Alen Iriškić & Hugo Ooms

1051158@hr.nl & 1055574@hr.nl

Abstract

In dit project werkte wij aan een automatic container mover die zonder user input containers van punt a naar punt b kunnen brengen

acm project

School project 2

Table of Contents

[Inleiding 2](#_Toc125544305)

[Productspecificaties 2](#_Toc125544306)

[Product- en sprintbacklog 3](#_Toc125544307)

[Elektrisch schema 4](#_Toc125544308)

[Flowchart & Code 5](#_Toc125544309)

[Schetsen & prototypes 5](#_Toc125544310)

[Testplan & testrapport 18](#_Toc125544311)

[Reflectie 21](#_Toc125544312)

[Alen Iriškić 21](#_Toc125544313)

[Hugo Ooms 21](#_Toc125544314)

# Inleiding

In dit project moeten we een Automatic Container Mover (ACM) maken, de ACM moet van punt A naar een punt B reiden zonder user input. Om dat te kunnen doen is er parallel op de grond zwarte tape aangebracht, waartussen de ACM zich moet bewegen.

# Productspecificaties

# Product- en sprintbacklog

# Elektrisch schema

# **Diagram, schematic Description automatically generated**

# 

# Flowchart & Code



#include <WebServer.h>

#include <ESPmDNS.h>

#include <Arduino.h>

#include "../lib/Analog.h"

#include "../lib/MotorDriver.h"

#include <WiFi.h>

#define SOUND\_SPEED 0.034

#define CM\_TO\_INCH 0.393701

const *int* trigPin = 21;

const *int* echoFront = 34;

const *int* echoBack = 35;

const *int* echoRight = 14;

const *int* echoLeft = 12;

const *int* leftFrontIR = 36;

const *int* rightFrontIR = 39;

const *int* rightBackIR = 5;

const *int* leftBackIR = 27;

// Motor B linksvoor

const *int* linksVoor1 = 32; // Pin

const *int* linksVoor2 = 33; // Pin

const *int* linksVoorPWM = 16; // Pin

// Motor C linksachter

const *int* linksAchter1 = 25; // Pin

const *int* linksAchter2 = 26; // Pin

const *int* linksAchterPWM = 15; // Pin

// rechtsvoor

const *int* rechtsVoor1 = 23; // Pin

const *int* rechtsVoor2 = 22; // Pin

const *int* rechtsVoorPWM = 4; // Pin

// Motor D rechtsachter

const *int* rechtsAchter1 = 18; // Pin

const *int* rechtsAchter2 = 19; // Pin

const *int* rechtsAchterPWM = 17; // Pin

*long* duration;

*float* distanceCm;

*int* state = 1;

*float* distanceDetection(*int* *pin*)

{

// Clears the trigPin

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

duration = pulseIn(*pin*, HIGH);

*float* distanceCm = duration \* SOUND\_SPEED / 2;

return distanceCm;

}

*void* setup()

{

// Set pins as outputs

pinMode(rechtsVoor1, OUTPUT);

pinMode(rechtsVoor2, OUTPUT);

pinMode(rechtsVoorPWM, OUTPUT);

pinMode(linksVoor1, OUTPUT);

pinMode(linksVoor2, OUTPUT);

pinMode(linksVoorPWM, OUTPUT);

pinMode(linksAchter1, OUTPUT);

pinMode(linksAchter2, OUTPUT);

pinMode(linksAchterPWM, OUTPUT);

pinMode(rechtsAchter1, OUTPUT);

pinMode(rechtsAchter2, OUTPUT);

pinMode(rechtsAchterPWM, OUTPUT);

pinMode(trigPin, OUTPUT);

pinMode(echoFront, INPUT);

pinMode(echoBack, INPUT);

pinMode(leftFrontIR, INPUT);

pinMode(rightFrontIR, INPUT);

Serial.begin(9600);

}

*void* stop()

{

// Move forwards

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksVoorPWM, LOW);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, LOW);

digitalWrite(linksAchterPWM, LOW);

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsVoorPWM, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, LOW);

digitalWrite(rechtsAchterPWM, LOW); // set LED brightness as PWM signal

}

*void* moveForward(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

}

}

*void* moveBackward(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, HIGH);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, HIGH);

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, HIGH);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, HIGH);

analogWrite(linksVoorPWM, 100);

analogWrite(linksAchterPWM, 100);

analogWrite(rechtsVoorPWM, 100);

analogWrite(rechtsAchterPWM, 100);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, HIGH);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, HIGH);

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, HIGH);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, HIGH);

analogWrite(linksVoorPWM, 100);

analogWrite(linksAchterPWM, 100);

analogWrite(rechtsVoorPWM, 100);

analogWrite(rechtsAchterPWM, 100);

}

}

}

*void* moveDiagonalRight(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 0);

analogWrite(rechtsVoorPWM, 0);

analogWrite(rechtsAchterPWM, 250);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 0);

analogWrite(rechtsVoorPWM, 0);

analogWrite(rechtsAchterPWM, 250);

}

}

}

*void* moveDiagonalLeft(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 0);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 0);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 0);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 0);

}

}

}

*void* moveLeft(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, HIGH);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, HIGH);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, HIGH);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, HIGH);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

}

}

*void* moveRight(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, HIGH);

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, HIGH);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, HIGH);

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, HIGH);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

}

}

*void* turnLeft(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, HIGH);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, HIGH);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(rechtsVoor1, HIGH);

digitalWrite(rechtsVoor2, LOW);

digitalWrite(linksVoor1, LOW);

digitalWrite(linksVoor2, HIGH);

digitalWrite(linksAchter2, LOW);

digitalWrite(linksAchter1, HIGH);

digitalWrite(rechtsAchter2, HIGH);

digitalWrite(rechtsAchter1, LOW);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

}

}

*void* turnRight(*int* *x*)

{

if (*x* == 0)

{

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, HIGH);

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, HIGH);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

else

{

static *unsigned* *long* startTime = millis();

if (millis() - startTime < *x*)

{

digitalWrite(rechtsVoor1, LOW);

digitalWrite(rechtsVoor2, HIGH);

digitalWrite(linksVoor1, HIGH);

digitalWrite(linksVoor2, LOW);

digitalWrite(linksAchter2, HIGH);

digitalWrite(linksAchter1, LOW);

digitalWrite(rechtsAchter2, LOW);

digitalWrite(rechtsAchter1, HIGH);

analogWrite(linksVoorPWM, 250);

analogWrite(linksAchterPWM, 250);

analogWrite(rechtsVoorPWM, 250);

analogWrite(rechtsAchterPWM, 250);

}

}

}

*void* hugRightLine()

{

while (digitalRead(rightFrontIR) && digitalRead(rightBackIR))

{

moveRight(0);

}

while (!digitalRead(rightFrontIR) && digitalRead(rightBackIR))

{

turnLeft(0);

}

while (digitalRead(rightFrontIR) && !digitalRead(rightBackIR))

{

turnRight(0);

}

stop();

}

*void* hugLeftLine()

{

while (digitalRead(leftBackIR) && digitalRead(leftFrontIR))

{

moveLeft(0);

}

while (!digitalRead(leftBackIR) && digitalRead(leftFrontIR))

{

turnLeft(0);

}

while (digitalRead(leftBackIR) && !digitalRead(leftFrontIR))

{

turnRight(0);

}

}

*void* spaceWalk()

{

}

*void* loop()

{

switch (state)

{

case 0: // Backward

moveBackward(0);

break;

case 1: // Forward

moveForward(0);

break;

case 2: // Diagonal right

moveDiagonalRight(0);

break;

case 3: // Diagonal left

moveDiagonalLeft(0);

break;

case 4: // Left

moveLeft(0);

break;

case 5: // Right

moveRight(0);

break;

case 6: // Turn left

turnLeft(0);

break;

case 7: // Turn right

turnRight(0);

break;

default: // Stop

stop();

}

}

Schetsen & prototypes

## Diagram Description automatically generatedV1

Diagram, engineering drawing

Description automatically generated

Chart

Description automatically generatedDiagram, engineering drawing

Description automatically generated

Boven is onze eerste versie van onze voor- en achterkant te zien. Hij was nog erg hoekig en niet aerodynamisch. Functioneel was die wel al compleet; het was mogelijk om alle sensoren te bevestigen op de ACM met het eerste design.

## Diagram, engineering drawing Description automatically generatedV2

Diagram, engineering drawing

Description automatically generated

Diagram

Description automatically generatedA picture containing text, appliance

Description automatically generated

Hier is onze tweede versie te zien. Het tweede design is aerodynamisch geoptimaliseerd en minder massief gemaakt. Functioneel hebben er geen veranderingen plaatsgevonden, het kan nog steeds twee ultrasone-sensoren en twee infrarood-sensoren huizen.

## 

## Diagram, engineering drawing Description automatically generatedV3

Diagram

Description automatically generated

Diagram, engineering drawing

Description automatically generated

In versie drie (de laatste versie) is er enorm veel bijgekomen, er is een accuhouder toegevoegd op de bovenkant, een container houder op de bovenkant, ultrasone-sensoren op beide zijkanten, batterijhouders op de onderkant, 4 motorhouders aan de onderkant en een breadboard houder in het midden.

# Testplan & testrapport

Om te voorkomen dat de ACM onverwacht gedrag toont wanneer het afgeleverd wordt aan de klant, wordt er eerst een testplan opgesteld en uitgevoerd. In het testplan staan alle voorwaarden waar de ACM aan moet voldoen maar ook de situaties die het kan tegenkomen.

Het testplan bestaat uit drie onderdelen (kolommen): de situatie, het verwacht gedrag van de ACM, en het werkelijk gedrag.

De situatie moet goed worden omschreven zodat het zo goed mogelijk getest kan worden, er moet in staan: Wat, Waar en waarom.

Het verwachte gedrag moet zo duidelijk mogelijk omschreven worden zodat het duidelijk is wat de klant wil.

Het werkelijke gedrag moet alleen als het niet overeenkomt met het verwachte gedrag duidelijk beschreven worden om zoveel mogelijk informatie te hebben voor het verbeteren van de ACM.

## Testplan

|  |  |  |
| --- | --- | --- |
| Situatie | Verwacht gedrag | Werkelijk gedrag |
| 1. De ACM kan door een tunnel rijden |  |  |
| 1. De ACM moet op een accu rijden |  |  |
| 1. De ACM kan over heuvels van minstens 20% rijden |  |  |
| 1. De ACM kan binnen 5 minuten de containers naar de bestemming op het haventerrein van 3 bij 3 meter brengen |  |  |
| 1. De ACM kan na een val van 50mm verder rijden |  |  |
| 1. De ACM kan binnen de lijnen van het rijvlak rijden |  |  |
| 1. De ACM kan obstakels ontwijken |  |  |
| 1. De ACM kan bij de kade(afgrond groter dan 5cm) keren |  |  |
| 1. De ACM kan 2 containers van 60 bij 30 bij 30 mm vervoeren |  |  |
| 1. De ACM kan magnetische strepen op het terrein detecteren als herkenningspunten |  |  |
| 1. De ACM kan zelfstandig (‘autonoom’) rijden |  |  |
| 1. De ACM kan op afstand worden gestart en gestopt |  |  |
| 1. De Operator kan de sensoren op afstand monitoren |  |  |
| 1. De ACM moet containers tussen 2 punten op het haventerrein vervoeren |  |  |

# Reflectie

## Alen Iriškić

## Hugo Ooms