via I²C, maar er is ook een SPI-connector voorzien voor het aansluiten van bijvoorbeeld een SD-kaart om data op te slaan.

HC-SR04 Ultrasonic Sensor

Dit board verlicht de rekenbelasting van de MCU door drie HC-SR04 afstandssensoren te laten uitlezen door een Attiny. Hierdoor kan het programma op de ESP32 worden verkleind en wordt de werkdruk verlaagd.

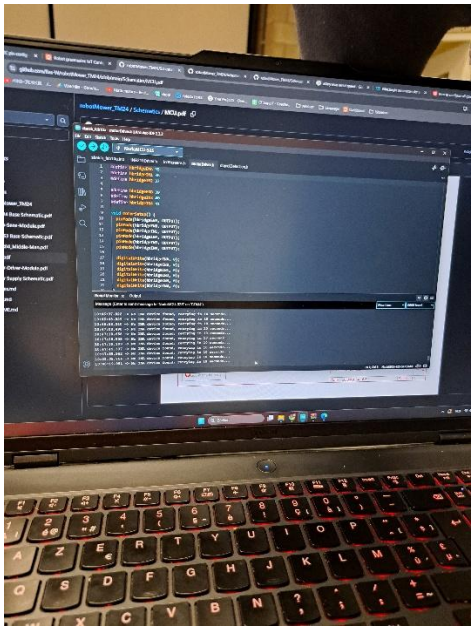
De Attiny heeft geen interne timer, waardoor de sensoren niet op de normale manier kunnen worden uitgelezen. Daarom wordt de "NewPing"-library gebruikt, die een timerfunctie op de Attiny mogelijk maakt. Wanneer de MCU een verzoek doet om 11 bytes aan data, stuurt de Attiny dit geformatteerd door. Op dit moment is er nog geen seriële output mogelijk vanuit de Attiny. Een mogelijke oplossing hiervoor is het creëren van een virtuele UART-poort en het gebruik van een UART-to-USB chip, bijvoorbeeld via een Arduino Uno.

Het PCB is eenvoudig en bevat alleen de essentiële componenten: de Attiny, twee I²C-connectors om door te verbinden en drie connectors voor de HC-SR04 sensoren. Hoewel de lay-out nog geoptimaliseerd kan worden, is het ontwerp functioneel en geslaagd.



3. Troubleshooting

Bij het uploaden van de attiny codes kregen we niet de verwachte resultaten. Op de seriel monitor verscheen er altijd “ *no INA devices found retrying in 10 sec* “. Ook kregen we geen adressen bij het uitvoeren van eeen simpele scanner voor de i²c bus. Na nieuwe verbindingen gemaakt te hebben als ook de attiny’s los gekoppeld te hebben, constateerden we dat deze componenten stuk waren. Me twee nieuwe attiny’s werkten het wel.



Verder hebben we ook alle afstandsensoren vervangen, aangezien deze niet werkte. We hebben dit getest aan de hand van een simpele code en kregen altijd ‘0’ als resultaat op de serieele monitor ongeacht de afstand. Nadit hebben we het programma op een nieuwe sensor gezet, en toen kregen we wel de juiste afstanden. Er steken dus drie nieuwe *hc sr04* sensoren in.

Het grootste probleem waar we tegenliepen, was dat de chip van de esp32 stuk was geraakt. Dit ontdekten we later, nadat we we met een multimeter metingen hadden verricht en geen correcte signalen ontvingen. We gebruiken nu een andere esp32. Deze is verbonden met draadjes en ligt er los in, aangezien we hier geen bord voor hebben.

LET OP de pinnen op de schema’s van de attiny’s zijn verkeerd. Deze staan in spiegelbeeld en werken dus niet deftig wanneer deze gevolgd worden.

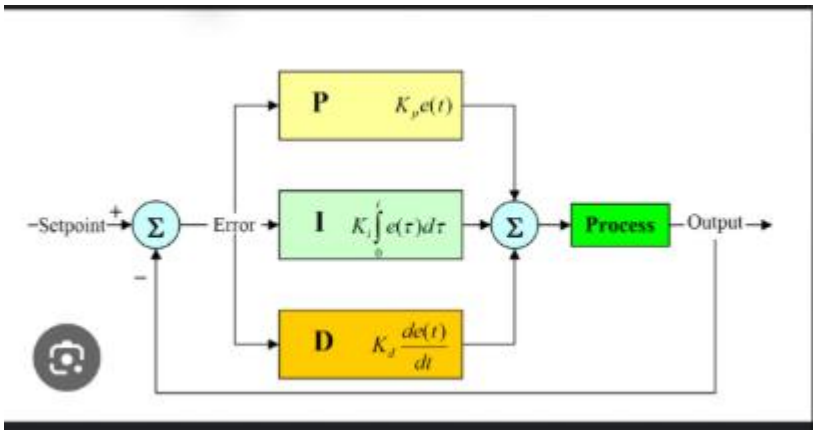
4. Gebruik PID

PID staat voor Proportioneel, Integrerend, Differentieel.
Het is een feedbacksysteem dat wordt gebruikt om systemen te sturen, zoals bijvoorbeeld een robotmaaier, waarbij we de fout (verschil tussen de gewenste en werkelijke waarde) zo klein mogelijk willen houden.
Het doel van PID is om de robot nauwkeurig te sturen door continu te reageren op de afwijkingen van een doelwaarde.

4.1. Basisprincipes PID

PID bestaat uit drie onderdelen, die elk op een andere manier reageren op de fout:

1. **Proportioneel (P)**
    - Reageert op de **huidige fout**.
    - Hoe groter de fout, hoe groter de correctie.
    - Bijvoorbeeld, als de robot ver van het pad is, wordt de correctie groter om snel naar het pad te sturen.
2. **Integrerend (I)**
    - Reageert op de **oplopende fout** over tijd.
    - Het helpt kleine, constante afwijkingen die blijven bestaan, zoals sensorfouten, te corrigeren.
    - Dit voorkomt dat de robot steeds een kleine fout maakt die zich opstapelt.
3. **Differentieel (D)**
    - Reageert op de **verandering in de fout**.
    - Het helpt te voorspellen hoe snel de fout zal toenemen en corrigeert voordat de fout te groot wordt.
    - Dit voorkomt dat de robot te ver van het pad afwijkt of overshoot maakt



4.2. Wiskunde van PID

De PID-regelaar is eenvoudig te begrijpen door naar de formules te kijken. De volledige PID-regelaar kan als volgt worden uitgedrukt:

$$u(t) = K_p \times e(t) + K_i \times \int e(t)\,dt + K_d \times \frac{de(t)}{dt}$$

waarbij:

- $u(t)$is de stuuractie (hoe de robot wordt aangestuurd).
- $e(t)$is de fout (verschil tussen de gemeten waarde en de gewenste waarde).
- K_p, K_I, K_D zijn de respectieve PID-parameters (proportioneel, integrerend en differentieel).
- $\int e(t)dt$ is de cumulatieve fout over tijd (integraal).
- $\frac{de(t)}{dt}$ is de snelheid waarmee de fout verandert (afgeleide).

4.3. Toepassen van PID

Snelheidsregeling

- De robotmaaier moet een constante snelheid behouden, zelfs als het terrein oneffen is. De PID-regelaar kan helpen door de motoren aan te passen zodat de robot niet te snel of te langzaam gaat, ongeacht het terrein.
- De fout e(t)e(t)e(t) zou in dit geval het verschil zijn tussen de gewenste snelheid en de werkelijke snelheid van de robot. Het doel is dat de robot de gewenste snelheid behoudt door PID te gebruiken om de motoren continu aan te passen.

Positieregulatie (zoals een willekeurige beweging)

- Stel je voor dat je robotmaaier een willekeurig pad volgt en constant zijn positie probeert te behouden (bijvoorbeeld in een vierkant of willekeurige beweging in een tuin). De PID-regelaar kan worden gebruikt om te corrigeren als de robot uit zijn pad dreigt te raken.
- De fout kan in dit geval de **hoek** of **positie ten opzichte van een startpunt** zijn. Bijvoorbeeld, als de robot verder van zijn gewenste positie komt, zal de PID-regelaar de stuurmotoren aansteken om de robot terug te brengen naar zijn doelgebied.

Stabiliteit en besturing

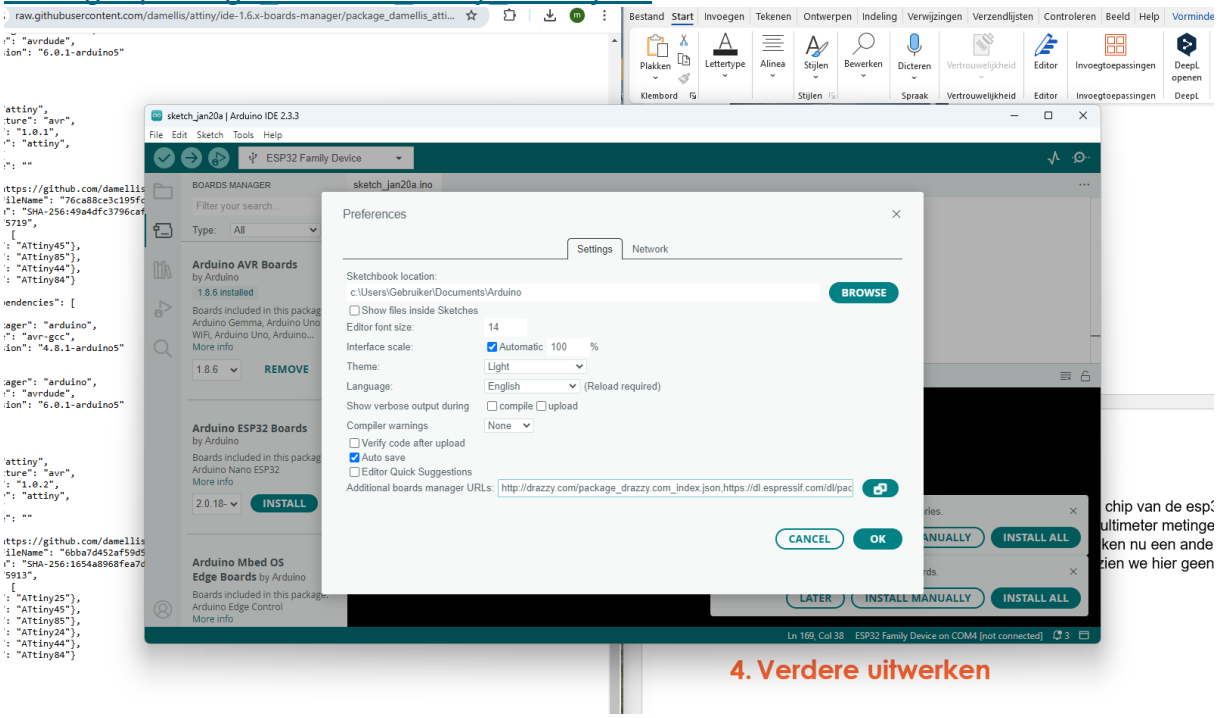
- Stel je voor dat de robotmaaier een bepaald gebied moet afbakenen, maar niet per se een vaste lijn volgt. Hier wordt PID gebruikt om de **hoek** of **oriëntatie** van de robot te stabiliseren,

bijvoorbeeld om ervoor te zorgen dat de robot niet in een cirkel draait of plotseling van richting verandert.

- De fout in dit geval kan het **verschil in hoek** zijn (de huidige oriëntatie van de robot vs. de gewenste oriëntatie).

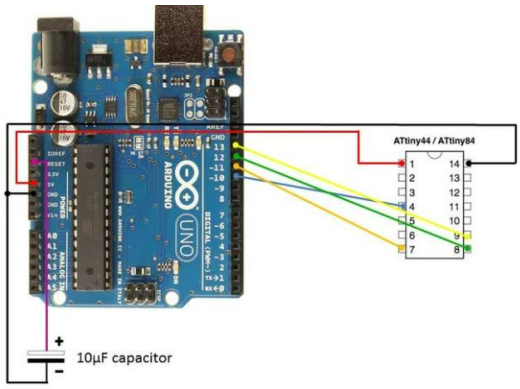
5. Attiny

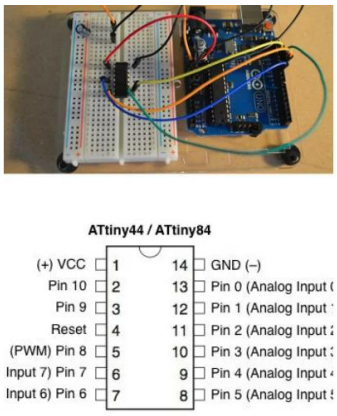
Om de attiny’s te kunnen gebruiken hebben we een nieuwe package moeten installeren. In de arduino ide kan je naar edit gaan en dan naar preferences. Alleen met deze package konden we onze de attiny’s gebruiken
Hier is de package die we hebben toegevoegd:
https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json



Verder hebben we een nieuwe folder toegevoegd waarin we de inhoud van een zipfolder hebben gezet. De inhoud is te downloaden via deze link.
https://github.com/SpenceKonde/ATTinyCore

Dit zijn de aansluitingen die wij gebruikt hebben om het programma te uploaden



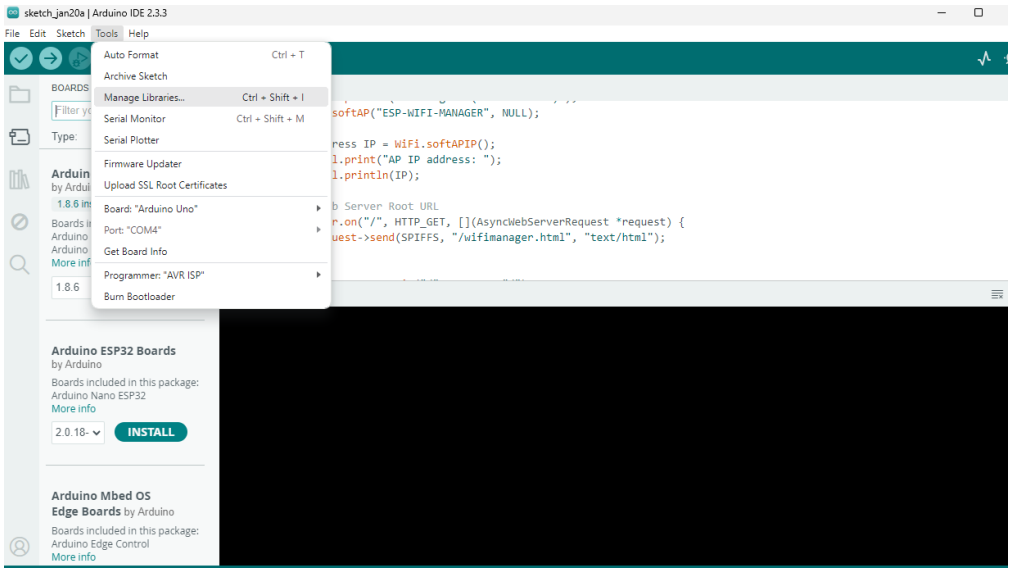


Connect the Arduino Pins to the ATtiny84 pins:

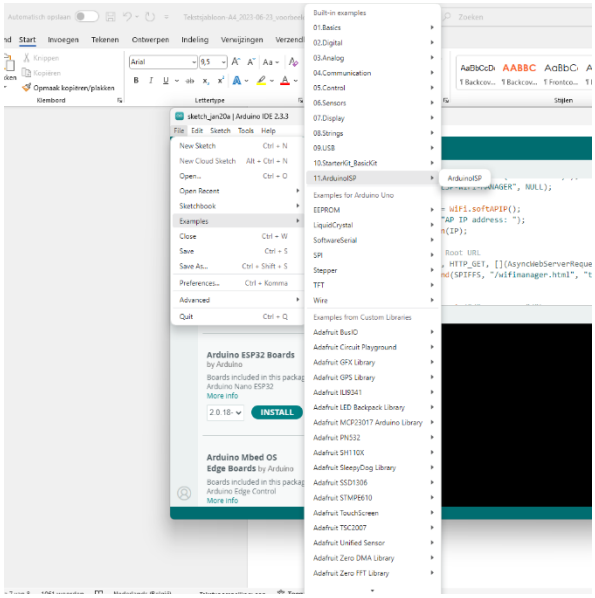
Arduino 5V to ATtiny84 Pin 1
Arduino Pin 10 to ATtiny84 Pin 4
Arduino Pin 11 to ATTiny84 Pin 7
Arduino Pin 12 to ATtiny84 Pin 8
Arduino Pin 13 to ATtiny84 Pin 9
Arduino GND to ATtiny84 Pin 14
Arduino RESET to 10uF capacitor (+ side / long leg)
GND to 10uF capacitor (- side / short leg)

Om de attiny’s te programmeren zijn wij als op volgende manier te werk gegaan.

1) Ga naar tool en bij boards moet de arduino uno aanduiden.
2) Weer bij tool ga je naar programmer en gebruik je AVR ISP

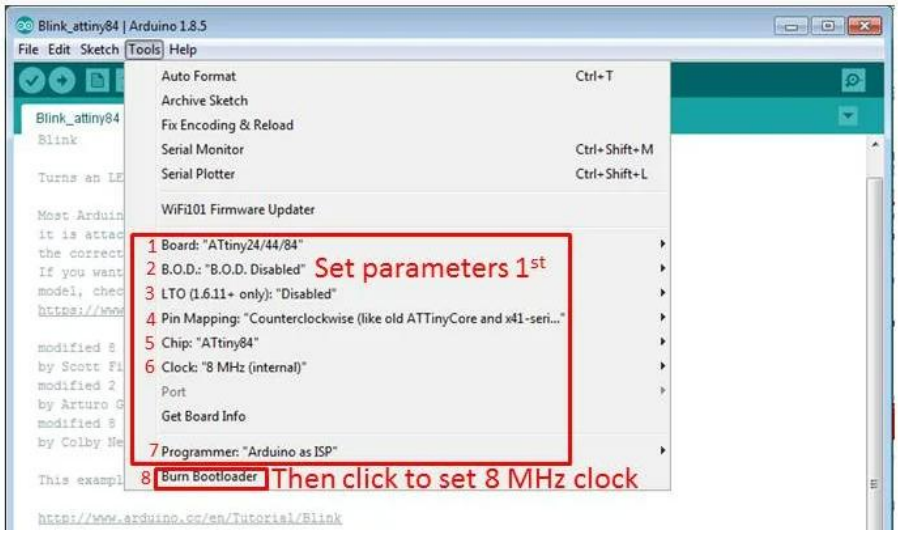


3) Wanneer board en programmer goed staan, kan je bij “file” gaan naarexamples en ga je voor “ArduinoISP”. Er komt een nieuw sketch open en deze kan je dan uploaden.



4) Wanneer dit allemaal in orde is kan je een nieuwe sketch opendoen en volgende code erin plakken

5) Daarna ga je volgende parameters goed zetten volgens onderstaande afbeelding

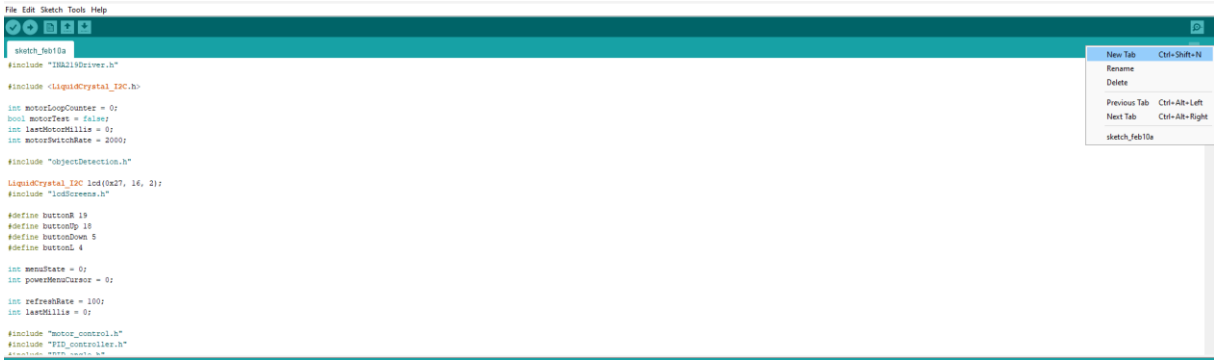


Wanneer alles goed is verlopen kan je volgende code plakken in de nieuwe sketch.

```
#include <Wire.h>      // Inclusie van de Wire-bibliotheek voor I2C-communicatie
#include <NewPing.h>   // Inclusie van de NewPing-bibliotheek voor de ultrasone sensoren

#define MAX_DISTANCE 200  // Maximale meetafstand in cm (sensorrange ligt tussen 400-500 cm)

// Instellen van de linker sensor
#define lTrig 10   // Trigger-pin voor de linker sensor
#define lEcho 9    // Echo-pin voor de linker sensor
NewPing lSonar(lTrig, lEcho, MAX_DISTANCE);  // Initialisatie van de linker ultrasone sensor
unsigned int lDist = 0;  // Variabele om afstandsmeting op te slaan

// Instellen van de middelste sensor
#define mTrig 8// Trigger-pin voor de middelste sensor
#define mEcho 7// Echo-pin voor de middelste  sensor
NewPing mSonar(mTrig, mEcho, MAX_DISTANCE);
unsigned int mDist = 0;

// Instellen van de rechter sensor
#define rTrig 5 // Trigger-pin voor de rechter sensor
#define rEcho 3 // Echo-pin voor de rechter sensor
NewPing rSonar(rTrig, rEcho, MAX_DISTANCE);
unsigned int rDist = 0;

void setup() {
  Wire.begin(8);          // Start de I2C-communicatie met adres 8
  Wire.onRequest(requestEvent);  // Stel een callback-functie in voor I2C-verzoeken
}

void loop() {
  sensorReadout();  // Lees de sensoren uit
}

// Functie om de ultrasone sensoren uit te lezen
void sensorReadout() {
  lDist = lSonar.ping_cm();  // Meet de afstand met de linker sensor
  delay(50);  // Wacht even om interferentie te verminderen

  mDist = mSonar.ping_cm();  // Meet de afstand met de middelste sensor
  delay(50);  // Wacht even om interferentie te verminderen

  rDist = rSonar.ping_cm();  // Meet de afstand met de rechter sensor
  delay(50);  // Wacht even om interferentie te verminderen
}

```

```
44
45  // Functie die wordt aangeroepen wanneer een I2C-verzoek binnenkomt
46  void requestEvent() {
47    String leftValueString = ".";   // Standaardwaarde als geen meting beschikbaar is
48    String middleValueString = ".";
49    String rightValueString = ".";
50
51    // Formatteer de linker sensorwaarde als een 3-cijferige string
52    if (lDist < 10) {
53      leftValueString = "00" + String(lDist);  // Bijvoorbeeld: "007"
54    } else if (lDist < 100) {
55      leftValueString = "0" + String(lDist);   // Bijvoorbeeld: "042"
56    }
57
58    // Formatteer de middelste sensorwaarde
59    if (mDist < 10) {
60      middleValueString = "00" + String(mDist);
61    } else if (mDist < 100) {
62      middleValueString = "0" + String(mDist);
63    }
64
65    // Formatteer de rechter sensorwaarde
66    if (rDist < 10) {
67      rightValueString = "00" + String(rDist);
68    } else if (rDist < 100) {
69      rightValueString = "0" + String(rDist);
70    }
71
72    // Bouw het bericht op in de vorm "xxx|xxx|xxx" (bv. "007|042|123")
73    String message = leftValueString + "|" + middleValueString + "|" + rightValueString;
74
75    Wire.write(message.c_str());  // Stuur het bericht via I2C naar de master
76  }
77
```

Wanneer de de code op de attiny staat kan je deze weer in het pcb bordje steken waar de afstandssensoren aan verbonden zijn. Nu kunnen we verder met de hoofdcode op de esp32 te zetten zodat de robotmaaier rijdt en objecten vermeid afhankelijk van de afstandssensoren. Hieronder staan de gebruikte codes die je moet uploaden. We hebben meer dan een code en gebruiken hiervoor verschillende tabs in de arduino IDE. In de IDE kan je rechtsboven op de drie puntjes drukken en new tab selecteren.



Dan kan je alle onderstaande codes erin zetten. De notites staan beschreven in de code zelf.
Deze code word gebruikt om stroom, spanning en vermogen uit te lezen.de gemeten waarden worden op de seriële monitor gezet.

```ino
#include <INA.h> // Inclusie van de INA-bibliotheek voor stroom-, spanning- en vermogensmetingen

// Definieer de weerstandswaarde van de shunt in micro-ohm
const uint32_t SHUNT_MICRO_OHM{
    6000}; // De shuntweerstand is 6000 micro-ohm (6 milli-ohm), gebruikt voor stroommeting

// Definieer de maximale verwachte stroom
const uint16_t MAXIMUM_AMPS{
    2}; // Verwachte maximale stroom is 2A, binnen het bereik van 1A tot 1022A

// Variabele om het aantal gevonden INA-sensoren bij te houden
uint8_t devicesFound{0}; // Aantal gevonden INA-sensoren op de I2C-bus

// Maak een instantie van de INA_Class aan om gebruik te maken van de EEPROM
INA_Class INA;

// Functie om de INA219-sensoren in te stellen
void INA219Setup() {
  Serial.print("\n\nDisplay INA Readings V1.0.8\n"); // Print softwareversie naar de seriële monitor
  Serial.print(" - Searching & Initializing INA devices\n"); // Meldt dat het op zoek gaat naar sensoren

  // Start de INA-sensoren met de verwachte maximale stroom en shuntweerstand
  devicesFound = INA.begin(MAXIMUM_AMPS, SHUNT_MICRO_OHM);

  // Controleer of er een INA-sensor is gevonden
  while (devicesFound == 0) {
    Serial.println(F("No INA device found, retrying in 10 seconds...")); // Als geen sensor wordt gevonden, wacht 10 seconden en probeer opnieuw
    delay(10000); // Wacht 10 seconden
    devicesFound = INA.begin(MAXIMUM_AMPS, SHUNT_MICRO_OHM); // Probeer opnieuw de sensor te vinden
  }

  // Als een sensor is gevonden, print het aantal gevonden apparaten
  Serial.print(F(" - Detected "));
  Serial.print(devicesFound);
  Serial.println(F(" INA devices on the I2C bus"));

  // Reset de eerste drie sensoren (indien aanwezig)
  INA.reset(0);
  INA.reset(1);
  INA.reset(2);


  // Initialiseer verschillende INA-sensoren met aangepaste instellingen
  INA.begin(5, 6000, 0);      // INA apparaat 0 met max 5A en shuntweerstand van 6000 µΩ
  INA.begin(MAXIMUM_AMPS, 40000, 1); // INA apparaat 1 met max 2A en shuntweerstand van 40000 µΩ
  INA.begin(MAXIMUM_AMPS, 40000, 2); // INA apparaat 2 met max 2A en shuntweerstand van 40000 µΩ

  // Instellen van de conversietijd en meetmodus
  INA.setBusConversion(8500);             // Zet de busconversietijd op max. 8.244ms
  INA.setShuntConversion(8500);           // Zet de shuntconversietijd op max. 8.244ms
  INA.setAveraging(1024);                 // Gemiddeld elke meting 1024 keer
  INA.setMode(INA_MODE_CONTINUOUS_BOTH);  // Continue metingen voor zowel bus- als shuntspanning
  INA.alertOnBusOverVoltage(true, 12000); // Activeer een waarschuwing als de spanning boven de 12V komt
}

// Functie om metingen van de INA219-sensoren uit te lezen
void INA219Read() {
  for (uint8_t i = 0; i < devicesFound; i++) { // Loop door alle gevonden sensoren

    // Lees de meetwaarden uit en sla ze op in een array
    currentSensorReading[i][0] = float(INA.getBusMilliVolts(i) / 1000.0);  // Busspanning in volt
    currentSensorReading[i][1] = float(INA.getShuntMicroVolts(i) / 1000.0); // Shuntspanning in millivolt
    currentSensorReading[i][2] = float(INA.getBusMicroAmps(i) / 1000.0);  // Stroom in milliampère
    currentSensorReading[i][3] = float(INA.getBusMicroWatts(i) / 1000000.0); // Vermogen in watt

    // Print de meetwaarden naar de seriële monitor
    Serial.print(i); // Sensor-ID
    Serial.print("|");
    Serial.print(currentSensorReading[i][0]); // Busspanning
    Serial.print("|");
    Serial.print(currentSensorReading[i][1]); // Shuntspanning
    Serial.print("|");
    Serial.print(currentSensorReading[i][2]); // Stroomsterkte
    Serial.print("|");
    Serial.println(currentSensorReading[i][3]); // Vermogen
  }
}

```

Deze code geeft de verschillende testgegevens en statusgegeven op de lcd.

```
void displayHomescreen() {
  lcd.clear();  // Maak het LCD-scherm leeg
  lcd.setCursor(0, 0);  // Zet de cursor op de eerste rij, eerste kolom
  lcd.print("RobotMower  V1.0");  // Print de naam en versie van de RobotMower
  lcd.setCursor(0, 1);  // Zet de cursor op de tweede rij, eerste kolom
  lcd.print("Nick & Bas");  // Print de namen van de makers
  lcd.setCursor(12, 1);  // Zet de cursor op de tweede rij, kolom 12
  lcd.print("2024");  // Print het jaartal
}

void displayMenu1() {
  lcd.clear();  // Maak het LCD-scherm leeg
  lcd.setCursor(0, 0);  // Zet de cursor op de eerste rij, eerste kolom
  lcd.print("Motor Test");  // Print "Motor Test" op het scherm

  if (motorTest) {  // Controleer of de motortest aan staat
    lcd.setCursor(14, 0);  // Zet de cursor op rij 1, kolom 14
    lcd.print("ON");  // Print "ON" als de motortest actief is
  } else {
    lcd.setCursor(13, 0);  // Zet de cursor op rij 1, kolom 13
    lcd.print("OFF");  // Print "OFF" als de motortest niet actief is
  }
}

void displayCurrentTest(int sensorIndex, float currentSensorReading[3][4]) {
  lcd.clear();  // Maak het LCD-scherm leeg
  lcd.setCursor(0, 0);  // Zet de cursor op de eerste rij, eerste kolom

  // Afhankelijk van de geselecteerde sensor, print de juiste naam
  if (sensorIndex == 0) {
    lcd.print("BAT: ");  // Print "BAT:" voor de batterij
    lcd.setCursor(5, 0);
  } else if (sensorIndex == 1) {
    lcd.print("5V: ");  // Print "5V:" voor de 5V-voeding
    lcd.setCursor(4, 0);
  } else if (sensorIndex == 2) {
    lcd.print("3V3: ");  // Print "3V3:" voor de 3.3V-voeding
    lcd.setCursor(5, 0);
  }

  lcd.print(currentSensorReading[sensorIndex][2]);  // Print de stroom (mA)
  lcd.setCursor(14, 0);
  lcd.print("mA");  // Print de eenheid "mA"
```

```
  lcd.setCursor(3, 1);
  lcd.print(currentSensorReading[sensorIndex][0]);  // Print de spanning (V)
  lcd.setCursor(8, 1);
  lcd.print("V");  // Print de eenheid "V"

  lcd.setCursor(10, 1);
  lcd.print(currentSensorReading[sensorIndex][3]);  // Print het vermogen (W)
  lcd.setCursor(15, 1);
  lcd.print("W");  // Print de eenheid "W"
}
void displaySonarTest(String lValue, String mValue, String rValue) {
  objectDetectionRequest();  // Voer een sonar-meting uit
  // Haal de gemeten waarden op
  lValue = leftValue;
  mValue = middleValue;
  rValue = rightValue;
  lcd.clear();  // Maak het LCD-scherm leeg
  lcd.setCursor(0, 0);
  lcd.print("  L    M    R  ");  // Print labels voor de linker, middelste en rechter sonar
  lcd.setCursor(2, 1);
  lcd.print(lValue);  // Print de waarde van de linker sonar
  lcd.setCursor(7, 1);
  lcd.print(mValue);  // Print de waarde van de middelste sonar
  lcd.setCursor(12, 1);
  lcd.print(rValue);  // Print de waarde van de rechter sonar
}
void displayButtonTest(bool button1Value, bool button2Value, bool button3Value, bool button4Value) {
  lcd.clear();  // Maak het LCD-scherm leeg
  lcd.setCursor(0, 0);
  lcd.print("  1    2    3    4");  // Print de labels voor de knoppen
  lcd.setCursor(2, 1);
  lcd.print(button1Value);  // Print de status van knop 1
  lcd.setCursor(7, 1);
  lcd.print(button2Value);  // Print de status van knop 2
  lcd.setCursor(12, 1);
  lcd.print(button3Value);  // Print de status van knop 3
  lcd.setCursor(14, 1);
  lcd.print(button4Value);  // Print de status van knop 4
}
```

De volgende code dient om de motoren van de robotmaaier de besturen met PWM-signalen, zo kan de robot vooruit, achteruit en bochten maken.

```
#ifndef MOTOR_CONTROL_H
#define MOTOR_CONTROL_H
// Definieer de motorbesturingspinnen
#define IN1  32  // Input voor motor 1 (richting)
#define IN2  33  // Input voor motor 1 (richting)
#define IN3  25  // Input voor motor 2 (richting)
#define IN4  26  // Input voor motor 2 (richting)
#define ENA  27  // PWM-snelheidsregeling voor motor 1
#define ENB  14  // PWM-snelheidsregeling voor motor 2
// Functie om de motoren te initialiseren
inline void motor_init() {
  // Zet de besturingspinnen als uitgang
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);  // Motor 1 inschakelen
  pinMode(ENB, OUTPUT);  // Motor 2 inschakelen
  // Zet beide motoren aan (standaard ingeschakeld)
  digitalWrite(ENA, HIGH);  // Activeer motor 1
  digitalWrite(ENB, HIGH);  // Activeer motor 2
}
// Functie om de motoren te besturen met snelheid (0-100) en hoek (-90 tot 90)
inline void motor_control(int speed, int angle) {
  Serial.print("Speed: ");
  Serial.println(speed);
  Serial.print("Angle: ");
  Serial.println(angle);
  // Beperk de snelheid tot -100 tot 100 en de hoek tot -90 tot 90
  speed = constrain(speed, -100, 100);  // Negatieve snelheid betekent achteruit
  angle = constrain(angle, -90, 90);
  // Converteer snelheid naar PWM-waarde (0-255)
  int pwmValue = map(abs(speed), 0, 100, 0, 255);  // Absolute waarde voor PWM
  int left_speed = pwmValue;
  int right_speed = pwmValue;
  // Pas snelheid aan voor draaien (gebaseerd op hoek)
  if (angle < 0) {  // Draaien naar links
    right_speed = pwmValue + (pwmValue * angle / 90);  // Verlaag snelheid rechter motor
    left_speed = pwmValue;  // Linker motor blijft op volle snelheid
  } else if (angle > 0) {  // Draaien naar rechts
    left_speed = pwmValue - (pwmValue * angle / 90);  // Verlaag snelheid linker motor
    right_speed = pwmValue;  // Rechter motor blijft op volle snelheid
  }
  Serial.print("Left Speed: ");
```

```
  }
  Serial.print("Left Speed: ");
  Serial.println(left_speed);
  Serial.print("Right Speed: ");
  Serial.println(right_speed);
  // Omkeren van de rijrichting als snelheid negatief is
  if (speed < 0) {
    left_speed = -left_speed;
    right_speed = -right_speed;
  }
  // Stel richting en snelheid in voor motor 1 (IN1, IN2)
  if (left_speed > 0) {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);  // Vooruit
  } else if (left_speed < 0) {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);  // Achteruit
  } else {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);   // Stop
  }
  // Stel richting en snelheid in voor motor 2 (IN3, IN4)
  if (right_speed > 0) {
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);  // Vooruit
  } else if (right_speed < 0) {
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, HIGH);  // Achteruit
  } else {
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);   // Stop
  }
  // Pas de snelheid aan via PWM-signalen
  analogWrite(ENA, abs(left_speed));
  analogWrite(ENB, abs(right_speed));
}
#endif // MOTOR_CONTROL_H

```

Dit deel van de code vraagt de afstand van onze attiny die in staat voor de afstandssensoren. Het verwerkt de gegevens zodat objectdetectie mogelijk wordt.

```
void objectDetectionRequest() {

  // Controleer of de ingestelde wachttijd is verstreken voordat een nieuwe aanvraag wordt gedaan
  if (millis() - lastObjectMillis >= delayMS) {
    lastObjectMillis = millis();  // Update de laatste aanvraag-tijd
    Wire.requestFrom(8, 11);  // Vraag 11 bytes op van I2C slave apparaat met adres 8
    Serial.println("Message Sent");  // Debugbericht in de seriële monitor
  }

  char buffer[11];  // Buffer om ontvangen gegevens op te slaan
  int i = 0;  // Teller om de buffer te vullen

  // Wachten tot er gegevens beschikbaar zijn via I2C
  while (Wire.available()) {
    buffer[i] = Wire.read();  // Lees een byte en sla deze op in de buffer
    Serial.print(buffer[i]);  // Toon de byte in de seriële monitor
    i++;

    testFlag = true;  // Zet de vlag om aan te geven dat er data is ontvangen
  }

  // Verwerk de bufferinhoud als er gegevens ontvangen zijn
  if (testFlag) {
    Serial.println("");
    Serial.print("Buffer: ");
    Serial.println(buffer);  // Toon de volledige ontvangen buffer

    // Lees de linker sensorwaarde uit de buffer (eerste 3 tekens)
    leftValue = "";
    for (int r = 0; r < 3; r++) {
      leftValue += buffer[r];
    }

    // Lees de middelste sensorwaarde uit de buffer (tekens 4-6)
    middleValue = "";
    for (int r = 4; r < 7; r++) {
      middleValue += buffer[r];
    }

    // Lees de rechter sensorwaarde uit de buffer (tekens 8-10)
    rightValue = "";
    for (int r = 8; r < 11; r++) {
      rightValue += buffer[r];
    }
```

```
    }

    // Debugberichten met de uitgelezen waarden
    Serial.print("Left Value: ");
    Serial.println(leftValue);
    Serial.print("Middle Value: ");
    Serial.println(middleValue);
    Serial.print("Right Value: ");
    Serial.println(rightValue);

    testFlag = false;  // Reset de vlag
  }
}
```

Deze code bevat een PID-regelaar voor het controleren van de hoek van een systeem, zoals een robotarm of een drone. Het gebruikt drie belangrijke termen (proportioneel, integraal, afgeleid) om de fout tussen de huidige hoek en de doelhoek te corrigeren. De PID-controller berekent de benodigde aanpassing van de hoek en zorgt ervoor dat deze niet te snel verandert, om overshooting te voorkomen. Het systeem heeft ingebouwde limieten en beschermingsmaatregelen, zoals het beperken van de integrale term om windup te voorkomen en het toepassen van een vertraging van 200 milliseconden tussen berekeningen om de stabiliteit van de regeling te waarborgen.

```
#ifndef PID_ANGLE_H
#define PID_ANGLE_H

// PID constanten voor hoekregeling
float kp_angle = 8.0;   // Proportionele gain voor hoekregeling, bepaalt de sterkte van de proportionele reactie
float ki_angle = 1.1;   // Integrale gain voor het elimineren van steady-state fout in de hoekregeling
float kd_angle = 0.1;   // Afgeleide gain om oscillaties te voorkomen in de hoekregeling
float min_angle = -90.0;  // Minimum waarde voor de hoekuitvoer (bijvoorbeeld -90 graden)
float max_angle = 90.0;   // Maximum waarde voor de hoekuitvoer (bijvoorbeeld 90 graden)

// Staat variabelen voor PID-hoekregeling
float target_angle = 0.0;           // Doelhoek die we willen bereiken (-90 tot 90 graden)
float current_angle = 0.0;          // Huidige hoek die wordt aangepast door de PID-regeling
float prev_error_angle = 0.0;       // Vorige foutwaarde voor de afgeleide term van de PID-regeling
float integral_angle = 0.0;         // Integrale accumulator voor hoekcontrole
unsigned long prev_time_angle = 0;  // Vorige tijd voor PID-hoekregeling, wordt gebruikt voor tijdsverschil
float integral_limit_angle = 50.0;  // Limiet voor de integrale term om 'windup' (overmatige ophoping) te voorkomen
float ramp_factor_angle = 0.1;      // Factor die bepaalt hoe snel de hoek naar de doelhoek gaat zonder overshooting
float angle_output = 0.0;           // PID-uitvoer die de hoek stuurt

// Initialisatie van de PID-controller voor hoekregeling
inline void pid_angle_init() {
  prev_time_angle = millis();  // Zet de tijd bij de start van de regeling
  prev_error_angle = 0.0;      // Reset de vorige fout
  integral_angle = 0.0;        // Reset de integrale waarde
  current_angle = 0.0;         // Begin met een hoek van 0 graden
  angle_output = 0.0;          // Begin met een uitgang van 0
}

// Stel de doelwaarde voor de hoek uitgave in (tussen -90 en 90)
inline void set_target_angle(float t) {
  target_angle = constrain(t, min_angle, max_angle);  // Beperk de doelhoek tussen min_angle en max_angle
}

// Bereken de PID-uitvoer voor hoekregeling
inline float compute_pid_angle() {
  unsigned long current_time_angle = millis();  // Verkrijg de huidige tijd voor de hoekregeling
  float dt_angle = (current_time_angle - prev_time_angle) / 1000.0;  // Bereken het tijdsverschil in seconden

  // Update alleen elke 200ms om de vernieuwing van de regeling te controleren
  if (dt_angle < 0.2) return current_angle;  // Wacht totdat het tijdsinterval is verstreken

// Bereken de fout tussen de doelhoek en de huidige hoek
float error_angle = target_angle - current_angle;

// Bereken de proportionele term (p-term)
float p_term_angle = kp_angle * error_angle;

// Bereken de integrale term (i-term) met anti-windup (beperk de integrale waarde)
integral_angle += error_angle * dt_angle;
integral_angle = constrain(integral_angle, -integral_limit_angle, integral_limit_angle);  // Beperk de integrale term
float i_term_angle = ki_angle * integral_angle;
// Bereken de afgeleide term (d-term)
float d_term_angle = kd_angle * ((error_angle - prev_error_angle) / dt_angle);
// Bereken de totale PID-uitvoer voor de hoekregeling
float pid_output_angle = p_term_angle + i_term_angle + d_term_angle;
// Beperk de PID-uitvoer om deze binnen de toegestane hoeken te houden
pid_output_angle = constrain(pid_output_angle, min_angle, max_angle);
// Update de vorige fout en tijd voor de volgende iteratie
prev_error_angle = error_angle;
prev_time_angle = current_time_angle;
// Pas de huidige hoek geleidelijk aan naar de doelhoek zonder overshooting
current_angle += (pid_output_angle - current_angle) * ramp_factor_angle;  // Geleidelijk de hoek aanpassen
// Zorg ervoor dat de huidige hoek de doelhoek niet overschrijdt en stel de hoek direct in als de fout klein is
if (abs(target_angle - current_angle) < 0.5) {  // Als de fout zeer klein is, stel de hoek direct in op de doelwaarde
  current_angle = target_angle;  // Zet de huidige hoek gelijk aan de doelhoek
}

return current_angle;  // Retourneer de aangepaste hoek
}

#endif // PID_ANGLE_H
```

Deze code brengt alles bij elkaar om de robotmaaier autonoom te laten rijden.

code.ino | sketch_jan20a.ino | lcd.ino | motoren.ino | objectdetection.ino | pid.ino | pidcontroller.ino

```
String leftValue;
String middleValue;
String rightValue;

bool testFlag = false;

int currentMillis = 0;
int lastObjectMillis = 0;

#define delayMS 500

#include <Wire.h>

float currentSensorReading[3][4];  // Huidige sensorlezen voor 3 sensoren (bijv. voor spanning)

#include "INA219Driver.h"  // INA219-bibliotheek voor spanningsmetingen

#include <LiquidCrystal_I2C.h>  // LCD-scherm bibliotheek voor I2C-aansluiting

int motorLoopCounter = 0;
bool motorTest = false;
int lastMotorMillis = 0;
int motorSwitchRate = 2000;  // Tijd in milliseconden voor het schakelen van motoren

#include "objectDetection.h"  // Objectdetectiebibliotheek

LiquidCrystal_I2C lcd(0x27, 16, 2);  // LCD-initiatief met adres en afmetingen
#include "lcdScreens.h"  // Bibliotheek voor schermindelingen

// Definiëren van knoppen op de Arduino
#define buttonR 19
#define buttonUp 18
#define buttonDown 5
#define buttonL 4

int menuState = 0;  // Variabele voor de status van het menu
int powerMenuCursor = 0;  // Cursorpositie in het stroommenu

int refreshRate = 100;  // Het interval voor het vernieuwen van scherminformatie
int lastMillis = 0;  // Tijdstip van de laatste update

```

```
#include "motor_control.h"  // Bibliotheek voor motorbesturing
#include "PID_controller.h"  // PID-regelaar voor vermogen
#include "PID_angle.h"  // PID-regelaar voor hoekcontrole

void setup() {
  Serial.begin(115200);  // Start de seriële communicatie
  Serial.println("Boot Start");
  delay(1000);  // Wacht even om systemen tijd te geven op te starten
  Wire.begin();  // Start I2C-communicatie

  lcd.init();  // Initialiseer het LCD-scherm
  lcd.backlight();  // Zet de achtergrondverlichting van het LCD aan

  // Zet knoppen in de juiste modus
  pinMode(buttonL, INPUT_PULLUP);
  pinMode(buttonUp, INPUT_PULLUP);
  pinMode(buttonDown, INPUT_PULLUP);
  pinMode(buttonR, INPUT_PULLUP);

  motor_init();  // Initialiseer de motorbesturing
  pid_init();  // Initialiseer de PID-regelaar voor vermogen
  pid_angle_init();  // Initialiseer de PID-regelaar voor hoekcontrole
  INA219Setup();  // Stel de INA219 in voor spanningsmetingen

  Serial.println("Boot End");
}

void loop() {
  // Als de motorTest is ingeschakeld, voer motorTestControl uit
  if (motorTest) {
    motorTestControl();
  } else {
    // Anders, zet de motoren uit
    motorControl(0, 0, 0);
    motorControl(1, 0, 0);
    motorLoopCounter = 0;
  }

```

```
80  // Menu-systeem voor het schakelen tussen verschillende opties
81  if (menuState == 0) {
82    if (!digitalRead(buttonDown)) {
83      menuState = 1;
84      while (!digitalRead(buttonDown)) {
85      }
86    } else if (!digitalRead(buttonUp)) {
87      menuState = 3;
88      while (!digitalRead(buttonUp)) {
89      }
90    }
91  } else if (menuState == 1) {
92    if (!digitalRead(buttonR)) {
93      motorTest = !motorTest;
94      while (!digitalRead(buttonR)) {
95      }
96    }
97    if (!digitalRead(buttonDown)) {
98      menuState = 2;
99      while (!digitalRead(buttonDown)) {
100     }
101   } else if (!digitalRead(buttonUp)) {
102     menuState = 0;
103     while (!digitalRead(buttonUp)) {
104     }
105   }
106 } else if (menuState == 2) {
107   if (!digitalRead(buttonDown)) {
108     menuState = 3;
109     while (!digitalRead(buttonDown)) {
110     }
111   } else if (!digitalRead(buttonUp)) {
112     menuState = 1;
113     while (!digitalRead(buttonUp)) {
114     }
115   }
116 } else if (menuState == 3) {
117   if (!digitalRead(buttonR)) {
118     if (powerMenuCursor == 2) {
119       powerMenuCursor = 0;
120     } else {
121       powerMenuCursor++;
122     }
```

```
      }
      while (!digitalRead(buttonR)) {
      }
    } else if (!digitalRead(buttonL)) {
      if (powerMenuCursor == 0) {
        powerMenuCursor = 2;
      } else {
        powerMenuCursor--;
      }
      while (!digitalRead(buttonL)) {
      }
    }
    if (!digitalRead(buttonDown)) {
      menuState = 0;
      while (!digitalRead(buttonDown)) {
      }
    } else if (!digitalRead(buttonUp)) {
      menuState = 2;
      while (!digitalRead(buttonUp)) {
      }
    }
  }

  // Als het interval voor vernieuwing is verstreken, update het systeem
  if ((millis() - lastMillis) > refreshRate) {
    lastMillis = millis();
    objectDetectionRequest();  // Vraag objectdetectie aan
    set_target_angle(calculate_target_angle(float front_distance, float left_distance, float right_distan
    power_output = compute_pid();  // Bereken de PID-uitvoer voor vermogen
    float angle_output = compute_pid_angle();  // Bereken de PID-uitvoer voor de hoek

    // Print de snelheid uit
    Serial.print("Speed Output: ");
    Serial.print(power_output);
    motor_control(power_output, angle_output);  // Zet de motoren aan met de berekende snelheden

```

```
    // Afhankelijk van de huidige menu-status, toon het juiste scherm
    switch (menuState) {
      case 0:
        displayHomescreen();  // Toon het beginscherm
        break;
      case 1:
        displayMenu1();  // Toon menu 1
        break;
      case 2:
        displaySonarTest("", "", "");  // Toon sonar-test scherm
        break;
      case 3:
        INA219Read();  // Lees de INA219-sensor
        displayCurrentTest(powerMenuCursor, currentSensorReading);  // Toon het huidige testscherm
        break;
    }
  }
}

// Functie om de doelhoek te berekenen op basis van afstanden van de sensoren
float calculate_target_angle(float front_distance, float left_distance, float right_distance) {
  // Drempelwaarden voor de sensoren
  const float LEFT_SENSOR_THRESHOLD = 20.0;  // 20 cm
  const float RIGHT_SENSOR_THRESHOLD = 20.0; // 20 cm
  const float MAX_ANGLE = 90.0;  // Maximale hoek (volle draai)

  // Functie om een sensorafstand naar een hoek te mappen
  auto map_distance_to_angle = [](float distance, float min_distance, float max_distance, float min_angle, fl
    if (distance > max_distance) {
      return min_angle;  // Als de afstand groter is dan max_distance, geef de min-hoek (geen draai)
    }
    if (distance < min_distance) {
      return max_angle;  // Als de afstand kleiner is dan min_distance, geef de max-hoek (volle draai)
    }
    // Map de afstand naar een hoek in een lineaire manier
    return map(distance, min_distance, max_distance, min_angle, max_angle);
  };

  float target_angle = 0.0;  // Standaard doelhoek (recht vooruit)

  // Als een object gedetecteerd wordt aan de linkerkant
  if (left_distance < LEFT_SENSOR_THRESHOLD) {
    // Map de afstand van de linker sensor naar een doelhoek (gladde overgang van 0 naar 90 graden)
    target_angle = map_distance_to_angle(left_distance, 10.0, 50.0, 0.0, MAX_ANGLE);
  }
  // Als een object gedetecteerd wordt aan de rechterkant
  else if (right_distance < RIGHT_SENSOR_THRESHOLD) {
    // Map de afstand van de rechter sensor naar een doelhoek (gladde overgang van 0 naar -90 graden)
    target_angle = map_distance_to_angle(right_distance, 10.0, 50.0, 0.0, -MAX_ANGLE);
  }

  // Als geen objecten gedetecteerd worden, blijf recht (0 graden)
  return target_angle;
}
```

THOMAS MORE