Inhoudsopgave

[Inleiding 3](#_Toc465690138)

[Context en Methodologie 4](#_Toc465690139)

[Inleiding 4](#_Toc465690140)

[Doelstelling van het onderzoek 4](#_Toc465690141)

[Verzameling van data 4](#_Toc465690142)

[Theorie 5](#_Toc465690143)

[PID Controller 5](#_Toc465690144)

[Verloop van het onderzoek 8](#_Toc465690145)

[Informatie verzamelen 11](#_Toc465690146)

[Kaart samenstellen 12](#_Toc465690147)

[Resultaten 14](#_Toc465690148)

[Reflectie 15](#_Toc465690149)

[Aanbevelingen 16](#_Toc465690150)

[BIJLAGEN 17](#_Toc465690151)

[Bijlage 1 Literatuurlijst 17](#_Toc465690152)

# Inleiding

De minor Business Innovation van de Haagse Hogeschool biedt een scala aan opdrachten waar studenten aan kunnen werken. Tijdens deze opdrachten kunnen ze onderzoek doen naar nieuwe technieken en technologieën en deze in een praktische omgeving testen. De opdracht die wij gekozen hebben heette het Cooperative Arduino Maze Solver Project.

Aan het begin van het onderzoek werden de white paper en de code van het voorgaande onderzoek aan ons gegeven. Ook kregen we alle onderdelen van de robot die de vorige onderzoeksgroep had achtergelaten.

Het voorgaande onderzoek van het project voegde een radio en een aantal vlakken toe aan het project. Deze gaven een opzet voor de communicatie en het maken van een testdoolhof. Hiermee zijn wij verder gaan ontwikkelen.

# Context en Methodologie

## Inleiding

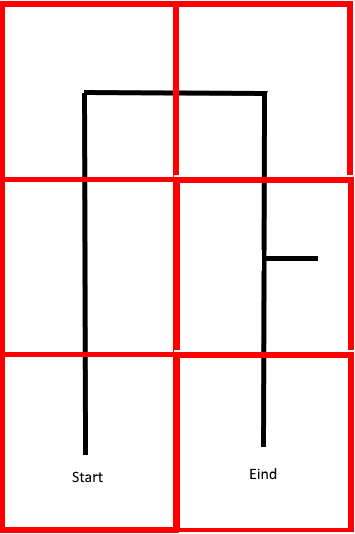
Dit onderzoek is onderdeel van het Cooperative Arduino Maze Solver Project. Het doel van dit project is om een aantal robots te programmeren die informatie van een doolhof lezen en deze vervolgens met elkaar delen. Deze gedeelde informatie moet er vervolgens toe leiden dat de robots de oplossing van het doolhof zo snel mogelijk kunnen vinden.

## Doelstelling van het onderzoek

De doelstelling van dit onderzoek was om de code van de Arduino's verder te ontwikkelen zodat deze informatie van het doolhof met elkaar delen en een kaart van het doolhof kunnen tekenen.

## Verzameling van data

Om de data van de functionaliteit en de betrouwbaarheid van de robots te verzamelen hebben we een testdoolhof opgezet. Vervolgens hielden we telkens bij hoeveel fouten de robot maakte en hoelang hij er over deed om door het hele testdoolhof te rijden. Hieronder een plaatje van het testdoolhof.



**Figuur 1 Abstracte weergave van het gebruikte testdoolhof.**

De rode lijnen geven de individuele vlakken aan.

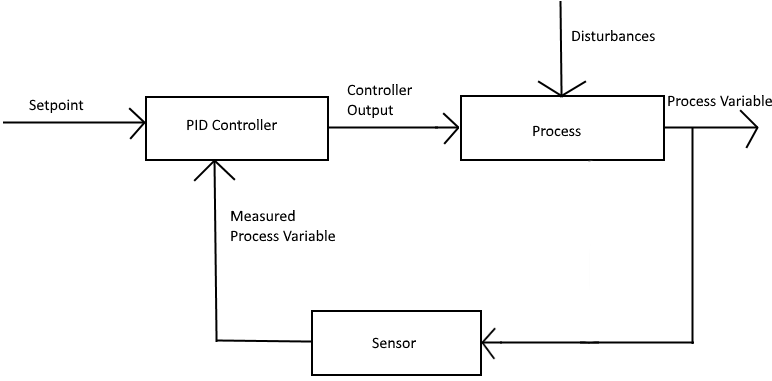
# Theorie

Het uiteindelijk doel van het project is om een aantal rijdende robots in een doolhof te plaatsen die vervolgens met elkaar samenwerken om zo snel mogelijk het doolhof op te lossen. Om dit mogelijk te maken zijn er verschillende methodes om de robots te laten communiceren.

Wij hebben gekeken naar de mogelijkheid om meerdere zendende robots in het doolhof te hebben die informatie verzamelen en doorsturen en een centrale ontvanger die de informatie verwerkt. Het voordeel van deze methode was dat de centrale ontvanger vastgemaakt kon worden aan een laptop. Daardoor hadden we veel meer geheugen tot onze beschikking dan de hoeveelheid die op een Arduino bordje paste. Hierdoor zou de centrale ontvanger gemakkelijk alle informatie van de zenders op kunnen vangen en tot een kaart kunnen verwerken voor elke individuele zender.

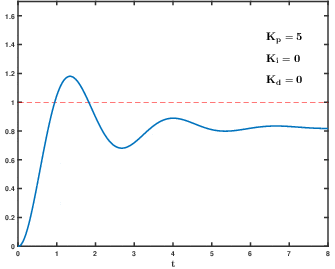
### PID Controller

Daarnaast hebben we ook gekeken naar een betere manier om de robot de lijn te laten volgen. Aangezien we tijdens een aantal vroege testen erachter kwamen dat het volgende van de lijn niet erg goed werkte, hebben we een Proportional Integral Derivative Controller (PID Controller) in de code geïmplementeerd. Met behulp van deze controller bleef de robot een stuk beter de lijn volgen.

Een PID Controller gaat uit van een bepaalde "Setpoint (SP)". Dit is de doelwaarde van de controller. Deze waarde vergelijkt de controller met de "Process Variable (PV)" ook wel de werkelijk gemeten waarde van de sensor waar het proces van de controller aan vast zit[9]. Als de waardes overeen komen hoeft de controller niets te doen, maar als de waardes niet overeen komen moet de controller in actie komen. **Figuur 2 Schematische weergave van de werking van een PID Controller**

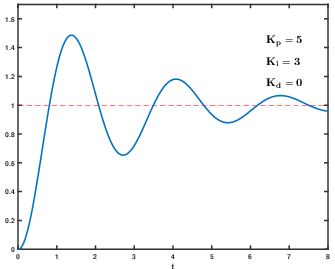
De controller gaat proberen om het verschil tussen de SP en de PV te verkleinen. Dit doet hij door de PV aan te passen aan de hand van het verschil, als de SP hoger is dan de PV dan moet de waarde van de PV groter worden en andersom.

Hoeveel de PV per iteratie van het proces wordt aangepast gebeurt aan de hand van drie andere waardes, namelijk de Proportional, Intergral en Derivative. De controller gebruikt de waarde van de Proportional om het gemeten verschil te vermenigvuldigen. Het vermenigvuldigde verschil wordt vervolgens opgeteld bij de PV. Dit zorgt ervoor dat de PV rond de SP gaat oscilleren zodat het proces minder gaat afwijken van de gewenste SP.



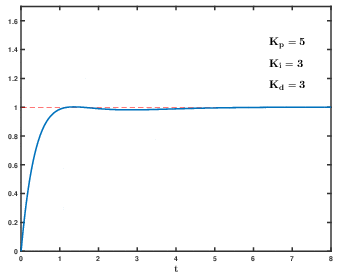
**Figuur 3 Voorbeeld van de grafiek van een PID Controller waarbij alleen de Proportional gebruikt wordt[4].**

Vervolgens wordt voor de Integral het totale verschil opgeteld van alle iteraties die het proces doorlopen heeft. Dit totaal wordt vervolgens met de waarde van de Integral vermenigvuldigd en daarna ook opgeteld bij de PV. Dit leidt ertoe dat de oscillaties groter worden, waardoor de PV rond de SP blijft oscilleren.



**Figuur 4 Voorbeeld van de grafiek van een PID Controller waarbij de Proportional en de Intergral gebruikt worden[4].**

Ten slotte is er nog de Derivative. Met de Derivative probeert de controller het verschil van zijn volgende iteratie te voorspellen. Voor de Derivative wordt het vorige verschil afgetrokken van het huidige verschil. Vervolgens wordt het resultaat vermenigvuldigd met de waarde van de Derivative en daarna opgeteld bij de PV. Hierdoor worden de oscillaties van de PV kleiner zodat de PV dichter in de buurt blijft van de SP. In het geval van ons onderzoek was de SP dat de lijn precies onder de robot doorliep en de PV de hoeveelheid stroom die naar de motoren geleid werd.



**Figuur 5 Voorbeeld van de grafiek van een PID Controller waarbij de Proportional, de Integral en de Derivative gebruikt worden[4].**

# Verloop van het onderzoek

De code die we hadden ontvangen van het voorgaande onderzoek bood nog niet heel betrouwbare resultaten. Het liet zien hoe de radio's werkten en deed al een poging om informatie van een rijdende Arduino naar een luisterende Arduino te versturen. De informatie die verstuurt werd was echter lang niet altijd de juiste informatie waardoor de luisteraar er niet veel mee kon. Deze situatie kon verbeterd worden door het ontwerp van de rijdende robots aan te passen. Het ontwerp dat de vorige groep gebruikte bestond uit vier batterijen als stroomtoevoer en drie sensoren om de lijn te volgen. Met dit ontwerp was de robot erg traag en kon hij alleen de lijn volgen door er telkens overeen te zigzaggen.

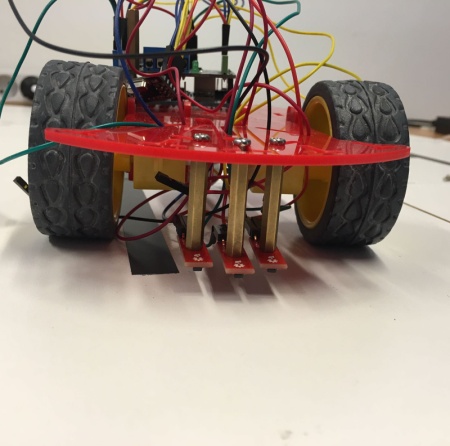
De gemiddelde spanning die vier oplaadbare batterijen kunnen leveren is 4.8V terwijl de aanbevolen spanning voor een Arduino Uno bord 7-12V is[7]. Daar kwam nog bij dat aan de rijdende robot drie stroomvretende sensoren waren bevestigd. Het gevolg was dat de robot weinig stroom over hield om zijn motoren aan te sturen. Dit maakte het moeilijk om de robots goed te testen aangezien ze telkens tot stilstand kwamen. Daarom zijn we als eerst gaan kijken naar het verbeteren van de stroomtoevoer. Het was misschien mogelijk om een accu aan de robot toe te bevestigen die een hogere spanning kon leveren. Een accu zou ook een oplaadbare stroombron vormen, maar het bleek dat deze al snel een grote investering zou zijn. Aangezien er geen budget voor het onderzoek bestond was een accu geen realistische mogelijkheid.

Wat een andere, goedkopere oplossing zou zijn was om twee batterijhouders in serie te schakelen. Het onderzoek had wel beschikking over een groot aantal batterijhouders en twee gevulde batterijhouders zouden acht batterijen kunnen bevatten die samen een spanning van 9,6V zouden leveren. Met deze spanningsverhoging behaalden we al snel veel betere resultaten.

Vervolgens zijn we gaan kijken naar het verbeteren van het volgen van de lijn. De rijdende robot in het voorgaande onderzoek gebruikte drie Analog Line Sensors[2] om de lijn te blijven volgen. De code ging ervan uit dat als de robot een bocht tegenkwam, dat oriëntatie van de robot ten opzichte van de lijn bijna parallel was. Maar tijdens het testen van die code werd al snel duidelijk dat die situatie heel weinig voorkwam. Hierdoor gaf de robot al snel de verkeerde informatie door aan de luisteraar.

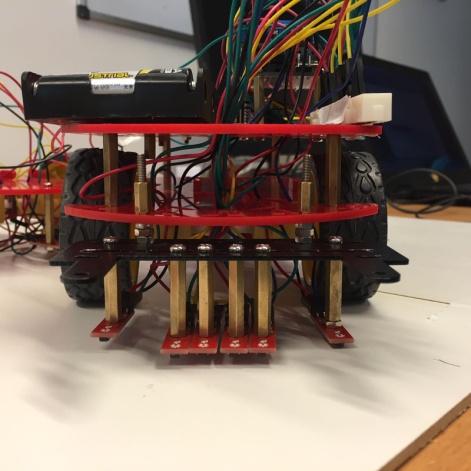
We zijn daarom gaan brainstormen om het lijn volgen van de robots betrouwbaarder te maken. We zijn nagegaan of de onderdelen van de robot wel goed werkten en hun functie vervulden. Dit hebben we gedaan door de bedrading en de waardes die de sensoren gaven te controleren. Het bleek dat de sensoren wel goed functioneerden, maar dat ze maar een kleine marge hadden waarin ze konden werken nadat ze gekalibreerd waren door de code.   
Na het natrekken van de specificaties van de sensoren bleek ook dat de optimale detectie afstand van een Analog Line Sensor 3mm was. De sensoren op onze robot zaten op 5mm hoogte. Hierdoor kregen de sensoren veel extra licht binnen en werden de waardes van het wit van de vlakken een stuk hoger. We hebben daarom de bevestiging van de sensoren aan de robot aangepast zodat de sensoren dichter bij de optimale 3mm kwamen. Hierdoor werden de witwaardes van de sensoren een stuk lager en werd dus het verschil tussen het wit van een vlak en de zwarte lijn een stuk groter.

Daarna zijn we gaan kijken hoe we de beweging en de berichtgeving van de robot konden verbeteren. We bedachten dat drie sensoren niet voldoende waren om betrouwbare berichtgeving te garanderen, omdat de robot alle drie de sensoren al gebruikte om de lijn te kunnen volgen. We kwamen daarom tot de conclusie dat er minstens vijf sensoren nodig zouden zijn om en de lijn te kunnen volgen en bochten te kunnen herkennen. De binnenste drie sensoren zouden dan het volgen van de lijn regelen en de buitenste twee het herkennen van bochten.



**Figuur 6 Foto's van de robot met drie en vijf sensoren**

De robot reed alleen nog steeds niet stabiel genoeg om goed gebruik te kunnen maken van de lijn volgende sensoren. De robot zigzagde nog steeds over de lijn waardoor hij vaak scheef aankwam bij een bocht. Om die stabiliteit te verhogen zijn we op zoek gegaan naar voorbeelden van andere "line followers"[3][6]. Het bleek dat veel van deze voorbeelden in de code gebruik maakten van een Proportional Integral Derivative Controller (zie hfdst Theorie PID Controller). Deze controller bevatte een algoritme waarmee de lijn vrij precies gevolgd kon worden. Op basis van deze voorbeelden hebben we onze eigen controller ontwikkeld en nog een zesde sensor aan de robot toegevoegd.



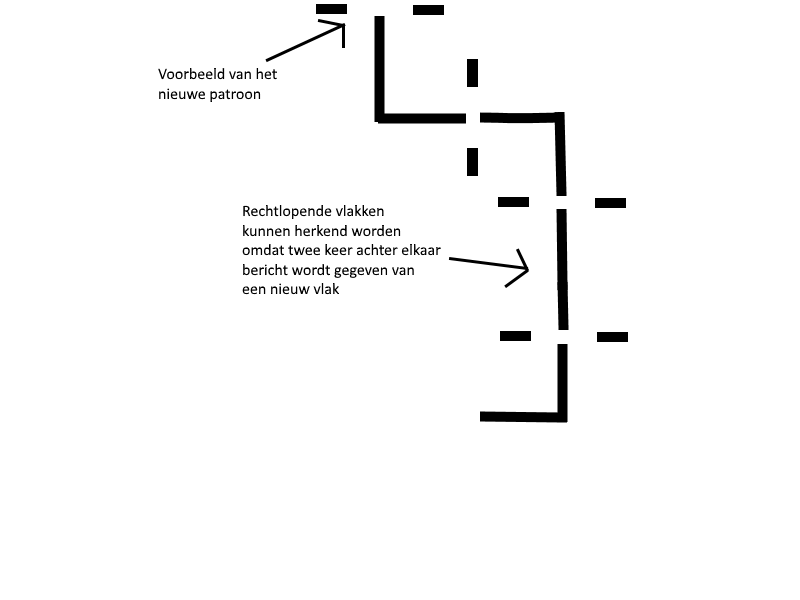
**Figuur 6 Foto van de robot met 6 sensoren**

We hadden vier sensoren beschikbaar om de lijn te volgen. Deze nummerden we van 1 tot 4. Als een sensor op de lijn zat, werd hij actief en gaf hij zijn nummer door. De lijn was dik genoeg zodat maximaal 2 sensoren tegelijk aan konden geven dat ze de lijn zagen. Vervolgens telden we de nummers van de sensoren bij elkaar op en deelden we het resultaat door het aantal sensoren dat actief was. Tenslotte trokken we nog 2,5 van het gedeelde resultaat af. Dit leverde onze Process Variable op wat een getal tussen de -1,5 en de 1,5 was. De Setpoint van onze controller was bereikt als de lijn precies onder het midden van de robot liep. Zodra er een mingetal uitkwam begon de robot naar links te sturen en als er een plusgetal uitkwam naar rechts. Als de PV 0 was betekende dat dat de robot zijn weg niet hoefde aan te passen en de lijn dus precies onder de robot liep. Nadat we de PID Controller geïmplementeerd hadden reed de robot veel stabieler en kwam de gewenste situatie veel vaker voor dan voorheen.

### Informatie verzamelen

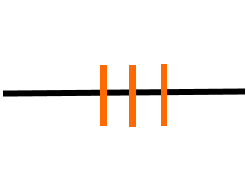
Nu konden we gaan kijken naar het verzamelen en verzenden van informatie. Hierbij konden we dezelfde logica gebruiken die het voorgaande onderzoek ook toepaste. In dat onderzoek werd al onderscheid gemaakt tussen bochten, kruispunten en doodlopende lijnen. Hier moest echter nog aan toegevoegd worden wat een recht vlak was. Om dat op te lossen zijn we eerst gaan kijken naar de mogelijkheid om de robot te laten herkennen wanneer hij een nieuw vlak tegenkwam, want als de robot doorgaf dat hij een nieuw vlak tegengekomen was en dat het bericht dat hij daarna verzond ook aangaf dat hij een nieuw vlak was tegengekomen moest dat betekenen dat hij over een rechte lijn gereden had.

Om een nieuw vlak te kunnen herkennen hadden we een uniek patroon ontwikkeld. Dit patroon was wit op de middelste sensoren en zwart op de buitenste. Als dit patroon gesignaleerd werd, stuurde de robot dat hij een nieuw vlak gevonden had. Tijdens het testen bleek echter dat het moeilijk was om in het programma af te vangen wanneer het patroon herkent moest worden. De robot dacht te snel dat hij een bocht moest maken waardoor het voorkwam dat hij vast kwam te zitten in een loop van bochten maken.



**Figuur 8 Abstract representatie van een doolhof met het nieuwe patroon**

Uiteindelijk hebben we afgezien van het nieuwe patroon en aan het rechte vlak zelf een aanpassing gemaakt. De zwarte lijn van het rechte vlak werd nu in het midden van het vlak afgebroken door drie witte strepen. De robot moest de strepen in een bepaalde tijd tellen en bericht geven van een recht vlak zodra hij alle drie de strepen geteld had. Dit patroon bleek wel effectief en correct bericht te geven en had geen invloed meer op de andere vlakken.



**Figuur 9 Abstracte weergave van een recht vlak**

De berichtgeving zelf hebben we versimpeld. In het voorgaande onderzoek werd de informatie van een heel vlak verzonden naar de luisteraar. Hierbij hadden de luisteraar en de rijdende robot allebei de hele definitie van het vlak nodig. Dit nam veel ruimte in beslag op beide Arduino's. Vanuit de theorie dat de luisteraar aan een laptop verbonden is, hoefde alleen de laptop de volledige beschrijving van een vlak te hebben. We hadden de code van de robot en de luisteraar zo aangepast dat de robot alleen een id naar de luisteraar verzond. De luisteraar ving het bericht met de id op en gaf deze door aan de laptop. Vervolgens vertaalde de laptop wat het id betekende voor de kaart.

### Kaart samenstellen

De ontvangen id's konden verschillende dingen betekenen. De id’s 10 tot en met 13 gaven de oriëntatie van de robot aan en deze veranderde elke keer wanneer hij een draai maakte. Hieruit kon het programma opmaken of de robot een bocht had gemaakt of een doodlopend einde was tegengekomen. Als de robot bijvoorbeeld eerst naar het noorden aan het rijden was en een bocht naar rechts tegenkwam, veranderde hij zijn oriëntatie naar oost. Het id van oost stuurde hij dan naar de luisteraar en die gaf het id vervolgens door aan het programma op de laptop.

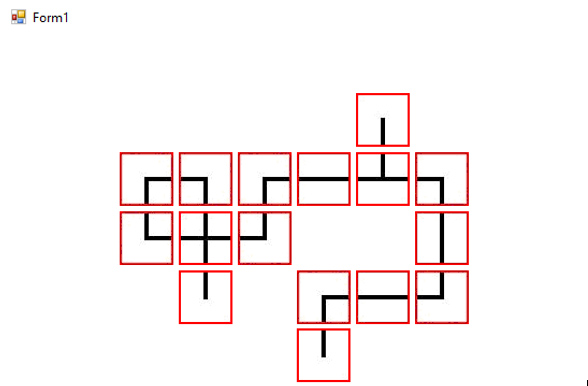
Het c# programma had een winform om visueel de kaart weer te kunnen geven. Het programma bevatte een Serial Input Reader die word aangeroepen telkens als de luisteraar iets naar zijn Serial Output printte. De ontvangen berichten werden daardoor uitgelezen door het programma en de informatie werd in een 2D array van vlakken gezet. Daarna tekende hij de vlakken op het winform. Elk vlak had een eigen plaatje.

**Figuur 10 Afbeeldingen van de vlakken in het programma**

Het programma had drie basis en twee samengestelde vlakken. De basis vlak konden afgeleid worden uit een enkel radiobericht. Een bocht vlak ontstond als de oriëntatie een stap veranderd was, bijvoorbeeld van noord naar oost. Een doodlopende vlak ontstond als de oriëntatie twee stappen veranderd was, bijvoorbeeld van noord naar zuid. Een recht vlak was een apart bericht die los stond van de oriëntatie aangezien de robot zijn oriëntatie op een recht vlak niet veranderde.

De samengestelde vlakken ontstonden met een combinatie van berichten. Dat betekende ook dat de robot meerdere keren over het vlak gereden moest hebben om alle informatie van dat vlak door gestuurd te hebben. Een T-vlak ontstond als de robot twee bochten op hetzelfde vlak had gemaakt of als de robot eerst rechtdoor over het vlak was gegaan en daarna een bocht over het vlak maakte. Een kruispunt ontstond als de robot drie keer een bocht had gemaakt over hetzelfde vlak. Door te volgen op welk vlak de robot het laatst was geweest ontstond zo de kaart van het doolhof.



**Figuur 11 Voorbeeld van het resultaat van een getekend doolhof**

# Resultaten

In de loop van het onderzoek is de beweging van de rijdende robot flink verbeterd. Als hij alleen de lijn hoeft de volgen blijft de lijn onder het midden van de robot. Ook de berichtgeving van de robot is flink opgehelderd. Hij hoeft alleen nog een simpele id naar de luisteraar te verzenden om zijn informatie door te geven. De luisteraar geeft het ontvangen id aan de laptop door en het programma op de laptop regelt het verwerken van de informatie en laat er een visuele representatie van zien.

Bochten maken in het doolhof is echter nog steeds sterk afhankelijk van hoeveelheid stroom die de stroombron van de Arduino levert. Hierdoor komt het nog steeds vaak voor dat de draaiing van de robot te vroeg of te laat afgekapt wordt waardoor hij de lijn kwijtraakt en verkeerde berichten gaat verzenden. Het komt ook nog vrij vaak voor dat de robot de weg kwijt geraakt denkt te zijn als hij de witte strepen van een recht vlak ziet. Tijdens het testen leek het vaker goed te gaan als de robot met genoeg snelheid over de strepen reed. Hieronder de werkelijke testresultaten. Voor het testen maakten we onderscheid tussen de robot met het ontwerp en de code zoals we hem kregen van het vorige onderzoek en de robot met de nieuwe code van dit onderzoek.

|  |  |  |
| --- | --- | --- |
|  | Gemiddelde tijd(s) | Gemiddeld aantal fouten |
| Code voorgaand onderzoek | 34,47 | 33 |
| Nieuwe code met 6 sensoren en PID Controller | 14,30 | 7 |

**Figuur 12 Testresultaten van het testen met het testdoolhof**

# Reflectie

De start van dit onderzoek verliep vrij traag en moeizaam. De code van het voorgaande onderzoek leek op een aantal punten logisch opgebouwd maar aangezien het moeilijk was om de situatie te creëren waarbij van de luisteraar en de rijdende robot werkelijk met elkaar communiceerden en samenwerkten, moesten we eerst een groot deel opnieuw schrijven om de boel aan de praat te krijgen. Dit leidde ertoe dat we niet meteen konden gaan werken aan het doel van ons onderzoek.

Nadat we eerst drie weken hebben gewerkt aan het aanpassen van de rijdende robot en het implementeren van de PID Controller, konden we pas echt gaan kijken naar het verzamelen van informatie van het doolhof. We dachten hier zo'n 4 weken mee bezig te zijn, maar tijdens het ontwikkelen kwamen er een hoop problemen naar boven die het afmaken van dit onderdeel flink vertraagden. De inconsistente snelheid van de motoren zorgde ervoor dat als we ook maar kleine aanpassingen maakte aan de code dat de robot opeens veel meer fouten kon gaan maken. Door heel het project heen hebben we hier nooit een echte oplossing voor kunnen vinden om af te vangen dat de robot soms veel trager reed. Het kwam ook voor dat een van de motoren zo weinig stroom kreeg dat hij bleef hangen, waardoor de robot helemaal in de war raakte.

We hadden er waarschijnlijk goed aan gedaan om een accu te kopen voor de robot om stroom naar en dus ook de snelheid van de robot consistenter te kunnen maken, maar aangezien dat allemaal op eigen kosten had gemoeten hadden we er voor gekozen om oplaadbare batterijen te gebruiken.

# Aanbevelingen

Voor toekomstige onderzoeken kunnen we sterk aanbevelen om nogmaals naar een goede stroombron te kijken. Aangezien de stroombron in ons onderzoek zoveel invloed had op de resultaten die we kregen, geloven we dat er nog terrein op dat gebied gewonnen kan worden.

Daarnaast kan ook de plaatsing van de sensoren nog verbeterd worden. In ons onderzoek waren ze met een aantal bouten aan het robot frame bevestigd. Deze raakten af en toe een beetje los waardoor de sensoren te dicht tegen de grond aan zaten. Hierdoor bleven ze haken achter de vlakken en de tape die we gebruikten.

# BIJLAGEN

## Bijlage 1 Literatuurlijst

[1] Maniacbug, Arduino driver for nRF24L01. Van,  
https://github.com/maniacbug/RF24

[2] iPrototype.nl, Line Tracking sensor – Analog. Van, https://iprototype.nl/products/components/sensors/linetracking-analog

[3] PID Line Following Robot. Van,  
http://waihung.net/pid-line-following-robot/

[4] PID\_controller. Van,  
https://en.wikipedia.org/wiki/PID\_controller

[5] PoluluQTRSensors. Van,  
https://github.com/pololu/libpololu-avr/blob/master/src/PololuQTRSensors/PololuQTRSensors.cpp

[6] PID Based Line Follower. Van,  
 http://letsmakerobots.com/node/38550

[7] Arduino/Genuino UNO. Van,  
https://www.arduino.cc/en/Main/ArduinoBoardUno

[8] PID Theory Explained. Van,  
http://www.ni.com/white-paper/3782/en/