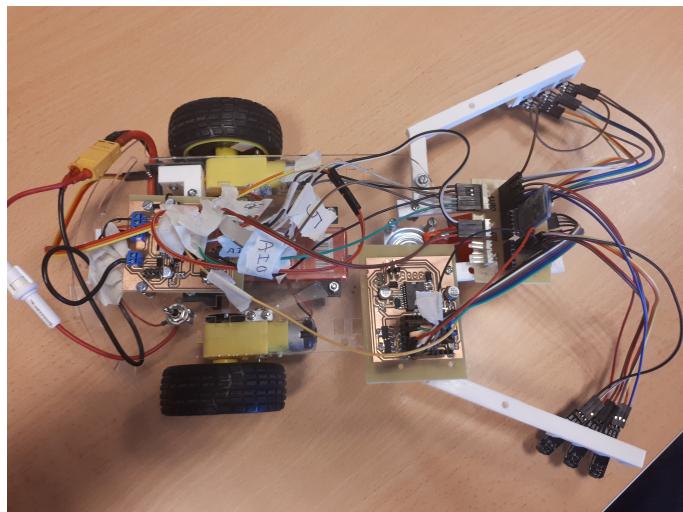


Line-following Robot



**Tom LIERMAN
Matthias ALLEMAN**

Begeleiders:

Guus Leenders
Stijn Crul
Carine Naessens
Liesbet Van der Perre

Bachelorproef ingediend tot het behalen van
de graad van Bachelor of Science in de
industriële wetenschappen: Elektronica-ICT

Coach:

Kevin Verniers

©Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Technologiecampus Gent, Gebroeders De Smetstraat 1, B-9000 Gent, +32 92 65 86 10 of via e-mail iiw.gent@kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Inhoudsopgave

1	Opdrachtbeschrijving	1
1.1	Hardware	1
1.2	Software	2
2	Onkosten	3
3	Vormgeving van de Robot	4
4	Hardware	7
4.1	Tussenstuk	7
4.2	Ardumoto	8
4.2.1	Ardumoto RFID	8
4.2.2	Ardumoto Motor	9
5	Software	11
5.1	Arduino	11
5.1.1	Motor	12
5.1.2	PD-algoritme	12
5.1.3	Detectie-algoritme	13
5.1.4	Communicatie	14
5.2	Raspberry Pi	15
6	Moeilijkheden	17
6.1	Software	17
6.2	Hardware	19
7	Coach	20
8	Besluit	21

Lijst van figuren

1.1	Motorshield gecombineerd met een Arduino.	2
3.1	Arm voor 3 sensoren.	5
3.2	Arm voor sensoren die we gebruiken in ons eindontwerp	5
4.1	Routing van de PCB die als verlengstuk dient.	8
4.2	De gebruikte Bluetooth-module, ingeplugged op de verlengPCB.	8
4.3	Gebruikte RFID-reader MRFC522.	9
5.1	Flowchart van de werking van de libraries.	11

Lijst van tabellen

2.1 Tabel met aankopen en hun prijs.	3
4.1 Tabel met mogelijkheden afmetingen koperbanen.	10

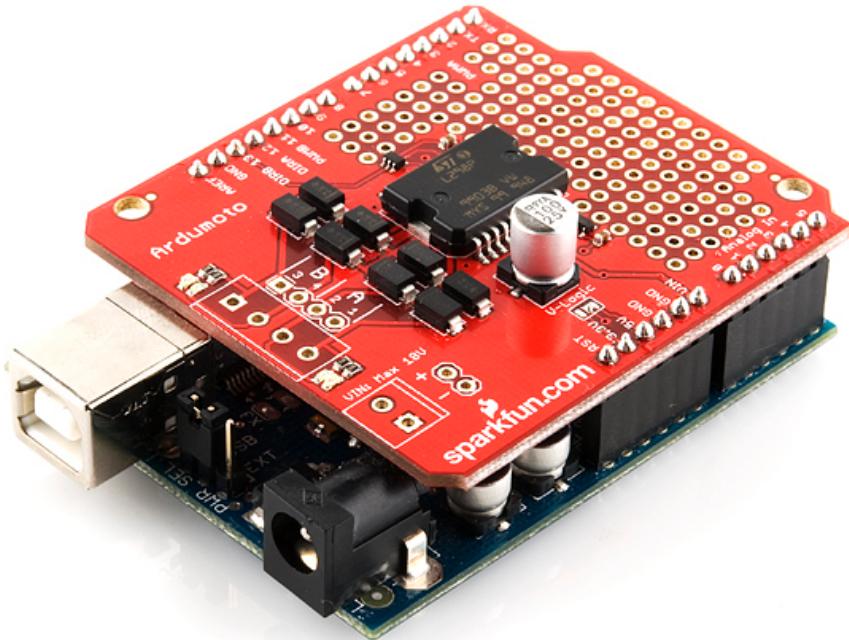
Hoofdstuk 1

Opdrachtbeschrijving

De opdracht bestaat erin om een line-following robot te maken. Deze robot wijkt enigszins af van de klassieke line-follower aangezien de parcours die wij moeten kunnen afleggen, bestaan uit 2 volle buitenlijnen en een gestreepte middellijn, waardoor we niet zo maar de middellijn kunnen volgen. De onderbrekingen maken een algoritme tot het volgen ervan zeer ingewikkeld en onvoorspelbaar. We kregen op voorhand 3 verschillende circuits die de robot autonoom zou moeten afleggen terwijl hij simultaan andere taken uitvoert zoals snelheden registreren en doorsturen of RFID-tags uitlezen en doorsturen. De breedte van de baan is ongeveer twee keer de breedte van de auto. We kregen een basispakket vanwaar we konden vertrekken dewelke bestond uit het chassis van de robot, 2 motoren, een batterij, een Sparkfun motor shield en 2 Arduino Uno's. We hadden ook steeds een 3D-printer ter beschikking om 3D-onderdelen de printen die we nodig hadden om sensoren te bevestigen en dergelijke. Verder kregen we ook nog een budget van 50 euro van de school om ons te voorzien van de nodige componenten. De opdracht bestaat zowel uit het bedenken en ontwerpen van de nodige hardware als het schrijven van de nodige software zodat de robot de 3 verschillende circuits autonoom zou kunnen afleggen.

1.1 Hardware

Voor het ontwikkelen van de Software konden we beroep doen op een Arduino Uno en een Motorshield van Sparkfun, zoals te zien in Figuur 1.1, maar voor de uiteindelijke opdracht moesten we natuurlijk gebruik maken van printplaten die we zelf vervaardigd hadden. Aan de hand van een schema van de Arduino Uno, dat we konden downloaden van het internet, konden we onze eigen PCB ontwerpen waarbij we alle niet-noodzakelijke componenten weglieten en de L298 Chip toevoegden voor de aansturing van de motoren. Voor de sensoren, Bluetooth-module en RFID-reader waren we vrij om te kiezen welke we kochten of deze zelf maakten.



Figuur 1.1: Motorshield gecombineerd met een Arduino.

1.2 Software

Het tweede onderdeel van de opdracht bestaat ervan om de Arduino (en dus later ook onze eigen PCB) dusdanig te programmeren dat hij autonoom verschillende circuits kan afleggen in een zo kort mogelijke tijd. Er werd ons aangeraden te werken met een PID-regeling om een zo stabiel mogelijke robot te verkrijgen. Tijdens het afleggen van het circuit, moet de robot in staat zijn om zijn snelheid te meten, RFID-tags uit te lezen en van beide, de data , via Bluetooth, door te sturen naar een Raspberry Pi.

Hoofdstuk 2

Onkosten

In dit hoofdstuk maken we een oplijsting van de kosten die we gemaakt hebben voor ons project. Alle basiscomponenten die we nodig hadden voor onze zelfgemaakte PCB met motorsturing en Arduino werden sowieso al voorzien voor ons en moesten we dus niet opnemen in deze onkosten tabel. In Tabel 2.1 vermelden we enkel de componenten die we extra aangekocht hebben voor ons project. Zoals we reeds vermeld hebben in hoofdstuk 1 konden we ook vrij gebruik maken van een 3D-printer waarvan we deze kosten niet in rekening brengen. We hebben dus slechts drie verschillende zaken aangekocht, namelijk een Bluetooth-module, een RFID-module en 10 line-following sensors, zoals u in onderstaande tabel kan zien.

Tabel 2.1: Tabel met aankopen en hun prijs.

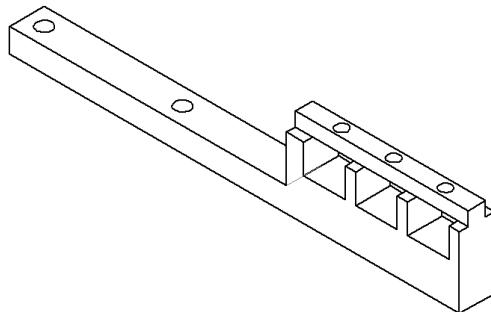
Beschrijving aankoop	Prijs (in euro)
HCO5 (Bluetooth module)	5.37
MRFC522 (RFID Reader)	4.15
10x Line-following Sensors	2.16
Totaal:	11.68

Hoofdstuk 3

Vormgeving van de Robot

Alvorens aan te vangen met het schrijven van de software en het ontwerpen van de PCB hebben we onderzocht hoe de ideale robot er zou moeten uitzien om het parcours foutloos en zo snel mogelijk te kunnen afleggen. Een eerste keuze die we moesten maken was of we het losse wieltje vooraan of achteraan wilden plaatsen. We bekeken beide situaties en besloten dat het losse wieltje vooraan de beste oplossing was aangezien de sensoren zich dan op een grotere afstand bevinden van de stuurwielen. De afstand van sensor tot de stuurwielen zorgt ervoor dat de kleinste fout reeds tot uiting komt en de wielen dus sneller correcties kunnen uitvoeren zodat de fout minimaal blijft. Ook bij de keuze van de geschikte arm zal blijken dat we de afstand tussen de stuurwielen en de sensoren trachten te maximaliseren.

We hadden als oorspronkelijk doel om onze robot autonoom de circuits te laten rijden met behulp van 1 arm van 13 cm waarop 3 sensoren bevestigd zijn zoals te zien in Figuur 3.1. We hebben dan ook zo snel mogelijk deze arm ontworpen en geprint met de 3D-printer zodat wij onmiddellijk konden beginnen met het ontwerpen van de software en het afstellen van de motoren en de sensoren met behulp van PID waarden. Nadat we deze arm aan de rechterkant hadden bevestigd, ging het allemaal vrij snel en hadden we in een mum van tijd een robot die het eerste circuit, het ovale, vloeiend kon afleggen zonder van het parcours te komen. De arm stond 90° gedraaid ten opzichte van de langsas van onze Robot waardoor we de afstand tussen de stuurwielen en de sensoren niet konden maximaliseren. Nadat we de arm onder een hoek van ongeveer 30° ten opzichte van de langsas van de robot geplaatst hadden, merkten we een grote prestatieverbetering op en konden we de snelheid opdrijven.

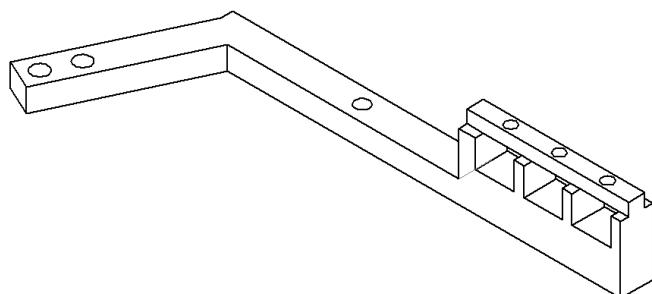


Figuur 3.1: Arm voor 3 sensoren.

Ons ontwerp was momenteel enkel nog maar in staat om bochten te nemen in 1 richting, namelijk tegenwijzerzin, wanneer we de buitenbochten volgden. We begrepen dat ons ontwerp nog niet voldeed om de ingewikkeldere circuits af te leggen waarbij er zowel bochten naar links als naar rechts zouden voorkomen. We hebben vervolgens onderzoek gedaan om de optimale arm te ontwerpen.

De tweede arm die we ontwikkelden kon plaats bieden aan 5 sensoren en moest onder eenzelfde hoek geplaatst worden als de bovenstaande arm. We ondervonden echter dat de 2 extra sensoren geen meerwaarde konden bieden aangezien we geen extra foutsituaties konden definiëren in onze software omdat de arm niet buiten het circuit mocht komen.

Ons laatste ontwerp bestond uit 2 armen, die elkaar spiegelbeeld zijn en waarbij de top van de arm opnieuw een hoek van 30° maakt met de langssas van de robot. U kan dergelijke arm vinden in Figuur 3.2. Ons doel was dat de armen van de robot zich zo dicht mogelijk bij de buitenlijnen van het circuit bevonden zodat, wanneer de robot afwijkt van de lijn, dit niet zou resulteren in een te grote afwijking van zijn richting. Daarom bezit de arm eerst nog een stuk van 5 cm in de dwarsrichting van de robot zodat de ene arm zich maximaal 2 cm van de buitenlijn bevindt, wanneer de andere zich op de lijn bevindt. Deze armen bleken voor ons project optimaal en gebruikten we dan ook in ons eindontwerp.



Figuur 3.2: Arm voor sensoren die we gebruiken in ons eindontwerp

We ontwierpen verder ook nog een houdertje voor de RFID-reader zodat deze zich net voor het losse wieltje zou bevinden. Dit is de plaats waar de robot, en dus ook de RFID-tag, zich met grootste zekerheid boven de lijn bevindt.

Om de snelheid van onze robot te meten, maakten we gebruik van eenzelfde infrarood sensor als de sensoren voor het registreren van de witte lijn. We printten een zwart schijfje met de 3D-printer en maakten een spie van de schijf wit zodat de sensor het wit kon waarnemen en op die manier telkens een rotatie van het wiel kon registreren. De sensor zelf hebben we subtiel met een klein houdertje bevestigd onder de robot.

Als laatste gaven we de robot nog vorm met 2 verlengstukjes om een extra printplaat, bedoeld om de bedrading overzichterlijker te maken, te bevestigen en eenhouder voor de batterij, zodat we deze onderaan konden bevestigen en deze niet meer in de weg zou liggen.

Voor de vormgeving hebben we dus rijkelijk gebruik gemaakt van de 3D-printer waardoor we alles konden positioneren op de exacte plaats waar we het wilden waardoor de robot een afgewerkt geheel werd.

Hoofdstuk 4

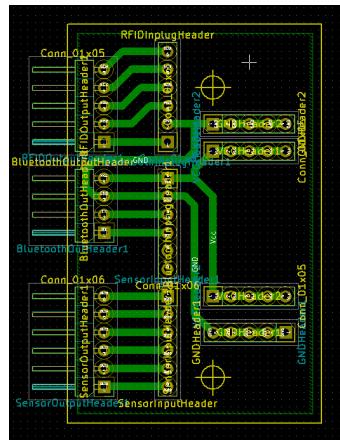
Hardware

Onze robot werkt met behulp van 3 zelfgemaakte PCB's. Twee ervan zijn de combinatie van de Motorshield en de Arduino/Atmega en de andere is een printplaat om de bekabeling te reduceren.

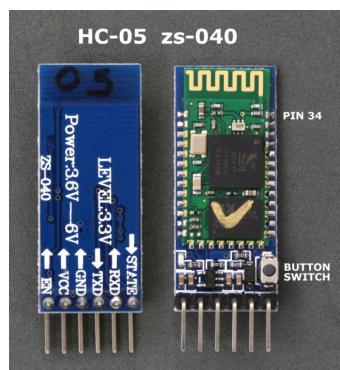
4.1 Tussenstuk

Het tussenstuk is een PCB waarop alle lijnen van de infra rood sensoren toekomen en worden doorgegeven aan de AtMega. Het voordeel van deze printplaat is dat we nu slechts 1 GND en 1 VCC lijn moeten voorzien tussen het printplaatje en de Arduino. We hebben op dit printplaatte ook de mogelijkheid voorzien om de Bluetooth-module en de RFID-reader aan te sluiten. Nadien hebben we ontdekt dat de RFID-reader niet kon worden aangesloten op onze hoofd Arduino waardoor we de RFID-reader ook niet verbonden met dit tussenstuk maar wel rechtstreeks aan de hulp-Arduino. Verbinden met het tussenstuk zou de vereenvoudiging van de bekabeling teniet doen. Het probleem omtrent de RFID-reader leggen we later nog uitgebreider uit.

We hebben dus 10 5V-aansluitingen voorzien, 10 GND-aansluitingen, 6 aansluitingen voor de outputsignalen van de sensoren en de pinnen die nodig zijn voor de Bluetooth-module (Key, Vcc, GND, TXD, RXD en State). Niet alle zes de uitgangen van de Bluetooth-module moeten worden doorgestuurd naar onze Atmega, deze vereenvoudiging vindt ook plaats op deze PCB. Enkel de signalen Vcc, GND, TXD en RXD worden van een uitgang voorzien. Deze uitgangen worden dus doorverbonden met de Atmega. De Vcc en de GND aan deze uitgang zorgen dus ook voor de voeding van de volledige PCB. We maken gebruik van de HC-05 Bluetooth-module, een foto van de module wordt weergegeven in afbeelding 4.2.



Figuur 4.1: Routing van de PCB die als verlengstuk dient.



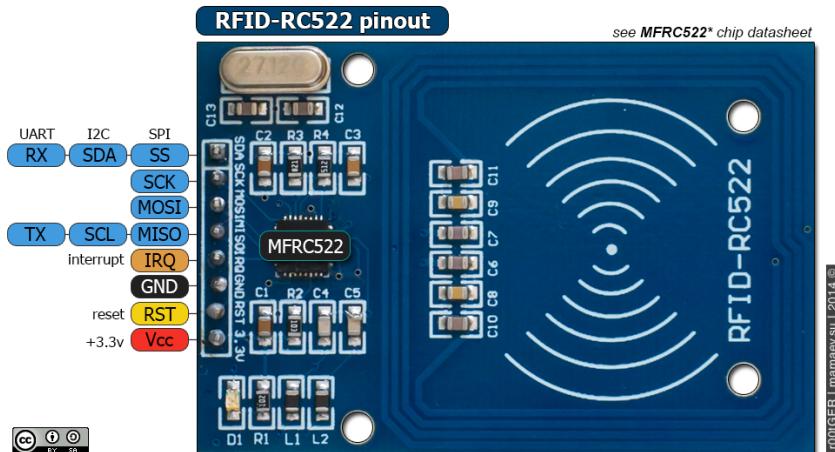
Figuur 4.2: De gebruikte Bluetooth-module, ingeplugged op de verlengPCB.

4.2 Ardumoto

4.2.1 Ardumoto RFID

De tweede printplaat die we ontworpen hebben, is eigenlijk de combinatie van de Atmega en de Motorshield, hoewel we hier enkel gebruik zullen maken van de Atmega en de L298 zullen laten voor wat het is. Deze PCB zorgt voor de aansturing en uitlezing van de RFID-reader. We hebben een tweede Atmega nodig omdat de RFID-reader volgende signalen nodig heeft om correct aangestuurd te worden: SS/RX, SCK, MOSI, MISO/TX, IRQ, GND, RST, Vcc. 3 van deze pinnen worden ook gebruikt voor de motoraansturing. SCK is namelijk dezelfde pin als die voor de aansturing van de richting van motor B. MOSI is dezelfde pin als die voor de aansturing van de snelheid van motor B. En de laatste overeenkomstige pin is die van MISO, dit is namelijk dezelfde pin als de aansturing van de richting van motor A. Deze pinnen kunnen geen twee signalen gelijktijdig verwerken waardoor we dus opteerden om gebruik te maken van de hulp-Arduino. Een andere mogelijkheid om dit probleem op te lossen kon erin bestaan om gebruik te maken van een RFID-reader die het I^2C principe hanteert, in plaats van ISP. Door I^2C te gebruiken zouden we geen conflicten met de pinnen gehad hebben en zouden we dus ook geen tweede PCB nodig gehad hebben. We maken gebruik van de RFID-reader-module,

namelijk de MRFC522, dewelke wordt weergegeven in figuur 4.3. Om een RFID-tag zo goed mogelijk uit te lezen moet de reader op ≈ 1 cm boven de tag passeren. We hebben hiervoor zelf een RFID-houdertje geprint met de 3D-printer zodat deze afstand ten alle tijden constant blijft.



Figuur 4.3: Gebruikte RFID-reader MRFC522.

4.2.2 Ardumoto Motor

Deze PCB is verantwoordelijk voor de aansturing van de motoren en ook voor het doorsturen van de signalen naar de Raspberry Pi via de HC-05. Na het finaliseren van de PCB bleken er nog enkele fouten te zijn gesloten in onze versie. Enkele footprints waren aan de kleine kant waardoor sommige componenten slechts heel nipt op de koper-eilandjes pasten. Na ons prototype hadden we deze fout aangepast maar bleek onze correctie nog onvoldoende te zijn om comfortabel te kunnen solderen. Ook een condensator bleek bij het testen niet goed vastgesoldeerd te zijn waardoor we slecht contact kregen wat we opgelost hebben met een externe verbinding. Maar de grootste fout waarop we gestoten zijn, was dat een ingang van de L298 (de IC verantwoordelijk voor de motoraansturing) niet verbonden was met het PWRIN signaal waardoor er geen vermogen naar de motoren gestuurd werd. Dit kwam omdat er een hiérarchical label ontbrak in ons schema op de betreffende locatie aan de L298. We hebben dit probleem kunnen oplossen door een extra kabel te solderen van het PWRIN signaal naar de betreffende pin van de L298.

Als vertrekpunt van ons ontwerp zijn we vertrokken van het schema van een Arduino Uno dat we op het internet konden downloaden. Als eerste konden we de USB interface voor de aansluiting van een USB-kabel weglaten aangezien we dit in ons finaal ontwerp niet nodig hadden. Voor het uploaden van een programma maken we dan gebruik van de ICSP-pinnen die we nauwkeurig verbinden met een andere Arduino die de code doorstuurt. Alle LED's die normaal gezien aanwezig zijn op de Arduino Uno hebben we weggelegd omdat we dit niet echt nodig hebben en dit onze arduino groter zou maken dan nodig. Om het ons eenvoudiger te maken, pasten we in het schema ook de namen van de labels aan naar meer betekenisvollere namen zodat we de namen van de lijnen zo eenvoudiger konden interpreteren. Zo veranderden we bijvoorbeeld MISO door DIR A aangezien deze lijn verantwoordelijk was voor de richting van Motor A. (FOTO ATMEGA) R8 hebben we ook weggelegd aangezien dit een weerstands-

waarde van 0Ω heeft. Deze weerstand wordt voornamelijk gebruikt om een brugje te creëren om er baantjes onder te kunnen trekken. Pinnen 27 en 28 hebben we niet nodig, dus mogen we de condensator hier rond ook verwijderen. De jumper hebben we weggeleggen omdat we die niet nodig achten. Eenmaal we het schema hadden die we nodig hadden was het tijd om te routen. We zijn er in geslaagd om een PCB te maken met afmetingen ($52.324\text{mm} \times 61.468\text{mm}$). We konden dus ook strikt genomen onze printplaat deze afmetingen geven maar we hebben er bewust voor gekozen om de printplaat wat groter te maken, nl. $53.34\text{mm} \times 73.152\text{mm}$. We hebben hiervoor geopteerd omdat we dan bevestigingsgaten konden voorzien die passen in de voorziene gaten op de auto. We hebben begonnen met baantjes van 10 mills($= 0.254\text{mm}$). En daarna hebben we zoveel mogelijk de baantjes verbreed tot maximaal 30 mills. Op de plaatsen waar het kon hebben we daarvoor ook de baantjes verlegd, zodat we ze breder konden maken. We hebben de via's ook op een grotere maat gezet, alle afmetingen worden grafisch weergegeven in tabel 4.1.

Tabel 4.1: Tabel met mogelijkheden afmetingen koperbanen.

	Clearance	Track Width	Via Dia	Via Drill
Default	0.25	0.25	0.6	0.4
0.5	0.25	0.5	0.7	0.5
12 mills	0.25	0.3	0.6	0.4
15 mills	0.25	0.38	0.6	0.4
30 mills	0.25	1	0.6	0.4

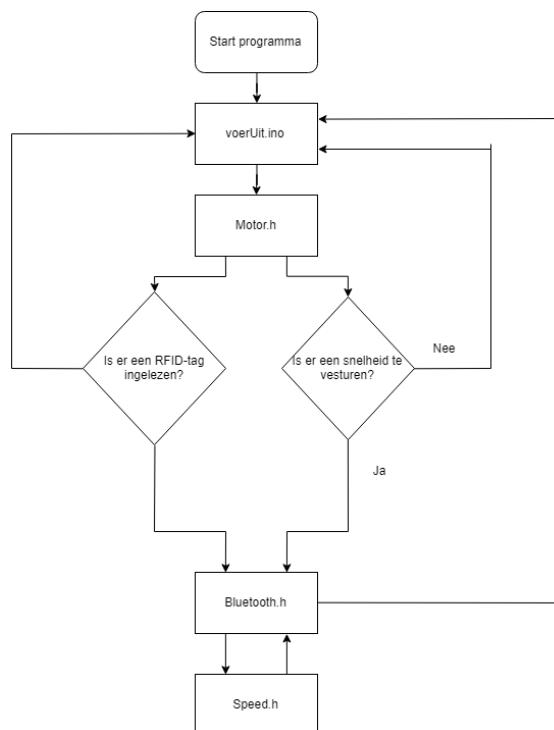
We hebben ook de 12 mills erbij gezet omdat deze dikte van de baantjes dan exact gelijk is aan de breedte van de bevestigingspinnen van een SMD-component. Enkele baantjes zijn ook ingesteld op 15 mills. We hebben natuurlijk ook geprobeerd het aantal via's te beperken, wat ons wel redelijk goed gelukt is, we hebben er slechts 20 gebruikt. We merkten dat bij onze eerste pogingen om de printplaten te drukken de fijnste baantjes vaak volledig opgelost waren door te lang in het zuurbad te liggen. We kregen de tip om gebruik te maken van een opgevuld vlak, met als reden dat we nu veel minder lang de PCB in het zuurbad moesten steken want er moest veel minder koper opgelost worden. Achteraf gezien konden we eigenlijk aan de beide vlakken een functie geven bv. GND-vlak, wij hebben dit geen betekenis gegeven. Eens de PCB volledig bestukt was, wilden we het eerst testen met een standaardprogramma voor de atmega nl. Getting To Blinky. Dit werkte direct en daarom veronderstelden we verkeerdelyk ook direct dat de volledige PCB werkzaam was. We bespreken dit uitgebreid in het hoofdstuk ??.

Hoofdstuk 5

Software

5.1 Arduino

In onze code hebben we er voor geopteerd om zo veel mogelijk gebruik te maken van zelfgemaakte bibliotheken om zo de onderdelen van ons programma zo goed mogelijk van elkaar te scheiden zodat er zo weinig mogelijk verwarring mogelijk is bij het bestuderen van de code. We gebruiken dan ook slechts 1 lijn code in ons .ino bestand om een bibliotheek aan te roepen van waaruit vervolgens het hele programma wordt aangestuurd. In Figuur 5.1 vindt u de flowchart van ons programma.



Figuur 5.1: Flowchart van de werking van de libraries.

5.1.1 Motor

We zijn eerst en vooral begonnen met het schrijven van de Motor-bibliotheek. Deze bibliotheek wordt aangeroepen vanuit de loop van het main programma genaamd voerUit.ino. We registreren, elke keer dat de loop wordt uitgevoerd, de digitale waarden aan de sensoren en plaatsen deze voortdurend in de array van integers, genaamd LFS[], die vervolgens eenvoudig gebruikt kan worden voor de verwerking en generatie van de effectieve correctiewaarden. Ik stel dergelijke array voor als volgt: 000 001 waarbij nu enkel de meest rechtse infrarood sensor wit detecteert en alle 5 de andere sensoren zwart detecteren. We hebben het programma opgebouwd volgens het principe waarbij we aan de verschillende combinaties van sensoren die wit detecteren, verschillende foutwaarden associëren zodat er in principe geen enkele rusttoestand is. Er wordt als het ware voortdurend gecorrigeerd maar in zo een kleine mate dat dit amper zichtbaar is voor ons. In ons programma krijgt de kleinste fout, namelijk de buitenste sensor die wit detecteert, de foutwaarde 1.1. In het allerslechtste geval, namelijk 000 100, waarbij een van de middelste sensoren wit detecteert, krijgt deze situatie een foutwaarde van 4.6 mee. Deze error values worden elke keer dat het programma doorlopen wordt, aangepast indien de toestand van één van de sensoren gewijzigd werd. Deze waarden zijn niet zomaar gekozen maar we hebben uitgezocht welke combinatie van PD-constanten en error values tot de beste prestaties kon leiden. Een voorbeeld van dergelijke code vindt u in Listing 5.1. De error value die gegenereerd werd, wordt vervolgens gebruikt in de corrigeer-methode waarin de werking van ons PD-algoritme in verwerkt zit.

```
if ((LFS[3]== 1 )||(LFS[4]== 1 )||(LFS[5]== 1 )){  
    rechtersensoractief=true;  
    if ((LFS[3]== 1 )&&(LFS[4]== 0 )&&(LFS[5]== 0 )) { error = 4.6; limitReached=true; }  
    else if ((LFS[3]== 1 )&&(LFS[4]== 1 )&&(LFS[5]== 0 )) { error = 3.2; limitReached=false; }  
    else if ((LFS[3]== 1 )&&(LFS[4]== 1 )&&(LFS[5]== 1 )) { error = 2.8; limitReached=false; }  
    else if ((LFS[3]== 0 )&&(LFS[4]== 1 )&&(LFS[5]== 1 )) { error = 2.3; limitReached=false; }  
    else if ((LFS[3]== 0 )&&(LFS[4]== 0 )&&(LFS[5]== 1 )) { error = 1.1; limitReached=false; }  
}
```

Listing 5.1: Het algoritme die de foutwaarden toekent aan de verschillende situaties

5.1.2 PD-algoritme

Dit algoritme is vrij beknopt maar toch zeer interessant aangezien het de prestaties van het wagentje heel erg verbetert. PID is de afkorting voor Proportioneel, Integrerend en Differentiërend. Deze regelaar is dus in staat om verschillende soorten fouten te detecteren en in de juiste mate te reageren om zo oscillaties weg te werken en een stabiel voertuig te verkrijgen. Het proportioneel aandeel in de regelaar is een vermenigvuldiging van KP met het verschil van de waarde die je zou willen dat de sensoren meten, verminderd met de waarde die effectief geregistreerd wordt. De integrerende actie zorgt voor een foutcorrectie die ontstaat door het constant optellen van de fout. Hoe langer een fout zich dus voordoet, des te groter de waarde van de integrerende term zal zijn. Ook deze waarde wordt nog vermenigvuldigd met een constante namelijk KI. Als laatste hebben we nog de differentiërende term. Deze term is functie van de snelheid van verandering van de fout. Wanneer de afwijking tussen de gewenste waarde en de gemeten waarde toeneemt, zal de D-term in functie van de snelheid van verandering reageren om zo een mogelijks zeer groot wordende fout te beperken in grootte. Ook in de andere richting zal de correctie afgeremd worden wanneer de gemeten waarde nadert naar de gewenste waarde.

De P-waarde wordt gelijkgesteld aan de error value die we hierboven geïntroduceerd hebben. De

I-waarde hebben we in dit project niet gebruikt aangezien deze waarde enorm snel toeneemt en niet eenvoudig was om perfect in te stellen. Achteraf gezien bleek deze term niet noodzakelijk om een stabiel rijdende robot te verkrijgen. We maken dus gebruik van een PD-regelaar niet van een PID-regelaar. De D-waarde wordt slechts om de 500 milliseconden vervangen door een nieuwe waarde. Mochten we deze tijdsrestrictie niet verwerkt hebben in de code, dan zou de D-waarde amper invloed hebben aangezien ons programma meer dan 1000 keer per seconde wordt uitgevoerd en dus met andere woorden, wanneer de fout gedetecteerd wordt en de D-waarde zijn correctie wilt uitvoeren, er onmiddellijk een nieuwe waarde wordt geregistreerd die meestal opnieuw diezelfde waarde is waardoor de D-correctiewaarde wegvalt. We testten de snelheid van ons programma uit met behulp van onderstaande code om tot dergelijk inzicht te komen en verkregen een uitvoer in de Seriele monitor van 0 milliseconden wat dus illustreert dat het programma meer dan 1000 keer per seconde wordt uitgevoerd.

```
void loop() {
    previousTijd=nutijd;
    utijd=millis();
    duration=nutijd-previousTijd;

    if (i==50){
        Serial.println(duration);
    }
    i++;
    motor.bepaalPID();
}
```

Listing 5.2: Het algoritme om de frequentie van de uitvoering van het programma te bepalen

De D-waarde en de P-waarde worden vervolgens nog vermenigvuldigd met de KP en KD constanten, zoals te zien in Listing 5.3, die we met behulp van beproeving hebben bepaald om de beste prestaties te verkrijgen. De som van deze producten wordt opgeslagen in PIDvalue en wordt meegegeven in de rij-methode. Wat niets anders is dan een methode voor het instellen van de algemene snelheid en de snelheid per wiel, rekening houdend met de PIDvalue die we net definieerden.

```
Kp=12;
Ki=0;
Kd=4;
P = error;

stopTijd=millis();
if((stopTijd-startTijd)>500){
    if((error-previousError)>0){
        //I = I + error;
        D = error-previousError;
        previousError = error;
        startTijd=millis();
    }
}
PIDvalue = (Kp*P) + (Ki*I) + (Kd*D);

rij(PIDvalue);
```

Listing 5.3: Het PD-algoritme dat wij hanteren

5.1.3 Detectie-algoritme

Een extra functionaliteit die we verwerkt hebben in onze code die van belang is bij de error value, is het feit dat als alle sensoren zwart detecteren dat de robot weet of hij zich nu binnen of buiten het parcours bevindt waardoor zijn error values verschillen. Bij alle verschillende situaties waarin de sensoren zich kunnen bevinden, wordt er een boolean aangepast. Zo zal de boolean limitReached in alle gevallen false zijn behalve in de 2 gevallen 000 100 en 001

000, dan zal de boolean true gemaakt worden. Dit zijn de situaties waarbij de robot over de buitenlijn aan het gaan is en de ene arm zich buiten het parcours bevindt en de andere arm zich binnen bevindt. Indien de fout nog een klein beetje verder toeneemt krijg je de situatie 000 000 waarbij de witte lijn zich tussen de sensoren bevindt. De robot zal nu zeer hevig gaan corrigeren aangezien de boolean limitReached true is.

Wanneer we werken met 2 armen ontstaan er extra problemen, namelijk naar welke arm moet er geluisterd worden en in welke richting moet de robot uitwijken wanneer alle sensoren zwart zijn. Elke keer dat er een arm wit registreert, wordt de boolean rechtersensorActief aangepast naar true of false naargelang de arm die het wit detecteert. Wanneer de sensoren volgende situatie registreren 000 001, dan wordt de boolean rechtersensorActief op true gezet. Wanneer je in dit geval los komt van de lijn, weet het programma dat het naar rechts moet gaan corrigeren om de rechterlijn terug te intercepten. Maar wanneer de boolean limitReached op true staat, wilt dat zeggen dat de situatie 000 100 zich net voordien heeft voorgedaan en het wagentje zich dus eigenlijk te ver rechts van het circuit bevindt. Nu zal het wagentje naar links moeten corrigeren met een grotere factor dan normaal aangezien de boolean limitReached deze hevige reactie genereert.

De 2e functionaliteit dat we hebben toegevoegd is het principe dat de robot weet of hij zich links of rechts van de lijn bevindt en zo dus in de juiste richting kan corrigeren. Dit was nodig om de 2 armen te kunnen laten samenwerken.

5.1.4 Communicatie

In de methode bepaalPID() bevinden er zich nog 2 andere methodes die aangeroepen worden, namelijk verstuurSnelheid() en verstuurTag(). Beide methodes bevatten een methode die de communicatie met de Bluetooth-bibliotheek afhandelt. In verstuurSnelheid() bevindt zich de methode bluetooth.stuurSnelheid(). In de Bluetooth-bibliotheek wordt er vervolgens een waarde opgevraagd van de Speed-bibliotheek waarin zich het algoritme voor de registratie van de huidige snelheid van de robot bevindt die je vindt in Listing 5.4. De snelheidswaarde wordt telkens wanneer het wiel een rotatie maakt aangepast met behulp van een eenvoudige snelheidsvergelijking. De snelheid wordt vervolgens 1 keer per seconde gereturned naar de Bluetooth-bibliotheek van waar het met behulp van Software Serial verstuurd wordt met de HC-05 naar de Raspberry Pi.

```
void Bluetooth::stuurSnelheid(){
    snelheidswaarde=speed.getSpeed();
    if (millis()%1000==0){
        BTSerial.println(snelheidswaarde);
    }
}

double Speed::getSpeed(){
    registreerPuls();
    return v;
}

void Speed::registreerPuls(){
    if(digitalRead(sensor)==1){
        if(!actief){
            getDuration();
            actief=true;
        }
    } else {
        actief=false;
    }
}

void Speed::getDuration(){
    huidigeTijd=millis();
    tijdsduur=huidigeTijd-vorigeTijd;
    vorigeTijd=huidigeTijd;
```

```

        v=0.235/(tijdsduur/1000);
    }
}

```

Listing 5.4: Methode om de snelheid te registreren

De verstuurTag()-methode is een methode die instaat voor het correct versturen van de RFID-ID van de tags die zich onder de mat bevinden. We stootten daarbij op het probleem dat de mfrc522 gebruikt maakt van de SPI-pinnen. Deze pinnen waren echter reeds in gebruik door de motoraansturing die verwerkt is in onze printplaat en volledig geroute was. De RFID-lezer anders aansluiten was geen optie alsook de printplaat opnieuw routen zagen we niet onmiddellijk zitten. We hebben vervolgens geopteerd om een hulp-arduino te maken die vervolgens zou instaan voor het lezen van de RFID-tags en deze informatie vervolgens via UART, met behulp van de TX-pin, te verzenden naar de RX-pin van de hoofd-Arduino. Hierdoor hebben we het probleem van de SPI-pinnen kunnen oplossen en hebben we via een eenvoudig protocol, communicatie tot stand gebracht tussen beide Arduino's. De ID-waarde die de hoofd-Arduino vervolgens ontvangt, wordt naar de Bluetooth-bibliotheek geleid vanwaar het via Bluetooth verzonden wordt naar de Raspberry Pi.

```

void Motor::verstuurTag(){
    huidigetijd=millis();
    if (Serial.available() > 0) {
        rfidavailable=true;
        incomingByte = incomingByte + String(Serial.read());
        vorigetijd=huidigetijd;
    }
    if((Serial.available()==0) && rfidavailable==true && (huidigetijd-vorigetijd >20)){
        bluetooth.stuurRFID(incomingByte);
        rfidavailable=false;
        incomingByte="";
    }
}

void Bluetooth::stuurRFID(String Rfidwaarde){
    teVersturenRfid=Rfidwaarde;
    BTSerial.println(teVersturenRfid);
}

```

Listing 5.5: Algoritme voor het versturen van data via de HC-05 naar de Raspberry Pi

5.2 Raspberry Pi

Om de Raspberry Pi werkende te krijgen, hebben we eerst enkele packages moeten installeren om het mogelijk te maken een seriële Bluetooth communicatie tot stand te brengen. Eerst en vooral installeerden we een Bluetooth Interface met behulp van onderstaand commando.

```
sudo apt-get install pi-bluetooth
```

We kunnen nu via preferences onze bluetooth apparaten beheren en paren met de HC-05 van onze Arduino. Van zodra we verbonden zijn, komt er een melding dat we gebruik maken van rfcomm0. Deze naam hebben we ook nodig om ons Python programma te laten samenwerken met de Bluetoothchip op de Raspberry Pi.

Het Python programma is bij ons vrij beknopt. Het is een programma dat voortdurend luistert of er data binnentkomt via Bluetooth en indien er data binnentkomt, deze data vervolgens onmiddellijk afprint. De 2 enige momenten waarbij de Raspberry Pi dus data ontvangt is de snelheid die elke seconde doorgestuurd wordt en elke keer er over een RFID-tag gereden wordt waarbij de ID doorgestuurd wordt. Het Python programma waarvan wij gebruik gemaakt hebben vindt u in Listing 5.6

```
#!/usr/bin/python
import serial
from time import sleep
bluetoothSerial = serial.Serial( "/dev/rfcomm1", baudrate=9600 )
while True:
    data=bluetoothSerial.readline()
    print data
```

Listing 5.6: Code die wordt uitgevoerd door de Raspberry Pi

Hoofdstuk 6

Moeilijkheden

6.1 Software

Het probleem met de integrator is iets waar we niet dieper op zijn ingegaan en niet hebben opgelost. Het probleem dat zich voordoet is het feit de I-waarde die we genereren, telkens de som is van de fout die zich voordoet. Aangezien de code meer dan 1000 keer per seconde wordt uitgevoerd kan deze waarde in een mum van tijd een extreem grote waarden aannemen wat niet gunstig is voor de robot. Het effect van dit probleem op onze robot was dat hij plots extreem versnelde en onmiddellijk alle controle verloor. We zouden dit probleem kunnen oplossen door de I-waarde voldoende klein te kiezen en eventueel slechts om de zoveel tijd deze waarde aan te passen. Na herhaalde pogingen om dit probleem op te lossen hebben we ingezien dat de I-waarde geen noodzakelijk onderdeel is voor een stabiel rijdende robot en hebben we dus besloten dit weg te laten.

Op het moment dat we onze PD-waarden aan het optimaliseren waren, is er een kortsluiting ontstaan in onze Arduino en motor shield terwijl de batterij was aangesloten. De batterij heeft toen een te grote stroom door beide printplaten gestuurd waardoor deze stuk gingen. Ook 3 sensoren hebben toen de brui gegeven. We hebben toen een hele middag gezocht of er een mogelijkheid was om alsnog het motor shield te herstellen aangezien er geen werkende shields op overschat waren. Jammer genoeg was het de L298 die stuk was en hebben we dan een motor shield gekregen waarbij het kroonsteentje stuk was. Na een kleine opknapbeurt hadden we opnieuw een werkende motorshield ter onze beschikking. Onze PD-waarden waren hierdoor weer niet optimaal en hebben dus 2 dagen verloren aan dit gebeuren.

Aangezien we gebruik maken van 2 armen voor het detecteren van beide lijnen van het parcours ontstaat er het probleem dat er 1 arm een hogere prioriteit heeft ten opzichte van de andere arm. De code die we geïmplementeerd hebben zorgt ervoor dat de rechterarm prioriteit heeft boven de linkerarm aangezien de code voor de rechterarm eerst wordt uitgevoerd. Van zodra er 1 sensor actief is van deze arm, zal de code voor de registratie van de sensoren van de linkerarm zelfs niet meer uitgevoerd worden. Dit heeft er voor gezorgd dat onze robot een voorkeursrichting, namelijk tegenwijzerzin, heeft waarin hij met grotere zekerheid de parcours foutloos zal kunnen afleggen. Wanneer we de buitenbocht volgen, wat meestal het geval is, dan bestaat de kans dat bij een plotse scherpe bocht naar links, de robot onvoldoende corrigeert en daardoor met zowel zijn rechterarm, de buitenlijn detecteert, als met zijn linkerarm, de middellijn detecteert. Deze situatie zou tot geen probleem leiden aangezien zijn rechterarm prioritair is en dus de wit-detectie van zijn linkerarm teniet gedaan wordt. Omgekeerd daaren-

tegen, wanneer links de buitenbocht is bijvoorbeeld 010 100, en hij komt met zijn rechterarm op de middellijn terecht, dan zal de robot de verkeerde kant uitschieten aangezien hij met een grote factor wordt gecorrigeerd omdat de sensor die de extreme situatie voorstelt, oplicht. Gelukkig zijn de sensor-armen instelbaar en konden we per circuit de richting en afstand van de armen instellen om zo optimale prestaties te verkrijgen zonder het probleem van prioriteit uit de kant te ruimen. Een mogelijkheid om dit probleem aan te pakken was om bij te houden met een boolean welke arm er actief is en welke arm er net nieuw contact maakt met de witte lijn. De arm die bijvoorbeeld al het langst actief was, kunnen we dan als prioritaire arm beschouwen en elke keer dat we de situatie 000 000 tegenkomen, wat overeenkomt met alle sensoren die zich in het zwart bevinden, kan de prioriteit van de arm opnieuw ingesteld worden.

Voor het eenvoudigste, ovale circuit hadden we onze code geoptimaliseerd op 72. Deze snelheid komt overeen met 0.48 m/s en was een zeer hoge snelheid als je onze robot zag rondrijden. Wanneer we vervolgens het 2e circuit te zien kregen werd het duidelijk dat de robot dit circuit niet aan deze snelheid zou kunnen afleggen. We verlaagden onze snelheid dan naar 65, zodat de robot zonder fouten dit 2e circuit zou kunnen afleggen. Deze snelheidsvermindering ging ook gepaard met nieuwe PD-waarden en we konden met andere woorden opnieuw beginnen met het instellen van de robot. Het probleem waar we nu mee te maken kregen, is dat we een lager vermogen hadden voor de motoren en dus bij sommige grote bulten een duwtje moesten geven om onze robot zijn rit te laten verderzetten. De snelheid opdrijven, zou resulteren in het opnieuw aanpassen van de error values en PD-waarden, waardoor we besloten genoegen te nemen met een snelheid van 0.28 m/s.

Op het moment dat onze printplaat, Arduino gecombineerd met een motor shield, klaar was, wilden we de RFID-lezer connecteren met de Arduino. We stuitten op het probleem dat de RFID-lezer gebruik maakt van de pinnen: 9, 10, 11, 12 en 13 terwijl we onze printplaat reeds geroute hadden waarbij de pinnen 11, 12 en 13 intern verbonden waren met de L298. Dit vormde een groot probleem aangezien het voor de RFID-lezer niet mogelijk was gebruik te maken van andere pinnen aangezien die gebruikt maakt van SPI. Dit probleem konden we oplossen door onze routing volledig opnieuw te doen waarbij we andere pinnen zouden gebruiken om de L298 aan te sturen. Aangezien daar geen tijd voor was, bedachten we dat we konden gebruik maken van een andere hulp-Arduino die instond voor het lezen van de RFID-tags en voor het verzenden via UART naar de hoofd-Arduino. Deze hoofd-Arduino verloor met als gevolg dus slechts 1 pin, namelijk de RX-pin, aan het lezen van de RFID-tags. Aangezien we voordien reeds een prototype-Arduino gemaakt hadden, konden we deze gebruiken als hulp-Arduino en konden we op het einde van de rit alsnog ons project afronden waarbij we enkel gebruik maakten van zelfgemaakte printplaten.

Wanneer we data via Bluetooth verzonden van de Raspberry Pi naar de Arduino en omgekeerd stootten we op het probleem dat er regelmatig karakters fout waren ontvangen en de communicatie dus niet optimaal was. We hebben vervolgens trachten te achterhalen wat het probleem exact was. We merkten op dat de RX- en TX-pinnen, waaraan wij de Bluetooth module hadden aangesloten, ook rechtstreeks verbonden zijn met de USB-poort. Wanneer we dus met behulp van Serial.println(), data wilden wegschrijven naar de seriële monitor, gebeurde dat als het ware op dezelfde datalijn als de data die we ontvingen via de Bluetooth module. We konden dit probleem oplossen door niet langer gebruik te maken van Hardware Serial maar wel door Software Serial waarbij we de pinnen A2 en 2 gebruikten als respectievelijk RX en TX.

6.2 Hardware

We waren er ons van bewust dat het vrijwel onmogelijk was dat onze PCB geen kinderziekten zou hebben bij de eerste uitvoering. Dit wisten we eigenlijk bijna zeker omdat we een probleem hadden bij de L298, het ground-vlak viel wat groter uit dan de voorziene plaats van de footprint. Dit ging daardoor contact maken met een baantje waarover +5V en dus eigenlijk kortsluiting maken. We hebben wat zitten denken en we hebben dit probleem opgelost door een plakker tussen het ground-vlak en het baantje te plakken waardoor ze dus geen contact konden maken. Dit was wel degelijk een oplossing voor dit probleem maar het was niet ideaal, dit was dus al een reden om een nieuwe printplaat te maken en dit te veranderen. We hadden ook problemen met de reset-knop, de afmetingen van de gebruikte footprint klopte ook niet. We moesten de resetknop anders dan normaal bevestigen omdat anders de verkeerde benen zouden verbonden zijn. We zien op de foto??twee beentjes aan twee zijden, deze twee beentjes worden verbonden door de knop in te drukken. Bij ons waren de afmetingen van de footprint omgewisseld waardoor de niet-verbonden beentjes constant verbonden gingen zijn waardoor de reset knop de werking van de PCB zou verhinderen. We hebben dit opgelost door gebruik te maken van een drukknop met slechts twee pinnen die verbonden worden met elkaar bij het indrukken van de knop. We zaten dan wel met het probleem dat deze knop veel langer was en dus de pinnen een opvulvlak konden raken. Dit wilden we liever verhinderen waardoor we een stukje van het vlak vlak naast de drukknop losgemaakt hebben van een groter ander deel van dat vlak door koper weg te krassen. Hierdoor waren we zeker dat eventuele andere verbindingen door solderingen met dit vlak geen kortsluiting kon veroorzaken of andere problemen geven. We controleerden wel telkens dat geen enkele solduur contact maakte met een opvulvlak maar we kunnen altijd iets over het hoofd zien en we wilden geen risico nemen. Toen we na het vele controleren bij het overbrengen van de kabels van de Arduino + Motorshield naar de ArduMoto merkten dat als we spanning op de PCB aanlegden, een paar componenten warm werden en een elco zelfs gloeiend heet werd. We wisten dus dat er nog enkele problemen waren met de motoraansturing. We hebben na zeer lang zoeken en met hulp van onze coach 3 problemen ontdekt. We hebben deze problemen ook aangeduid op figuur??Probleem 1 was een Schottky-diode die niet goed verbonden was met zijn baantje. De diode maakte soms wel contact en soms niet. Daar de footprint nog steeds klein uitviel hebben we het opgelost door een extern draadje te strippen en aan het contactvlak te solderen en te verbinden met het baantje. Probleem 2 was simpelweg een Via die we vergeten waren aan te leggen waardoor een pin van de L298 niet verbonden was. Probleem 3 was het de grootste boosdoener. We hebben in het schema vergeten de IC voor de motoraansturing te voeden met de PWRIN. Hierdoor werden de signalen niet goed verwerkt door de IC en kregen de motoren dus ook niet de goede signalen. We hebben dit nogmaals opgelost door een draad te strippen en de verbinding aan te leggen tussen de IC en de PWRIN. De PWRIN is het signaal dat rechtstreeks via de kroonsteentjes binnenkomt van de batterij. We hebben hiervoor gebruik gemaakt van een dikkere draad omdat er toch enkele volts door deze kabel moeten.

Hoofdstuk 7

Coach

Dit jaar was het de eerste keer dat aan elk team een persoonlijke coach werd toegekend bij wie we konden langsgaan indien we met vragen zaten over de uitwerking van het project. Tijdens de projectweek moesten we nadenken over onze zwakke punten en en onze sterktes en aan de hand daarvan hebben we een persoonlijke begeleider gekregen, in ons geval Kevin. Wij hadden vooraf vooral schrik voor het PCB-design en het kunnen verklaren van het al dan niet weglaten van bepaalde componenten. Kevin stond altijd klaar om op onze vragen te antwoorden en als het nodig was ons te helpen zoeken naar het probleem. Dit hadden we vooral nodig voor het installeren van de L298 voor de motoraansturing waar er nog enkele foutjes in zaten. Kevin onzag het niet om enige tijd te helpen zoeken naar de fout in de PCB, die we uiteindelijk hebben kunnen vinden en oplossen. Kortom, voor ons was een persoonlijke coach een zeer goede ervaring. We konden altijd terecht bij de hoofdbegeleiders Guus en Stijn, maar het was altijd handig om bij meer specifieke vragen te kunnen vragen aan Kevin.

Hoofdstuk 8

Besluit

We kunnen terugkijken op een zeer leerrijke en toch wel zeer plezante bachelorproef. De twee aspecten van Elektronica zaten erin verwerkt, nl. hardware maar ook een belangrijk deel software. We hebben ook enkele testjes uit moeten voeren om bijvoorbeeld te beslissen hoe we de sensors moesten plaatsen, ver/dicht van het wagentje, schuin of recht,... Dan moesten we ook zelf beslissen welke modules we gebruikten, wat we wel en wat niet zelf moesten maken. We hebben toch wel de meeste foutjes en problemen tegengekomen die iedereen in een leerproces moet meemaken. Bij ieder nieuw PCB-design zitten er wel altijd enkele foutjes, dit was in ons geval natuurlijk niet anders. Sommige problemen konden we zeer snel en gemakkelijk oplossen, andere problemen hebben bloed, zweet en tranen gekost om te vinden en op te lossen. Bij het software onderdeel was het vooral de uitdaging om de PID (in ons geval PD) afregeling te doen, vooral omdat we dit nog in geen enkel vak in theorie hadden gezien. We kunnen wel met een positieve blik terugkijken op onze vormgeving en afregeling van ons wagentje. We waren over alle vier de parcours die we voorgesloten kregen de snelste, we moeten er wel bij vertellen dat we 4x uit het parcours zijn geraakt. Twee keer meer dan de tweede groep die in het totaal 2x uit het parcours ontsnapte. Kortom uit deze bachelorproef hebben we volgende zaken geleerd: PID afregelen, Bluetooth connectie maken met RPI, RFID-tags uitlezen, PCB ontwerpen, PCB debuggen, SMD solderen, 3D-printen.

FACULTEIT INDUSTRIELE INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GENT
Gebroeders De Smetstraat 1
9000 GENT, België
tel. + 32 92 65 86 10
fax + 32 92 25 62 69
iiw.gent@kuleuven.be
www.iiw.kuleuven.be



LID VAN
**ASSOCIATIE
KU LEUVEN**