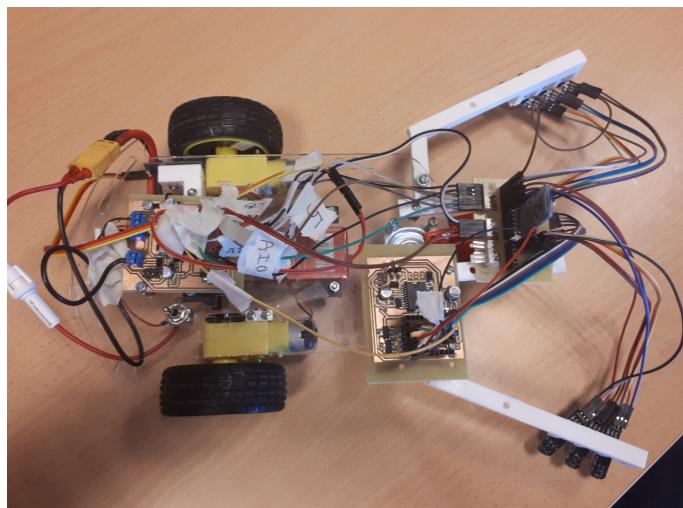


Robot Bartje The Champion of 3EictE



**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	Dankwoord	1
2	Opdrachtbeschrijving	2
2.1	Hardware	2
2.2	Software	3
3	Planning	4
4	Onkosten	5
5	Vormgeving van de Robot	6
6	Hardware	9
6.1	Tussenstuk	9
6.2	Arduino met L298	10
6.2.1	Hulp Arduino	10
6.2.2	Hoofd Arduino	11
7	Software	17
7.1	Arduino	17
7.1.1	Motor	18
7.1.2	PD-algoritme	18
7.1.3	Detectie-algoritme	20
7.1.4	Communicatie	20
7.2	Raspberry Pi	21
8	Moeilijkheden	23
8.1	Software	23
8.2	Hardware	25
9	Coach	26
10	Besluit	27

Lijst van figuren

2.1	Een Arduino gecombineerd met een motorshield	3
3.1	Planning met milestones	4
5.1	Arm voor 3 sensoren.	7
5.2	Arm voor sensoren die we gebruiken in ons eindontwerp	7
6.1	Routing van de PCB die als verlengstuk dient.	10
6.2	De gebruikte Bluetooth-module, ingeplugged op op het tussenstuk	10
6.3	Voorzijde eigen PCB	11
6.4	Achterzijde eigen PCB	11
6.5	Gebruikte RFID-reader MRFC522.	11
6.6	USB interface die we weg hebben gelaten	12
6.7	Voorbeeld van LED die we weggelaten hebben	13
6.8	Voorbeeld van weerstand die we weggelaten hebben rond de Atmega328	13
6.9	Front banen van onze eigen PCB (= onderkant voor ons)	14
6.10	Back banen van onze eigen PCB (= bovenkant voor ons)	14
6.11	Eindresultaat achterkant PCB	15
6.12	Eindresultaat voorkant PCB	15
6.13	Volledig schema van de hoofd- en hulp-Arduino	16
7.1	Flowchart mainprogramma	17
7.2	Flowchart MotorPID	19
8.1	Probleem 1 en 3 bij het PCB ontwerp	25
8.2	Probleem 2 bij het PCB ontwerp	25

Lijst van tabellen

4.1 Tabel met aankopen en hun prijs.	5
6.1 Tabel afmetingen koperbanen.	13

Hoofdstuk 1

Dankwoord

Graag willen wij enkele mensen bedanken die ons de afgelopen maanden geholpen hebben met ons project. Om te beginnen willen we Guus Leenders en Stijn Crul bedanken daar ze ons geholpen hebben met het 3D-printen van onderdelen. Dit waren ook de hoofdbegeleiders van het project bij wie we terecht konden met vragen over de opdracht . We willen verder ook onze coach Kevin Verniers bedanken bij wie we altijd terecht konden met vragen over de uitwerking van de robot. We bedanken als laatste onze klasgroep, iedereen hielp elkaar met kleine vraagjes en problemen.

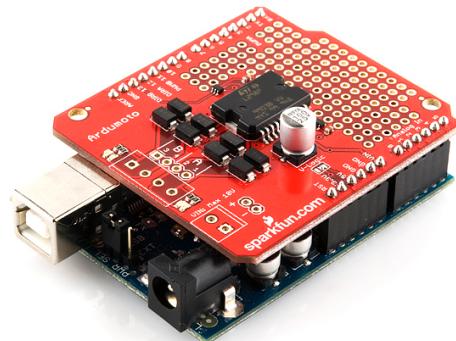
Hoofdstuk 2

Opdrachtbeschrijving

De opdracht bestaat erin om een line-following robot te maken. Deze robot wijkt enigszins af van de klassieke line-follower aangezien de circuits die wij moeten kunnen afleggen, bestaan uit 2 volle buitenlijnen en een gestreepte middellijn, waardoor we niet zomaar de middellijn kunnen volgen. Deze uitdagingen maken een volgalgoritme zeer ingewikkeld en onvoorspelbaar. Er zijn in totaal 3 verschillende circuits die de robot autonoom zou moeten afleggen terwijl hij simultaan andere taken uitvoert zoals snelheden registreren of RFID-tags uitlezen en deze doorsturen. De breedte van de baan is ongeveer twee maal de breedte van de auto. We kregen een basispakket waarmee we aan de slag konden 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 te printen die we nodig hadden om sensoren te bevestigen en dergelijke. Verder kregen we ook nog een budget van 50 euro ter beschikking van de school om ons te voorzien van de nodige componenten. De opdracht bestaat uit zowel 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.

2.1 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 2.1, maar voor de uiteindelijke opdracht moesten we natuurlijk gebruik maken van zelfvervaardigde printplaten (PCB's). Aan de hand van een schema van de Arduino Uno, dat beschikbaar is op het internet, konden we onze eigen PCB ontwerpen waarbij we alle niet-noodzakelijke componenten weglieten en de L298, de chip die verantwoordelijk is voor de motoraansturing, toevoegden. Voor de sensoren, Bluetooth-module en RFID-reader, waren we vrij om te kiezen wat we wilden gebruiken.



Figuur 2.1: Een Arduino gecombineerd met een motorshield

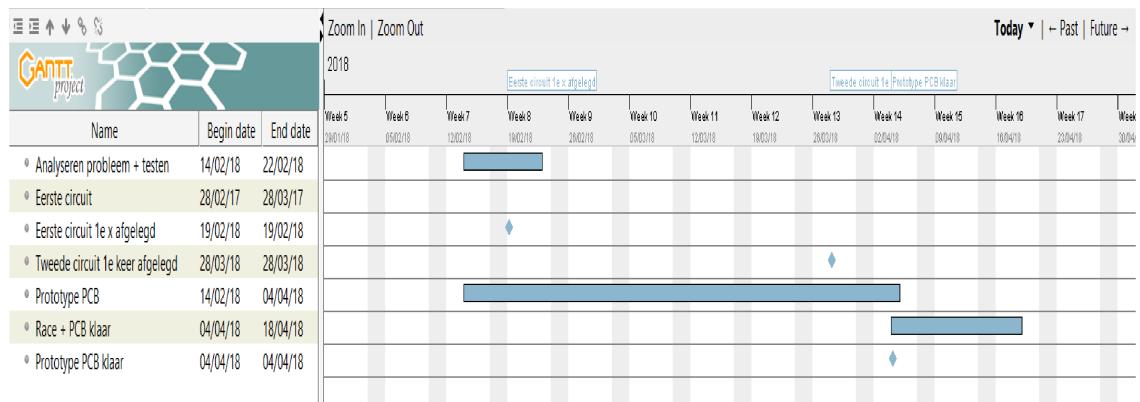
2.2 Software

Het tweede onderdeel van de opdracht bestaat erin om de Arduino (en later 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 data via Bluetooth doorsturen naar de Raspberry Pi.

Hoofdstuk 3

Planning

We hadden een planning opgesteld die weergegeven is in figuur 3.1. We hebben eerst ruim de tijd genomen om het probleem uitgebreid te analyseren zodat we ons vanaf het begin konden focussen op één ontwerp. We konden vrij snel het eerste circuit afleggen maar hebben dan nog veel tijd gespendeerd om de code te optimaliseren vooraleer we naar volgende circuits over gingen. We hebben deze planning algemeen gezien zeer goed kunnen volgen.



Figuur 3.1: Planning met milestones

Hoofdstuk 4

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 al voorzien en hebben we dus niet opgenomen in deze onkostentabel. In onderstaande tabel vermelden we enkel de componenten die we extra aangekocht hebben voor ons project. Zoals we reeds vermeld hebben in hoofdstuk 2, konden we ook vrij gebruik maken van een 3D-printer waarvan we de kosten niet in rekening brengen. Slechts drie verschillende zaken zijn aangekocht en kan u terugvinden in tabel 4.1.

Tabel 4.1: Tabel met aankopen en hun prijs.

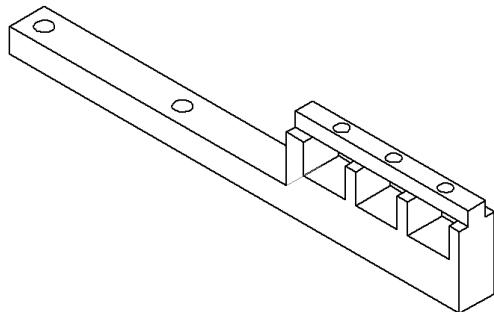
Beschrijving aankoop	Prijs (in euro)
HCO5 (Bluetooth module)	5.37
MRFC522 (RFID Reader)	4.15
10x Line-following Sensors	2.16
Totaal:	11.68

Hoofdstuk 5

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 robot er zou moeten uitzien om het circuit zo foutloos en snel mogelijk te kunnen afleggen. Een eerste keuze is de plaatsing van het losse wieltje vooraan of achteraan. We bekeken beide opties 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.

Het oorspronkelijke doel was om onze robot autonoom de circuits te laten rijden met behulp van één arm van dertien cm waarop drie sensoren bevestigd zijn zoals te zien in Figuur 5.1. We hebben dan ook zo snel mogelijk deze arm ontworpen en geprint met de 3D-printer zodat wij onmiddellijk konden beginnen met het schrijven 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 langas van onze robot waardoor we de afstand tussen de stuurwielen en de sensoren niet maximaliseerden. Nadat we de arm onder een hoek van ongeveer 30° ten opzichte van de langas van de robot geplaatst hadden, merkten we een grote prestatieverbetering op en konden we de snelheid opdrijven.

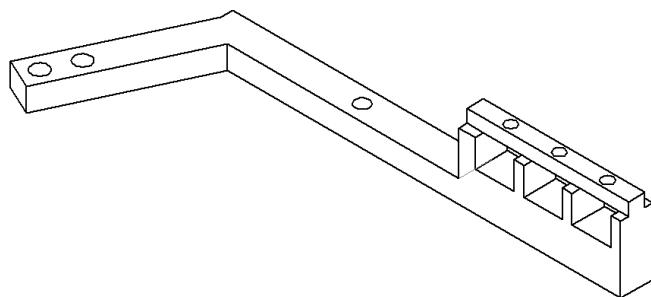


Figuur 5.1: Arm voor 3 sensoren.

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, leek sterk op de eerste, maar 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 bestaat uit 2 armen, die elkaars spiegelbeeld zijn en waarbij de top van de arm opnieuw een hoek van 30° maakt met de langas van de robot zoals u kan zien in Figuur 5.2. U kan dergelijke arm vinden in Figuur 5.2. 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 boven de lijn begeeft. Deze armen bleken voor ons project optimaal en gebruikten we dan ook in ons eindontwerp.



Figuur 5.2: Arm voor sensoren die we gebruiken in ons eindontwerp

We ontwierpen verder ook nog een houder 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 kleine houder 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 een houder 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 6

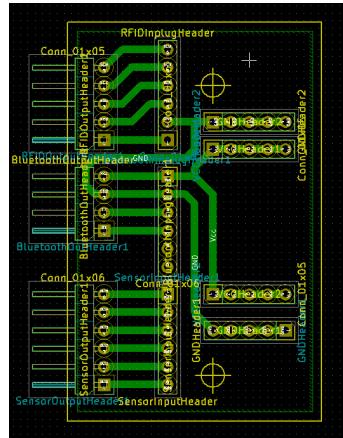
Hardware

Onze robot werkt met behulp van 3 zelfgemaakte PCB's. Twee ervan zijn de combinatie van de Motorshield en de Arduino/Atmega328 en de andere is een tussenstuk om de bekabeling te reduceren.

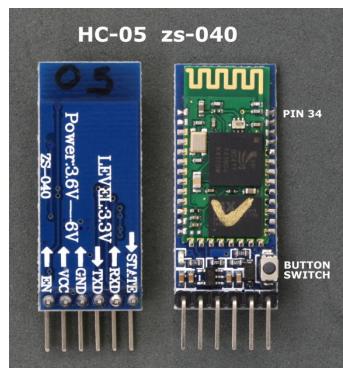
6.1 Tussenstuk

Het tussenstuk is een PCB waarop alle lijnen van de infrarood sensoren toekomen en worden doorgegeven aan de Atmega328. Het voordeel van deze printplaat is dat we nu slechts één GND en één VCC lijn moeten voorzien tussen de PCB en de Arduino. We hebben op deze PCB 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 tien 5V-aansluitingen voorzien, tien GND-aansluitingen, zes 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 Atmega328. Enkel de signalen VCC, GND, TXD en RXD worden van een uitgang voorzien. De VCC en de GND aan deze uitgang zorgen 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 6.2.



Figuur 6.1: Routing van de PCB die als verlengstuk dient.



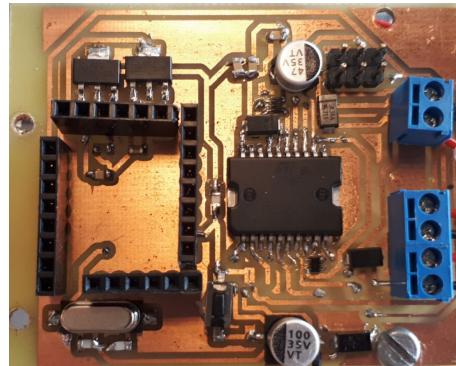
Figuur 6.2: De gebruikte Bluetooth-module, ingeplugged op het tussenstuk

6.2 Arduino met L298

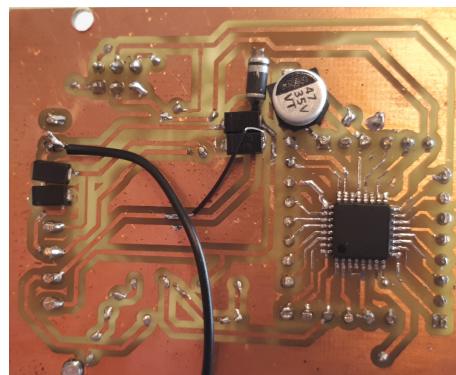
6.2.1 Hulp Arduino

De tweede printplaat die we ontworpen hebben, is een combinatie van de Atmega328 en de motorshield, hoewel we hier enkel gebruik zullen maken van de Atmega328. De voorzijde van deze PCB kan u zien op Figuur 6.3 en de achterzijde op Figuur 6.4. Deze PCB zorgt enkel voor de aansturing en uitlezing van de RFID-reader. We konden de RFID-reader niet aansluiten op onze hoofd Arduino, aangezien 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 SPI. Door I^2C te gebruiken zouden we geen conflicten met de pinnen gehad hebben waardoor er geen

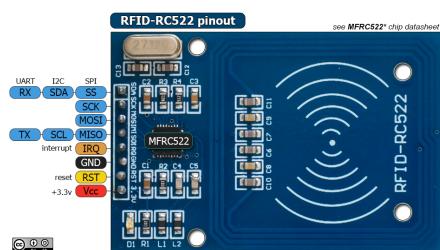
tweede PCB nodig was. We maken gebruik van de MRFC522, dewelke wordt weergegeven in figuur 6.5. Om een RFID-tag zo goed mogelijk uit te lezen moet de reader op ≈ 1 cm boven de tag passeren. We hebben hiervoor zelf een RFID-houder geprint met de 3D-printer zodat deze afstand te allen tijde constant blijft.



Figuur 6.3: Voorzijde eigen PCB



Figuur 6.4: Achterzijde eigen PCB



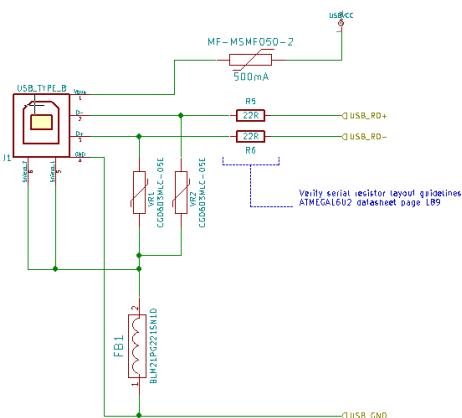
Figuur 6.5: Gebruikte RFID-reader MRFC522.

6.2.2 Hoofd Arduino

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 (module verantwoordelijk voor de Bluetooth-communicatie). Deze PCB is identiek aan de hulp Arduino en kan u opnieuw zien op Figuur 6.3 en op Figuur 6.4. Het schema kan u raadplegen op Figuur 6.13. Na het finaliseren van

de PCB bleken er nog enkele fouten te zijn geslopen 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 diode bleek bij het testen niet goed gesoldeerd 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 hiëarchical 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.

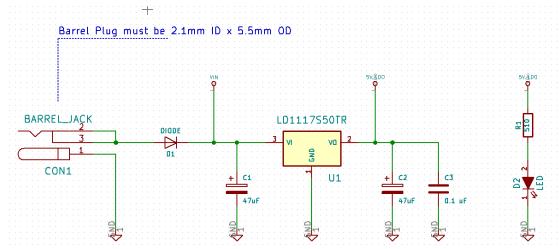
Als vertrekpunt hadden we het schema van een Arduino Uno ter beschikking op het internet. Als eerste konden we de USB-interface voor de aansluiting van een USB-kabel weglaten aangezien we dit in ons finaal ontwerp niet nodig hadden. Deze USB-interface wordt weergegeven in figuur 6.6. Voor het uploaden van een programma maken we dan



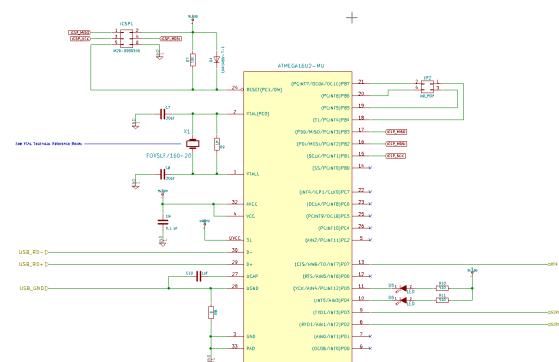
Figuur 6.6: USB interface die we weg hebben gelaten

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. Een voorbeeld van een LED die we weggelegd hebben wordt weergegeven in figuur 6.7. 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 kunnen interpreteren. Zo veranderden we bijvoorbeeld MISO door DIR A aangezien deze lijn verantwoordelijk was voor de richting van Motor A. R8 hebben we ook weggelegd aangezien dit een weerstandswaarde van $0\ \Omega$ heeft. Deze weerstand wordt voornamelijk gebruikt als brug om er banen onder te kunnen trekken. Pinnen 27 en 28 hebben we niet nodig, dus mogen we de condensator verwijderen. De jumper hebben we weggelegd omdat we die niet nodig achten. Dit alles wordt weergegeven in figuur 6.8. Eenmaal het schema klaar was, konden we starten met het routen van de printbanen. De afmetingen van de PCB bedragen (52,324mm x 61.468mm). De uiteindelijke grootte van de printplaat bedraagt 53.34mm x 73.152mm, zodat de bevestigingsgaten van de PCB overeenkomen met de vijzen van het chassis. De breedte van de printbanen van onze oorspronkelijke routing bedroeg 10 mills(= 0.254mm). Aangezien

dit zeer smal was en de kans groot was dat er een baan weggeëet zou worden, hebben we zoveel mogelijk banen verbreed tot 30 mills. De breedte van de printbanen en de diameters van de drills, worden weergegeven in tabel 6.1.



Figuur 6.7: Voorbeeld van LED die we weggelaten hebben



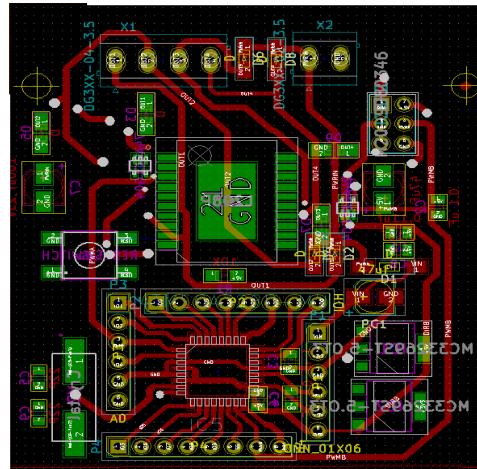
Figuur 6.8: Voorbeeld van weerstand die we weggelaten hebben rond de Atmega328

Tabel 6.1: Tabel afmetingen koperbanen.

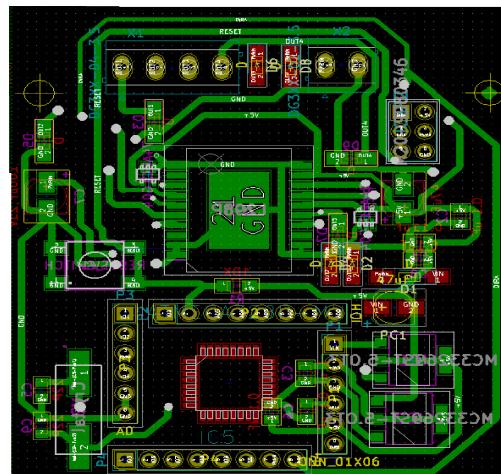
	Clearance	Track Width	Via Dia	Via Drill
Default	0.25	0.25	0.6	0.4
0.5	0.25	0.5	0.7	0.5
12 mills	0.25	0.3	0.6	0.4
15 mills	0.25	0.38	0.6	0.4
30 mills	0.25	1	0.6	0.4

Enkele banen hebben een breedte van maximaal 15 mills, aangezien de printplaat zodanig compact gemaakt was dat nog meer verbreden geen optie was. Één van onze uitdagingen was om de printplaat te voorzien van zo weinig mogelijk via's, wat we konden minimaliseren tot 20, op de hele PCB. Bij de eerste pogingen om de printplaat te etsen, stootten we op het probleem dat de fijnste banen oplost waren door er niet langer verbinding was. We kregen de tip om gebruik te maken van een kopervlak, zodat de PCB minder lang in het zuurbad moest liggen bij het etsen. Achteraf gezien konden we beide vlakken gebruiken als bijvoorbeeld een GND-vlak, wat in ons ontwerp niet is toegepast. De routing van voor- en achterkant wordt weergegeven in

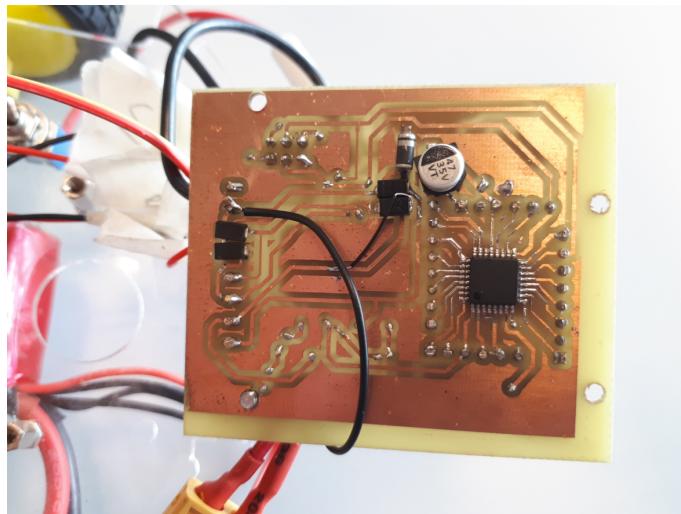
figuren 6.9 en 6.10. Er moet wel bij verteld worden dat we per ongeluk de zijden omgewisseld hebben, de Back-zijde op KiCad is voor ons de voorkant en vice versa. Achteraf gezien gaf dit toch enkele problemen omdat we altijd in spiegelbeeld moesten redeneren. Dit zullen we in het vervolg dus proberen te vermijden. Eens de PCB volledig bestukt was, wilden we deze eerst testen met een standaardprogramma voor de Atmega nl. Blink-Led. Dit programma werkte onmiddellijk waardoor we verkeerdelyk veronderstelden dat de hele PCB correct was. We bespreken dit uitgebreid in het hoofdstuk 8. De volledige praktische PCB wordt weergegeven in figuren 6.11 en 6.12.



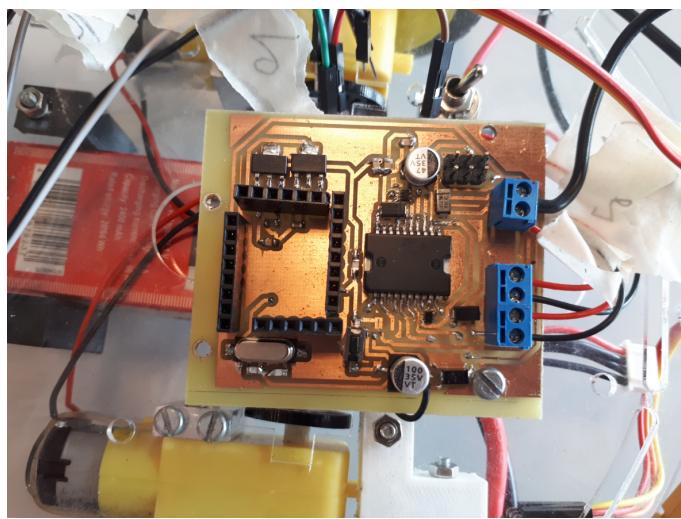
Figuur 6.9: Front banen van onze eigen PCB (= onderkant voor ons)



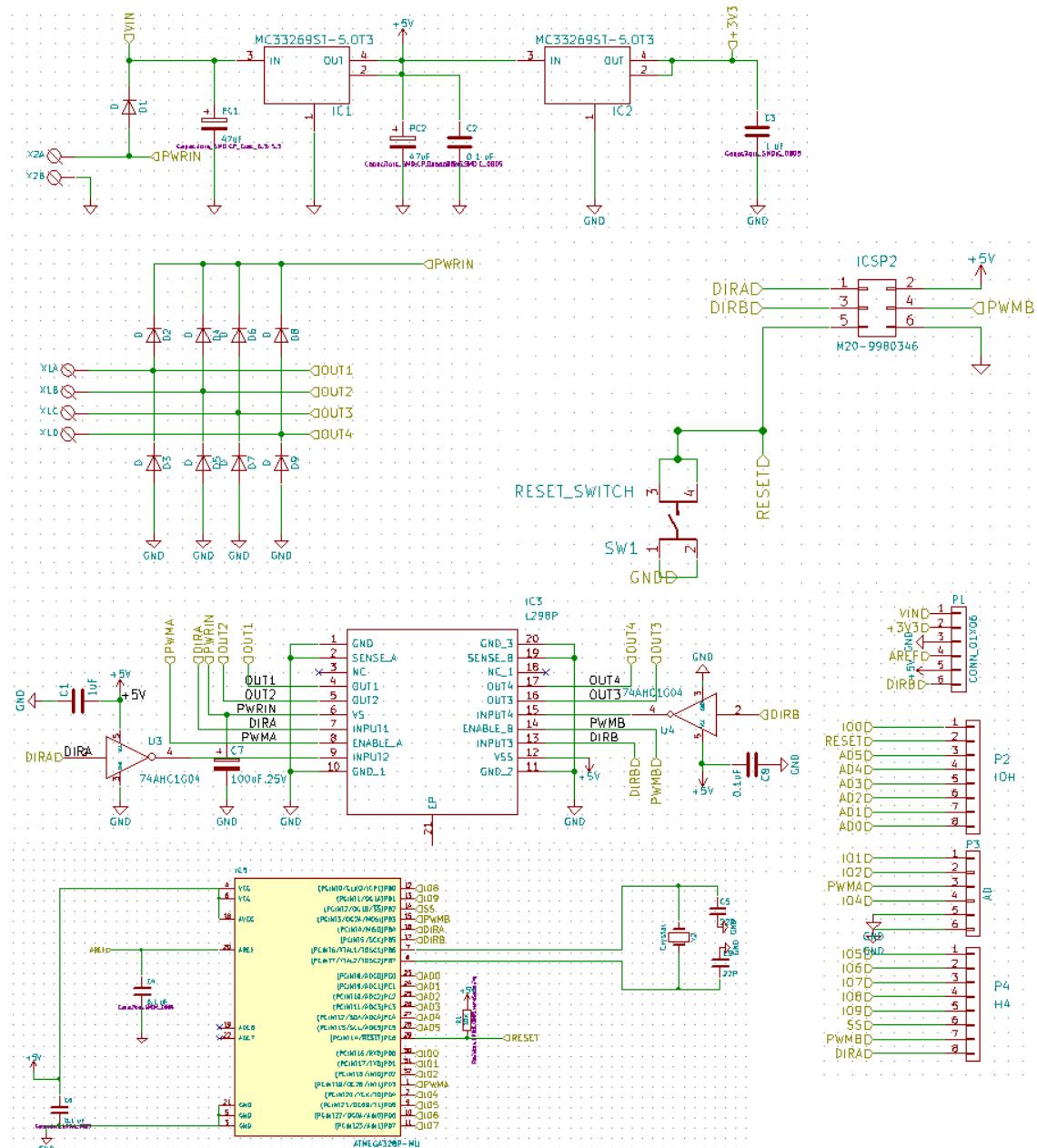
Figuur 6.10: Back banen van onze eigen PCB (= bovenkant voor ons)



Figuur 6.11: Eindresultaat achterkant PCB



Figuur 6.12: Eindresultaat voorkant PCB



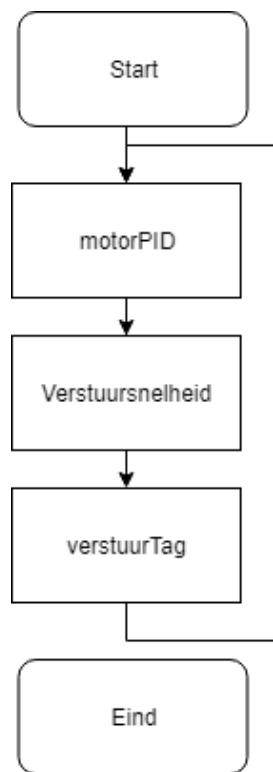
Figuur 6.13: Volledig schema van de hoofd- en hulp-Arduino

Hoofdstuk 7

Software

7.1 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. Dit zorgt ervoor dat het programma overzichtelijker wordt. We gebruiken dan ook slechts één lijn code in ons .ino bestand om een functie uit een bibliotheek aan te roepen van waaruit vervolgens het hele programma wordt aangestuurd. In Figuur 7.1 vindt u de flowchart van ons programma.



Figuur 7.1: Flowchart mainprogramma

7.1.1 Motor

We zijn eerst en vooral begonnen met het schrijven van de Motor-bibliotheek. De functie `beaalPID()` 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. We stellen dergelijke array voor als volgt: 000 001 waarbij nu enkel de meest rechtse infrarood sensor wit detecteert en alle vijf 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 klein mogelijke 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 7.1. De error value die gegenereerd werd, wordt vervolgens gebruikt in de corrigeer-methode waarin de werking van ons PD-algoritme 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 7.1: Het algoritme die de foutwaarden toekent aan de verschillende situaties

7.1.2 PD-algoritme

Dit algoritme is vrij beknopt maar toch zeer interessant aangezien het de prestaties van de robot 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 foutcorrectie, gegenereerd door de integrerende actie, is niet alleen afhankelijk van de grootte van de fout, maar ook van de tijd gedurende dewelke de fout zich voordoet. De I-waarde maakt het mogelijk om de offsetfout zelfs volledig te herleiden tot nul. Ook deze waarde wordt nog vermenigvuldigd met een constante namelijk KI. Als laatste hebben we nog de differentiërende term. Deze waarde reageert op een plotselinge verandering, wat de reactiesnelheid van onze robot doet toenemen. Wanneer de afwijking tussen de gewenste waarde en de gemeten waarde plots toeneemt, zal de D-waarde hevig reageren om zo een mogelijks zeer groot wordende fout te beperken in grootte. Ook in de andere richting zal de correctie afgerekend 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

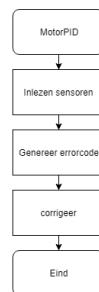
I-waarde hebben we in dit project niet gebruikt aangezien deze waarde enorm snel toeneemt en niet eenvoudig was om perfect in te stellen. Achteraf gezien bleek deze term niet noodzakelijk om een stabiel rijdende robot te verkrijgen. We maken dus gebruik van een PD-regelaar, niet van een PID-regelaar. 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 honderden keren per seconde wordt uitgevoerd. Wanneer de fout gedetecteerd wordt en er geen tijdsrestrictie is, zou er onmiddellijk een nieuwe waarde worden 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.

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

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

Listing 7.2: 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 7.3, die we met behulp van beproeving hebben bepaald om de beste prestaties te verkrijgen. De som van deze producten wordt opgeslagen in PIDvalue en gebruikt in de rij-methode. Deze methode is verantwoordelijk voor het instellen van de snelheid per wiel, rekening houdend met de PIDvalue die we net definiereiden. Dit alles wordt nog eens weergegeven in een flowchart in figuur 7.2.



Figuur 7.2: Flowchart MotorPID

```
Kp=12;
Ki=0;
Kd=4;
P = error;
stoptijd=millis();
if((stoptijd-starttijd)>500){
    if(abs(error-previousError)>0{
        //I = I + error;
        D = error-previousError;
        previousError = error;
        starttijd=millis();
    }
}
PIDvalue = (Kp*P) + (Ki*I) + (Kd*D);
rij(PIDvalue);
```

Listing 7.3: Het PD-algoritme dat wij hanteren

7.1.3 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. 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 twee 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 verkrijg 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 twee 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 tweede 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 twee armen te kunnen laten samenwerken.

7.1.4 Communicatie

In de methode bepaalPID() bevinden er zich nog twee andere methodes die aangeroepen worden, namelijk verstuurSnelheid() en verstuurTag(). Beide methodes maken gebruik van functies van de Bluetooth-bibliotheek. Bluetooth.stuurSnelheid() stuurt een waarde door die gegenereerd wordt door een methode uit de Speed-bibliotheek, waarin zich het algoritme voor de registratie van de huidige snelheid van de robot bevindt die u kan raadplegen in Listing 7.4. De snelheidswaarde wordt telkens wanneer het wieltje een rotatie maakt, aangepast met behulp van een eenvoudige snelheidsvergelijking. De snelheid wordt vervolgens één keer per seconde met behulp van Software Serial verstuurd 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 7.4: 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 ontvangt, wordt vervolgens via Bluetooth verzonden 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 7.5: Versturen van data via de HC-05

7.2 Raspberry Pi

Om de Raspberry Pi werkende te krijgen, hebben we eerst enkele packages moeten installeren om een seriële Bluetooth communicatie mogelijk te maken. 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 bijvoorbeeld rfcomm0. Deze naam hebben we ook nodig om ons Python programma te laten samenwerken met de Bluetooth-chip op de Raspberry Pi.

Het Python programma is bij ons vrij beknopt. Het is een programma dat pollt of er data binnenkomt via Bluetooth en indien er data binnenkomt, deze vervolgens onmiddellijk afprint.

De Raspberry Pi ontvangt slechts op twee momenten data, namelijk de snelheid die elke seconde doorgestuurd wordt en tijdens het overrijden van een RFID-tag. Het Python programma waarvan wij gebruik gemaakt hebben vindt u in Listing 7.6

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

Listing 7.6: Code die wordt uitgevoerd door de Raspberry Pi

Hoofdstuk 8

Moeilijkheden

8.1 Software

Het probleem met de integrator is iets waar we niet dieper op ingegaan zijn en niet hebben opgelost. Het probleem dat zich voordoet is het feit dat de I-waarde die we genereren, telkens de som is van de fout die zich voordoet. Aangezien de code honderden keren per seconde wordt uitgevoerd kan deze waarde in een mum van tijd een extreem grote waarde 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 KI-waarde voldoende klein te kiezen en eventueel slechts om de zoveel tijd de I-waarde aan te passen. Na herhaaldeijke 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 deze term 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 aangesloten was. De batterij heeft toen een te grote stroom door beide printplaten gestuurd waardoor deze stuk gingen. Ook 3 sensoren gingen toen in rook op. We hebben toen een hele middag gezocht of er een mogelijkheid was om alsnog de motorshield 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 door de nieuwe shield niet meer optimaal en waardoor we dus 2 dagen verloren hebben 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. Deze situatie zou tot geen probleem leiden aangezien zijn rechterarm prioritair

is en dus de wit-detectie van zijn linkerarm teniet gedaan wordt. Omgekeerd daarentegen, 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 0.40 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. 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.

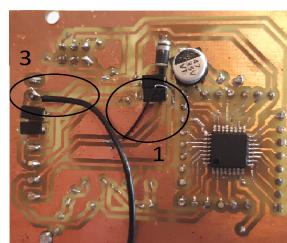
Op het moment dat onze printplaat, Arduino gecombineerd met een motor shield, klaar was, wilden we de RFID-lezer connecteren met de Arduino. We stuitten op het probleem dat de RFID-lezer gebruik maakt van de pinnen: 9, 10, 11, 12 en 13 terwijl we onze printplaat reeds geroute hadden waarbij de pinnen 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 omdat er gebruik gemaakt wordt 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.

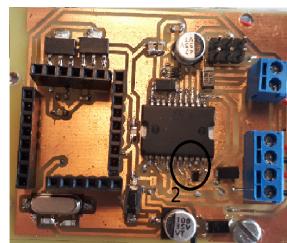
8.2 Hardware

We waren er ons van bewust dat het vrijwel onmogelijk was dat onze PCB geen kinderziekten zou vertonen bij ons eerste ontwerp. De problemen kwamen vrijwel onmiddellijk tot uiting bij de L298. Het ground-vlak viel wat groter uit dan door de footprint voorzien was waardoor dit vlak contact maakte met een baan waarover +5V stond en dus eigenlijk kortsluiting vormde. We hebben dit probleem kunnen oplossen door plakband tussen het ground-vlak en de printbaan te kleven om zo het contact te verbreken. Hoewel de oplossing niet ideaal was, waren we hiermee wel geholpen. We hebben dan onmiddellijk de footprint gewijzigd zodat onze finale PCB dit probleem niet zou vertonen. We hadden ook problemen met de reset-knop waarbij de afmetingen van de gebruikte footprint niet klopten. 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.

Bij het testen van de PCB merkten we vrijwel onmiddellijk op dat een paar componenten warm werden. Dit was een directe indicatie van het feit dat de printplaat nog niet op punt stond. We hebben na lang zoeken en met hulp van onze coach, 3 problemen ontdekt. We hebben deze problemen ook aangeduid op figuren 8.1 en 8.2. 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 van de diode nog steeds te klein uitviel hebben we het probleem opgelost door een draadje te solderen aan de contactzijde om zo een verbinding te maken met de baan. Probleem 2 was simpelweg een via die we over het hoofd gezien hadden waardoor een pin van de L298 niet verbonden was. Probleem 3 was de grootste boosdoener. We waren in het schema één label vergeten te plaatsen waardoor de L298, die in staat voor de motorsturing, niet gevoed werd met PWRIN. Hierdoor werden de signalen niet goed verwerkt door de L298 en kregen de motoren dus ook niet de juiste signalen. We hebben dit nogmaals opgelost door een verbinding te solderen tussen de IC en de PWRIN. De PWRIN is het signaal dat rechtstreeks via de kroonsteentjes binnenkomt van de batterij.



Figuur 8.1: Probleem 1 en 3 bij het PCB ontwerp



Figuur 8.2: Probleem 2 bij het PCB ontwerp

Hoofdstuk 9

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 en sterke punten 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 dat zich voordeed. Kevin ontzag het niet om enige tijd te helpen zoeken naar de fout in de PCB, die we uiteindelijk hebben kunnen vinden en oplossen. Verder konden we ook bij hem terecht voor feedback van ons verslag. Kortom, voor ons was de persoonlijke coach een zeer goede ervaring. We konden ook altijd terecht bij de hoofdbegeleiders Guus en Stijn, hoewel we voor de uitwerking van het project zelf eerder geneigd waren contact op te nemen met Kevin.

Hoofdstuk 10

Besluit

We kunnen terugkijken op een zeer leerrijke en plezante bachelorproef. De twee aspecten van elektronica zaten in het project verwerkt, namelijk de combinatie hardware en software. We hebben ook testen uitgevoerd om tot bepaalde inzichten te komen omtrent het plaatsen van de sensoren. We zijn gestoten op enkele problemen tijdens de uitwerking van de opdracht wat ons zeker ten goede gedaan heeft waardoor we inzichten verworven hebben in de elektronica. 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 optimaliseren. We waren over alle vier de circuits, die we voorgeshoteld kregen, de snelste wat het natuurlijk nog een extra prestatie maakt. Kortom, uit deze bachelorproef hebben we volgende zaken geleerd: PID afregelen, Bluetooth connectie maken met RPI, RFID-tags uitlezen, PCB ontwerpen, SMD solderen en 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**