KU LEUVEN

FACULTEIT INDUSTRIELE
INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS OOSTENDE

Besturen van een auto met behulp van zichtbaar licht communicatie

Pieter BERTELOOT
Pascal BARBARY
Pieter DEWACHTER
Kristof T'JONCK

Projectlab bachelor electronica-ICT

Academiejaar 2016-2017

INHOUD

1 PROBLEEMSTELLING

KRISTOF T'JONCK

De opdracht bestaat er in een applicatie te maken die digitale communicatie met zichtbaar licht demonstreert. Deze applicatie moet aan volgende vereisen voldoen:

- Minimum 1 microcontroller
- Communicatie tussen een PC/smartphone en een microcontroller.
- Een applicatie op de PC/Smartphone (stand-alone, smartphone app of een webapplicatie)

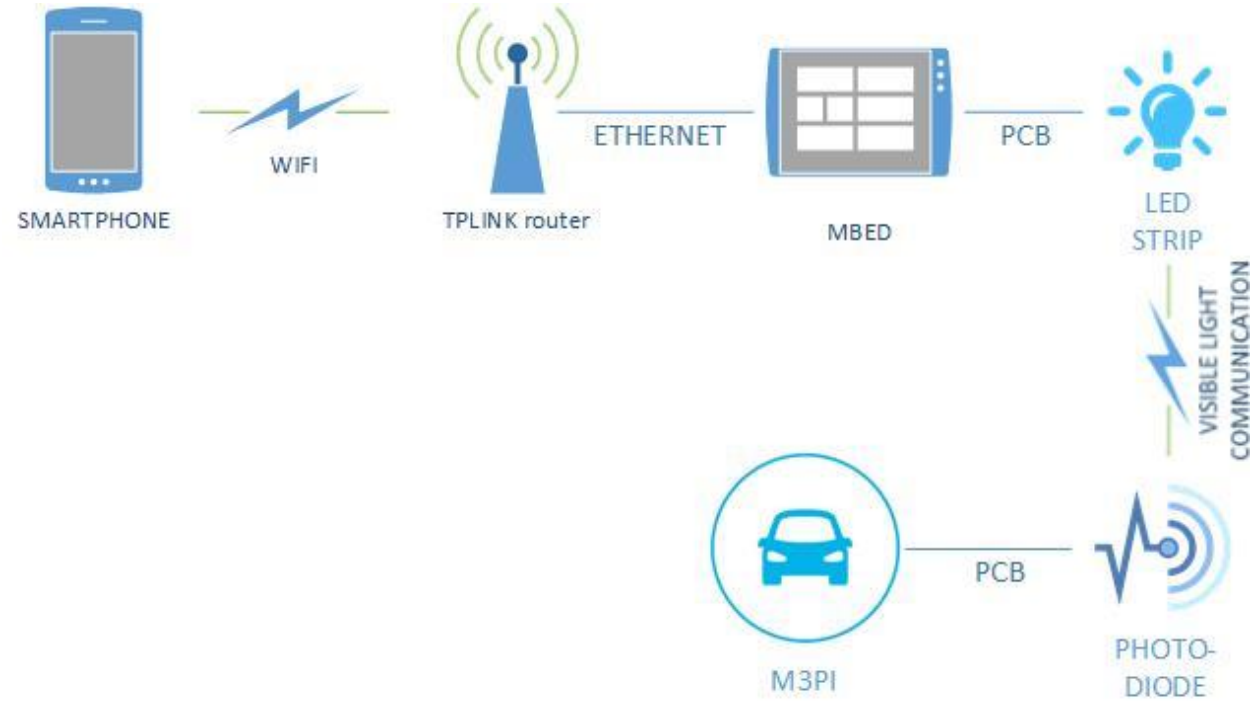
2 UITWERKING & OVERZICHT

KRISTOF T'JONCK

Na een kleine breinstormsessie hebben we ervoor gekozen om een auto te besturen met behulp van zichtbaar-licht-communicatie. De taken werden verdeeld in kleinere stukken zoals vermeld in het algemeen overzicht en tevens op github geplaatst als "issues" om deze op te volgen.

Op figuur 2-1 kan men het algemeen overzicht zien. Dit overzicht beschrijft hoe het programma in elkaar zit. Een smartphone app zendt via wifi, besturingscommando's door voor de auto die men wil besturen, via een mbed server en dit met behulp van een router. Deze data wordt vervolgens omgezet in de mbed server naar zichtbaar-licht-communicatie. Dit door middel van een ledstrip. De data wordt vervolgens door de photodiode op de mbed van de Polulu M3PI ontvangen. Deze data wordt tenslotte door diezellfde embed omzet naar de verschillende motorbesturingssignalen.

Hoofdstuk 3 bespreekt de implementatie van al deze onderdelen.



Figuur 2-1 Algemeen overzicht van het project

3 IMPLEMENTATIE

De implementatie en uitwerking van de verschillende onderdelen vermeld in het overzicht van hoofdstuk 2 worden in de volgende secties uitgelegd. Ook eventuele verbeteringen worden hierin vermeld.

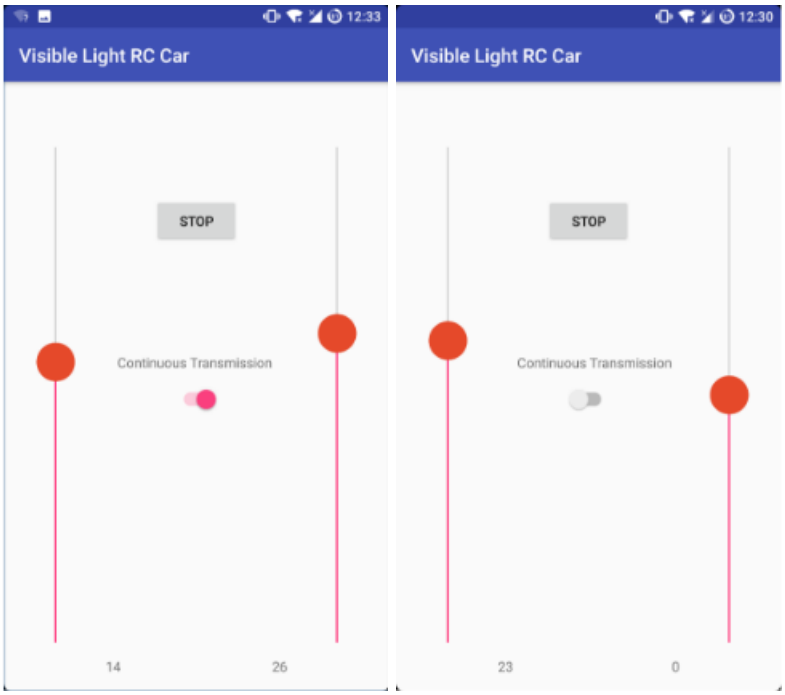
3.1 Android applicatie

Pieter Dewachter

3.1.1 De grafische interface

Zoals zichtbaar is op de figuren 3-1-1 en 3-1-2, bestaat de app hoofdzakelijk uit twee grote verticale sliders welke de aansturing van de twee motoren aangeven. Onderin wordt tekstueel aangegeven met welke waarden dit effectief overeen komt bij het versturen van de data. In het midden van het scherm kan de toggle worden teruggevonden voor het in- of uitschakelen van de continue modus van de applicatie (zie ook het volgende onderdeel).

In het midden bovenaan werd er ook nog een stop-knop voorzien, deze zorgt ervoor dat de Polulu M3PI meteen tot stilstand komt. Dit is vooral handig bij het verliezen van de controle over het stuur, iets wat af en toe wel voorkomt. Deze knop biedt een snelle oplossing waardoor eventuele schade door een botsing kan worden vermeden.



Figuur 3-1-1 en 3-1-2 Grafische interface van de Android applicatie

3.1.2 Algemene werking

De implementatie van de Androidapplicatie bevat een thread waarin de verbinden met de mbed server opgezet wordt. Het gaat hierbij om een verbinding via TCP/IP naar 192.168.0.100 via poort 4000. Niet enkel verplicht Android het gebruik van een aparte thread bij het opzetten van een TCP-socket, het zorgt er ook voor dat de grafische interface steeds responsief zal blijven. Dit komt ten goede van de eindegebruiker, welke een vloeiendere app zal ervaren.

Eerst werd er gebruikt gemaakt van twee event handlers (één voor elke slider) om de huidige waarde van de sliders bij te kunnen houden als attribuut van de hoofdklasse. Echter bleken deze handlers niet erg betrouwbaar en werden deze niet altijd correct opgeroepen. Daarom werd er gebruik gemaakt van een alternatieve implementatie die betrouwbaarder was.

De thread met de TCP-socket gaat na het verzenden van data steeds in sleep-state (Thread.sleep()), waarna deze telkens opnieuw zal starten. Via een if-structuur kan de huidige waarde van sliders opgevraagd worden en vergeleken met de vorige (die opgeslagen werd als attribuut van de hoofdklasse). Standaard zal er enkel een socket worden opgezet indien er een verandering was bij minstens één van de twee sliders, zodat dit kan worden doorgegeven.

Door de duur van de sleep te beperken (100ms), treedt er geen merkbare vertraging op. Verder zorgt dit er ook voor dat de rest van de opstelling voldoende tijd heeft om aan transmissie van data te kunnen doen, met name de communicatie die verloopt via het zichtbaar licht. Deze is namelijk een stuk trager, waardoor het belangrijk is om ervoor te zorgen dat de vorige transmissie reeds afgerond is bij het starten van een nieuwe.

3.1.3 Continue modus

Indien de gebruiker hiervoor kiest, kan er ook gekozen worden voor een continue modus. Hierbij zal er altijd om de 100ms een TCP-verbinding worden opgezet met de mbed server en data verstuurd worden. De reden hiervoor is dat de lichtcommunicatie niet altijd succesvol is. Het herhaaldelijk versturen van dezelfde pakketten zorgt ervoor dat eventueel foutief gedrag tot een minimum kan worden beperkt.

3.1.4 Gebruikte protocol voor datatransmissie

De data die verstuurd wordt via de TCP-socket is van het JSON-formaat. Dit staat voor JavaScript Object Notation en is een veelgebruikt formaat voor datatransmissie. Algemeen bestaat het uit verschillende attributen met elk een bijhorende waarde, welke allemaal leesbaar zijn voor de mens. Er werd geopteerd voor dit formaat omwille van de simpliciteit en ondersteuning, er bestaan heel wat libraries voor verschillende platformen.

Het eerste attribuut dat werd gebruikt, is het “client” attribuut. Deze bevat een uniek ID om ervoor te zorgen dat er een onderscheid kan gemaakt worden tussen verschillende Android apparaten en ook meerdere wagens. Enkel de wagen met hetzelfde ID zal reageren op een ontvangen pakket, andere wagens kunnen deze negeren.

Het tweede en derde attribuut bevatten elke een integer waarde tussen -100 en 100, welke de aansturing van de twee motoren aangeven. Er werd gekozen voor een integer formaat omdat deze minder bits in beslag neemt dan floating point notatie, de deling gebeurd pas bij het ontvangen op de Polulu M3PI.

3.1.5 Aangepaste slider klasse

Standaard worden sliders horizontaal geplaatst binnen een Androidapplicatie, dit is ongewenst voor het specifieke doel van dit project. Doordat Java echter een volledig object georiënteerde programmeertaal is, kan er gebruik gemaakt worden van overerving (inheritance) om zelf een

verticale slider klasse aan te maken. De functionaliteit die aangeboden wordt door de slider klasse wordt hiermee overgenomen, maar er kunnen wel aanpassing gebeuren waar dat nodig blijkt.

Bij een touch event wordt telkens bekeken welke waarde er moet toegekend worden aan de slider, aan de hand van enkele omzettingen om op de juiste schaal te kunnen werken. De onSizeChanged() methode van de superklasse wordt dan aangeroepen om zo een verandering van een horizontale slider na te bootsen.

Visueel ziet men echter een verticale slider door de toevoeging van een rotatie en translatie op de onDraw() methode van de superklasse. Verder werd er ook gebruik gemaakt van “custom bullets”, welke ervoor zorgen dat de bediening een stuk vlotter kan verlopen. Het grote verschil is dat deze veel groter zijn dan deze die standaard worden aangeboden door Android. De implementatie hiervan gebeurde door het aanmaken van een xml-bestand die de bullet definieert, hierbij waren slechts een minimaal aantal regels code nodig.

3.1.6 Mogelijke verbeteringen

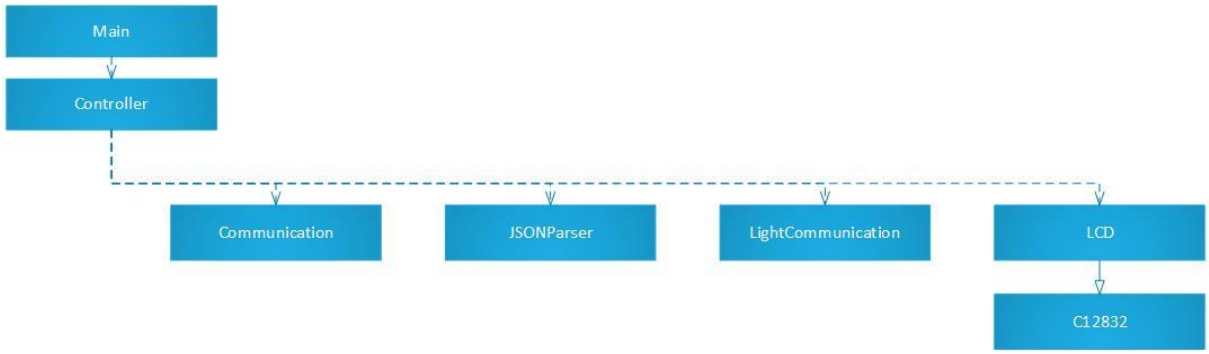
Er werd een “client” attribuut voorzien in de JSON-pakketten die worden verstuurd, maar deze worden steeds ingesteld op 0. In een latere versie zou het mogelijk zijn om de gebruiker een ID te laten kiezen welke overeenkomt met een ID van een Polulu M3PI. Op die manier is het ook daadwerkelijk mogelijk om meerdere auto’s te laten bedienen door meerdere personen, om op die manier te kunnen racen tegen elkaar. Hiervoor werd al een stuk code geïmplementeerd, maar er zijn nog enkele aanpassingen nodig om dit volledig werkend te maken.

Een andere verbetering heeft betrekking tot de continue modus van de app, deze zou verwijderd kunnen worden in een later versie. Het zou beter zijn om de functionaliteit te implementeren op de mbed server in plaats van de Androidapplicatie, want nu wordt er telkens bandbreedte verspild om dezelfde data te blijven sturen. TCP heeft reeds een ingebouwd systeem dat zorgt voor gegarandeerde levering van data, dus dat is geen reden om het te implementeren in de Androidapplicatie.

Belangrijker nog is echter dat er minder energie zal gebruikt worden door het Android apparaat, want er hoeft niet telkens een TCP-socket te worden opgezet. Aangezien batterijduur een belangrijk aspect is van de huidige smartphones, is dit een niet te overzien voordeel. De mbed server kan gevoed worden via het net (USB-adapter) aangezien er reeds stroom nodig is om de LED strip te kunnen aansturen.

3.2 mbed (server) ***Kristof T'Jonck***

Het programma op de mbed zorgt ervoor dat de data van de android applicatie via een TCP Server ontvangen kan worden, deze data wordt vervolgens omgezet om te versturen via lichtcommunicatie naar de polulu M3PI. Op figuur 3-1 zien we een algemeen overzicht van hoe het klassendiagram van deze applicatie er uit ziet.



Figuur 3-2-2 Klassendiagram van de mbed server

3.2.1 Controller

De controller klasse zorgt voor de samenhang van het gehele programma. De run functie van deze klasse is de hoofdmethode die het gehele programma aanstuurt in de "main".

Via de "Communication" klasse wordt data ontvangen die vervolgens verwerkt wordt in de "JSONParser", dat op zijn beurt dan verzonden wordt via de "LightCommunication" klasse.

3.2.2 Communication

De communication klasse zorgt voor een ethernetverbinding via de "EthernetInterface" package. Om zo dan via een socket informatie van een client kan verkrijgen.

3.2.2.1 Connectie met ethernet

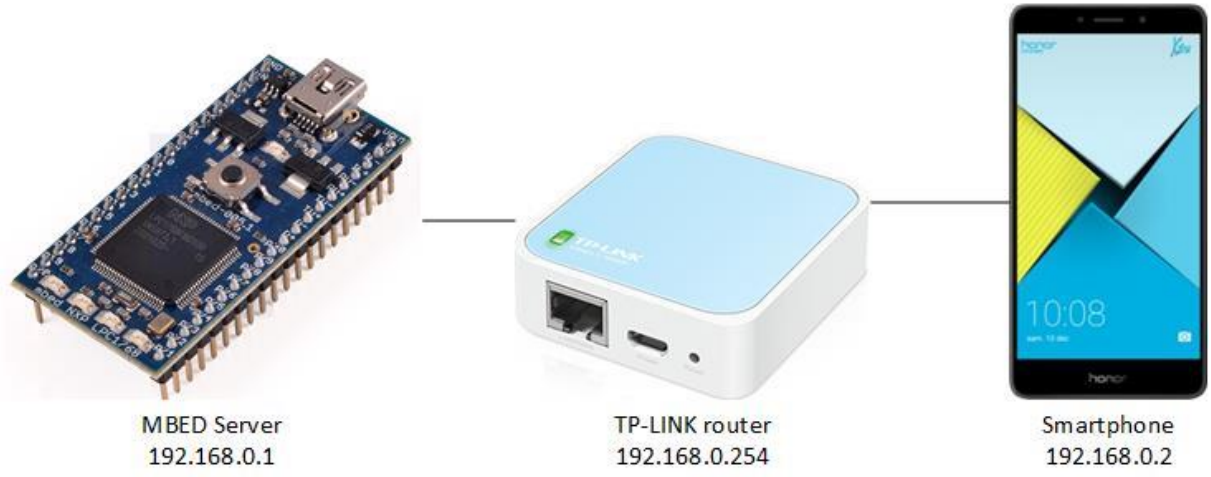
De ethernetinterface zal als volgt geconfigureerd worden:

- IP-address: 192.168.0.1
- Subnetmask: 255.255.255.0
- Default gateway 192.168.0.1

Dit wil zeggen dat de Android-applicatie ook een IP-adres moet hebben in dezelfde range. Dus een IP-adres dat tussen 192.168.0.2 en 192.168.0.255 gelegen ligt. We stellen zelf onze IP-adressen statisch in of laten die van de android applicatie bepalen door de DHCP-server in de router. De router dat we eerst gebruikten maakte gebruik van 192.168.0.x adressen. Deze router was de TP-LINK WR702N, deze maakte gebruik van het IP-adres 192.168.0.254. Deze

gaf dus automatisch IP-adressen in het juiste bereik. Een overzicht van hoe de IP's verdeeld zijn is te zien op figuur 3-2.

Aangezien de verbinding met deze router wegviel hebben we een andere router gebruikt. Deze gebruikte 192.168.1.x als lokale adressen. De DHCP-server zal dus geen adressen geven in de juiste range en dus moeten we alles statisch instellen. Door op de Smartphone dus een statisch IP-adres in te stellen, alsook op de mbed, kan er dus een communicatie voorzien worden.



Figuur 3-2-3 Overzicht IP-adressen van de verschillende apparaten

3.2.2.2 Verbinden met een socket

Deze verbinding wordt gemaakt om zo via TCP/IP betrouwbaar data te krijgen van de Android-applicatie. We gebruiken TCP i.p.v. UDP omdat we geen merkbaar verschil zouden zien in snelheid. Via TCP/IP zenden we namelijk veel sneller dan dat we via het licht zenden.

De "EthernetInterface" library bevat methodes waarmee een TCP-socket server opgezet kan worden. De server zal luisteren op poort 4000 om zo inkomende data van de Androidapplicatie op te vangen. Dit door een met een client via een "TCPSocketConnection" te connecteren als er data komt. Het ontvangen is volledig "blocking", de mbed zal dus wachten tot er data binnen gekomen is vooraleer er ontvangen wordt.

3.2.3 JSONParser

De binnengekomen data zal in de vorm van een JSON(JavaScript Object Notation) String zijn. De data van de client, left en right worden uit de JSON-string gehaald via een parser en vervolgens in een char array geplaatst met een check byte.

3.2.3.1 Parsen van de data

De data moet uit de json string gehaald worden, hiervoor gebruiken we de "Picojson" library. Deze zal een error geven als geen correcte string ingelezen is om zo fouten te voorkomen. Als een foute string doorgekomen is zal dit pakket simpelweg weggegooid worden. Een voorbeeld van hoe de JSON string er uit kan zien is volgende: $\{\text{"}client\text{"}= 1,\ \text{"}left\text{"}=37, \text{"}right\text{"} = -15\}$

Deze data moet worden omgezet naar een array van int8_t waarbij we een zo'n klein mogelijk aantal bytes willen doorsturen. De data moet namelijk via het licht doorgestuurd worden, hoe langer de data hoe langer het zal duren om door te sturen (zie 3.2.5.2). De data zal dus 4 bytes lang zijn. Één byte voor de client, één voor left, één voor right en één voor de checkbyte. Left en right zijn waarden van -100 tot 100, aangezien een int8_t alle waardes kan bevatten tussen -128 tot 127, is dit ideaal om deze data in op te slaan met zo weinig mogelijk bits. In tabel 3-2-1 staat een overzicht over hoe de byte array eruit ziet.

Tabel 3-2-1 Overzicht van de byte array

int8_t[0]	int8_t[1]	int8_t[2]	int8_t[3]
client	left	right	check byte

3.2.3.2 Berekenen checkbyte

De checkbyte wordt berekend met de volgende formule:

$$Checkbyte = (left + right + 127) \,\%\, 251$$

Dit zorgt ervoor dat er kan gecheckt worden als de left en right byte correct zijn. 251 is het grootste priemgetal onder de 255, dus onder de maximale waarde van 8 bits. De 127 is het priemgetal in de helft hiervan. Priemgetallen worden vaak gebruikt in cryptografie om een betere security te garanderen.

We hadden voor deze simpele methode gebruikt aangezien er maar 2 bytes in rekening gebracht moesten worden. Echter bij nader inzien was deze methode niet zo goed. (left + right) kan namelijk soms gelijk uitkomen bij verschillende getallen waardoor de checkbyte vaker collisions zal geven bij verschillende getallen. Deze komen niet veel voor maar er zijn betere alternatieven om een checkbyte te creëren.

Tabel 3-2-2 Voorbeeld pakket voor lichtcommunicatie

0x01	0x37	0xF1	0x10
1	55	-15	16

Omdat de checkbyte methode niet goed was, is de checkbyte veranderd in een CRC-6 checksum met polynoom: $x\text{^}6 + x\text{^}4 + x + 1$, deze zal een betere error detectie geven. Deze methode wordt namelijk vaak gebruikt voor error detectie in digitale netwerken. In ons voorbeeld zal de checksum 0x10 geven. In de volgende tabel 3-2-2 zien we hoe het uiteindelijke pakket er dan uit ziet voor ons voorbeeld. Meer info over de berekening staat in de cpp file op github.[1]

[1]https://github.com/GimoHD/Project2/blob/master/project2_server/JSONParser.cpp

3.2.4 LCD

De LCD klasse wordt enkel gebruikt voor testdoeleinden. Bij het initialiseren van de server wordt er data naar de LCD geprint waardoor kan gezien worden als de server goed gestart is.

3.2.5 Lightcommunication

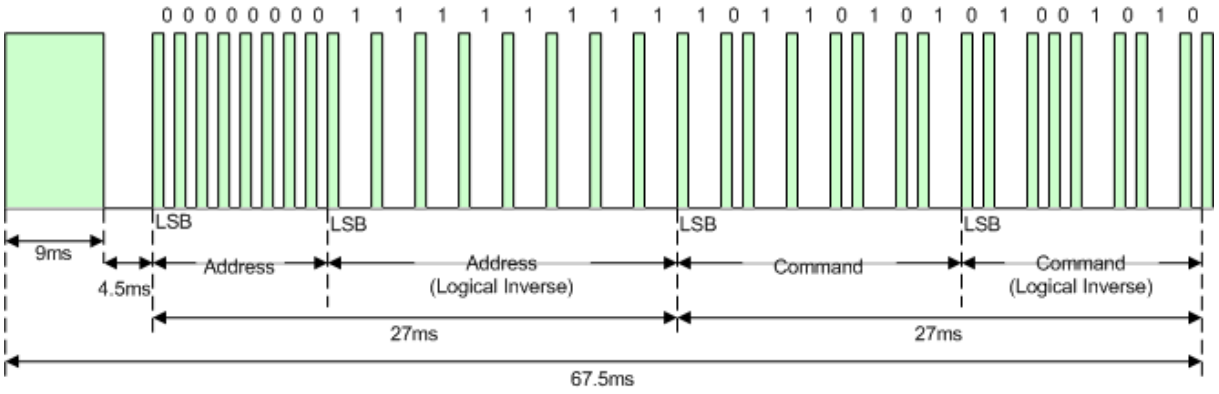
Twee methodes van lichtcommunicatie zijn uitgewerkt:

- NEC
- Serieel

Aangezien de NEC-package moeilijk aan te passen was, en dat bij het testen van de data er enorme flikkeringen waren in het licht hebben we besloten een tweede manier te gebruiken. Beide methodes zijn geïmplementeerd in de source code. Het serieel verzenden wordt momenteel gebruikt.

3.2.5.1 Verzenden via NEC

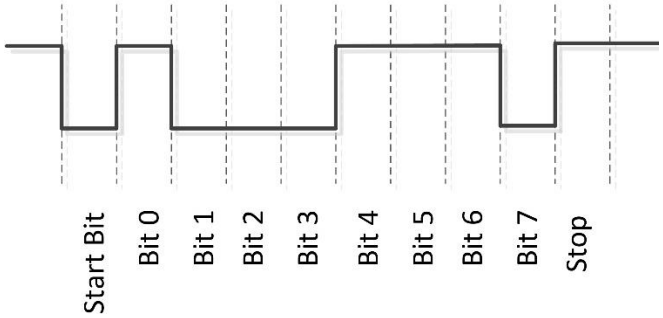
De “Lightcommunication” klasse zorgt voor het verzenden van de data via licht communicatie. De eerste methode dat we gebruikten om te verzenden was via NEC. Figuur 3-2-3 geeft een voorbeeld hoe een NEC-pakket eruitziet. Om dit te verzenden gebruiken we de “RemoteIR” package, deze wordt normaal gebruikt voor het verzenden via infrarood. Het signaal wordt op een draaggolf van 38kHz geplaatst zoals vermeld is in de package. (Nakamura, 2010)



Figuur 3-2-4 Voorbeeld van een NEC-pakket

3.2.5.2 Serieel verzenden (UART)

De tweede methode is de momenteel gebruikte code. Serieel wordt verzonden via de “Serial” klasse die in de mbed library inbegrepen zit. We hebben gekozen voor een baud rate van 9600, we zien hier nog flikkeringen in het licht maar deze zijn veel minder dan bij het verzenden via NEC. Bij een baud rate van 9600 zal de duur per bit $\frac{1}{9600} = 104\mu s$ zijn.



Figuur 3-2-5 UART data frame

Aangezien er per byte een stop en startbit is zijn er dus 10 bits per karakter, dit kan men ook zien op figuur 3-2-4. Aangezien er vier bytes zijn zullen er dus 40 bits doorgestuurd moeten worden.

$40\ bits\ *\ 104\ \mu s\ =\ 4160\ \mu s\ of\ dus\ 4.12\ ms\ per\ 4\ databytes$

Bij een verhoging van baud rate daalt het aantal bits die correct ontvangen wordt door de receiver, 9600 was ideaal om mee door te sturen.

3.2.6 Verbeteringen

3.2.6.1 Foutcorrectie

Momenteel werken we enkel met een foutdetectie om te kijken als de data correct doorgestuurd werd. Met een errorcorrectie kan data verwerkt worden zodat er een foute bit bit gerecupereerd kan worden. Er moeten echter enkele parity bits bij komen waardoor het frame langer wordt maar de check byte kan in principe weggelaten worden. Met een encodering zoals de Hamming encodering kan door het bijvoegen van enkele pariteit bits de fouten in sommige bits gedetecteerd worden en als er een enkele fout is kan deze ook gecorrigeerd worden.

3.2.6.2 Wegwerken zichtbare flikkeringen

Bij het versturen van data is er nog veel flikkeringen omdat we per 100ms doorsturen. Momenteel is bij ons het aantal nullen en eentjes variabel omdat we puur via uart doorzenden waardoor we telkens met ons oog een flikkering zien. Aan de hand van een codering (bv. een manchester encoding) kan een gemiddeld dc-level gecreëerd worden die zal gelijk zijn aan de helft van de maximale amplitude. (dit zien we met het oog als een licht op halve sterkte) Het licht wordt namelijk maar met een gemiddeld half maximaal vermogen weergegeven bij het signaal van manchester encoding. Het aantal nullen en ééntjes zal namelijk gelijk zijn over die bepaalde tijdsduur.

Als we dan bij het niet sturen van een signaal een DC- level sturen dat de helft is van de maximale amplitude zou deze flikkering nog amper te zien mogen zijn. Dit hoeft niet softwarematig te zijn, hardware matig is dit ook te verwezenlijken.

3.3 Lichtcommunicatie (zender) ***Pascal Barbary***

3.3.1 Probleemstelling

- De uitgangsspanning van de embed is slechts 3.3V en zakt met een ratio van 0.4V/4mA. De MOSFET zal dus bij een vrij lage gatespanning moeten kunnen schakelen.
- Daarnaast dient te worden nagegaan of de uitgangsstroom voldoende groot is om de ingangscapaciteit C_{iss} te laden binnen de schakel frequentie van onze toepassing.
- Wat zijn de schakelverliezen en liggen deze binnen de waarden van een natuurlijke koeling van de halfgeleider.

3.3.2 Gegevens

Een LED-strip met een maximum vermogen van 40W en een schakelfrequentie van maximum 100kHz, als rekenvoorbeeld.

$$P_{load} = 40W\,; U = 12V => I = \frac{40W}{12V} = 3.35A\,;\; f_{sw} = 100kHz;\; I_{GPIOmax} = 4mA$$

Na wat zoekwerk zijn er twee geschikte MOSFET's gevonden, de IRLB8721PbF en de IRLZ44-SiHLZ44. Om de berekeningen uit te voeren hanteren we volgende waarden uit hun datasheet (International Rectifier, n.d.) (Vishay, n.d.). De waarde van de ingangsimpedantie moet met wat voorzichtigheid worden aangenomen. De gatespanning is immers wat lager waardoor de ingangsweerstand beslist wat groter zal zijn.

IRLB8721PbF	IRLZ44-SiHLZ44
$R_{DS}(ON) = 0.016\Omega\, met\, V_{GS} = 4.5V$	$R_{DS}(ON) = 0.039\Omega\, met\, V_{GS} = 4V$
$V_{DS\,max} = 30V$	$V_{DS\,max} = 60V$
$V_{off} = 12V$	$V_{off} = 12V$
$t_{sw\,ON} = 9.1 + 93 = 102.1ns$	$t_{sw\,ON} = 17 + 230 = 247ns$
$t_{sw\,OFF} = 9 + 17 = 26ns$	$t_{sw\,OFF} = 42 + 110 = 152ns$
$C_{iss} = 1.077nF$	$C_{iss} = 3.3nF$
$C_{oss} = 0.36nF$	$C_{oss} = 1.2nF$

3.3.3 Formules

3.3.3.1 Berekening van de gate-ingangsstroom

De stijg- en daaltijd van de ingangs-gate is afhankelijk van de stroom die kan worden geleverd door de GPIO-pin van de microcontroller! Men dient er zich dus van te vergwissen of deze stijg - en daaltijd mogelijk is binnen een periode van het ingangssignaal.

$$I_{gate} = C_{iss} * \frac{dU}{dt} \implies I_{gate} \int dt = C_{iss} * U \implies t = \frac{C_{iss} * U}{I_{GPIOmax}}\; met\; t = t_{rise}\; of\; t_{fall}$$

3.3.3.2 Verliezen MOSFET

De schakelverliezen van een MOSFET zijn afhankelijk van enerzijds de ingangsweerstand- als anderzijds de capacitieve verliezen.

- ***Het weerstandsverlies***

$$P_{con} = I_{on}^2 * R_{DS_{on}}$$

- ***De verliezen t.g.v. de parasitaire capaciteit***

$$P_{sw1} = \frac{t_{sw_{on}} * V_{off} * I_{on} * f_{sw} + t_{sw_{off}} * V_{off} * I_{on} * f_{sw}}{2}$$

$$P_{sw2} = C_{oss} * V_{off}^2 * f_{sw}$$

- ***Totaal verliezen en rendement***

In onderstaande formules wordt ervan uitgegaan dat het weerstandsverlies in verhouding even groot is als het schakelverlies. Doch dit is beslist meestal niet zo, en al zeker niet in hoe de code hier werd geïmplementeerd. De code-loze ON-tijd van 100ms weegt immers veel zwaarder door dan die van het schakelen.

$$P_{loss} = P_{con} + P_{sw1} + P_{sw2} \quad \& \quad P_{tot} = P_{loss} + P_{load}$$

$$\eta = \frac{P_{load}}{P_{tot}}$$

3.3.4 Berekeningen

IRLB8721PbF

$t = \frac{1.077nF * 3.3V}{4mA} = \mathbf{0.9\mu s} \ll \boldsymbol{T} = \mathbf{10\mu s}$

$P_{cond} = 3.35^2\text{I} * 0.016\Omega = \mathbf{0.18W}$

$P_{sw1} = \mathbf{0.257W}$

$P_{sw2} = 0.36\text{nF*12²V*100kHz} = \mathbf{0.005W}$

$P_{loss} = 0.18W + 0.257W + 0.005W =$**0.442W**

$P_{tot} = 0.442W + 40W =$**40.442W**

$\eta = \frac{40W}{40.442W} * 100 = \mathbf{98.9\%}$

IRLZ44-SiHLZ44

$I_{gate} = \frac{3.3nF * 3.3V}{4mA} = 3\mu s < \boldsymbol{T} = \mathbf{10\mu s}$

$P_{cond} = 3.35^2 I * 0.039\Omega = 0.44W$

$P_{sw1} = 0.802W$

$P_{sw2} = 1.2\text{nF*12²V*100kHz} =0.017\text{W}$

$P_{loss} = 0.44W + 0.802W + 0.017W =1.259\text{W}$

$P_{tot} = 1.259W + 40W =41.259\text{W}$

$\eta = \frac{40W}{41.259W} * 100 = 96.9\%$

3.3.5 Keuze van MOSFET

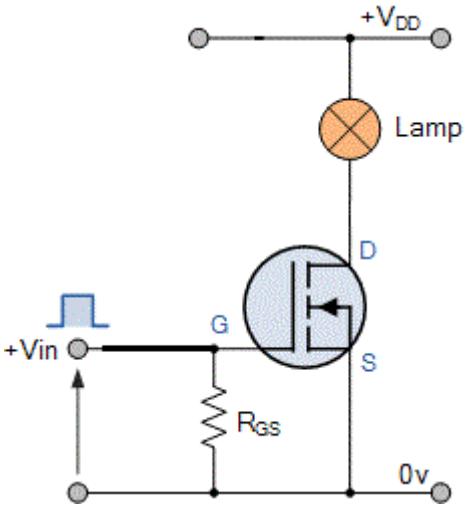
Op alle punten blijkt de IRLB8721PbF (figuur 3-3-1) de betere keuze te zijn. De schakelfrequentie is ruim laag genoeg opdat de stijg- en daaltijd van een flank geen obstakel vormt binnen een signaalperiode. Met een gedissipeerd vermogen van slechts 0.442W kan de MOSFET gedurende een langere tijd zonder bijkomende koeling werken.

Figuur 3-3-1 de MOSFET

3.3.6 De schakeling

De schakeling (figuur 3-3-2) is vrij eenvoudig van opzet.

De weerstand verzekert dat de gate-ingang steeds laag is indien deze niet wordt aangestuurd.

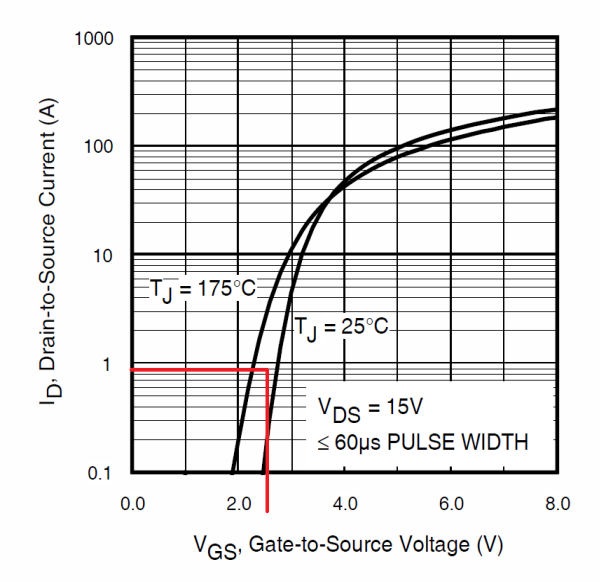


Figuur 3-3-2 de schakeling zonder Rin

Deze schakeling heeft als nadeel dat er geen enkele vorm van restrictie is opdat de gate van de MOSFET meer dan de toegelaten stroom van een GPIO-pin zou ‘sourcen’ dan wel ‘sinken’.

Mocht de mbed niet zo goed zijn beveiligd dan zou de microcontroller beslist beschadigd kunnen worden. Een microcontroller is meestal gevoeliger aan stroom terugvloei dan aan stroomvraag. Deze stroomrestrictie is eenvoudig toe te voegen door een weerstand in serie te plaatsen met de gate.

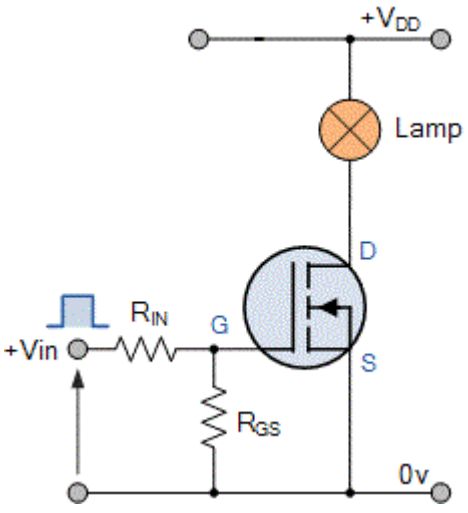
Om deze waarde te bepalen dient men te kijken naar stroom die de verbruiker zal consumeren via de source van de MOSFET. Anderzijds dient men de gatespanning, in het geval van de IRLB8721PbF, af te leiden uit het I_{drain} versus I_{gate} diagram.



Figuur 3-3-3 I_D/V_{GS}-diagram

In het geval van de uitgevoerde schakeling in het labo was de stroom van de LED-strip slechts 900mA. De gate-spanning waarbij deze door de MOSFET wordt doorgelaten is slechts +/-2.6V. Bij een controlemeting werd deze waarde bij het opmeten ook vastgesteld.

De restrictie-weerstand R_{in} zal dus het verschil van 3.3V-2.6V moeten opvangen. Met een GPIO-stroom van 4mA komt dit neer op een weerstandswaarde van 175Ω. Voor R_{gate} kunnen we een waarde nemen in verhouding met de gekozen gatespanning 650Ω.



Figuur 3-3-4 de schakeling met restrictie 1

3.4 Lichtcommunicatie (Ontvanger) ***Pieter Berteloot***

Het doel van de ontvanger is om het lichtsignaal om te zetten in een bruikbaar, digitaal signaal. Dit digitaal signaal wordt dan gebruikt om de Polulu aan te sturen.

3.4.1 Algemeen

We kunnen de ontvanger opdelen in 5 grote blokken:

- Signaal ontvangen
- Filteren
- Versterken
- Omzetten naar een digitaal signaal
- Leesbaar maken voor de mbed

3.4.2 Signaal ontvangen

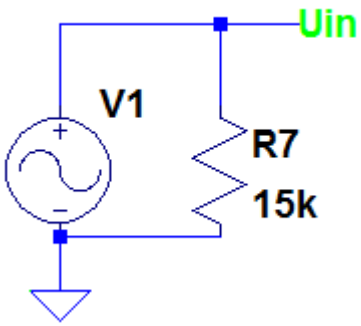
3.4.2.1 Keuze sensor

Om een goed signaal te detecteren hebben we een sensor nodig met een bandbreedte voor zichtbaar licht en een snelle responsie tijd om hoge frequenties op de vangen. Wij hebben gekozen voor de VTB8440BH fotodiode.

Figuur 3-4-1 VTB8440BH

3.4.2.2 Schakeling

We gebruiken de lichtsensor in fotovoltaïsche mode zoals te zien in figuur 3-4-1. De spanning over de weerstand is afhankelijk van de lichtsterkte die de sensor ontvangt.

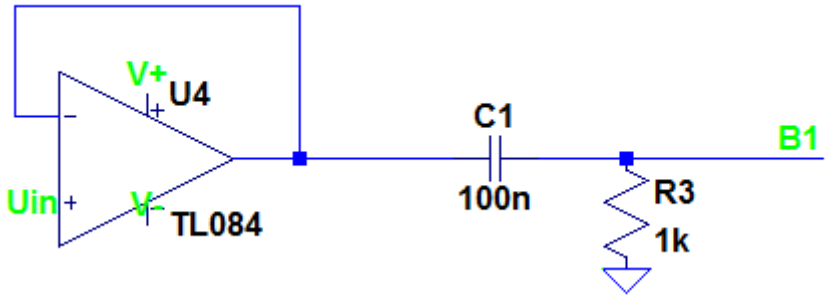


Figuur 5-2-2 fotovoltaïsche mode

3.4.3 Filteren

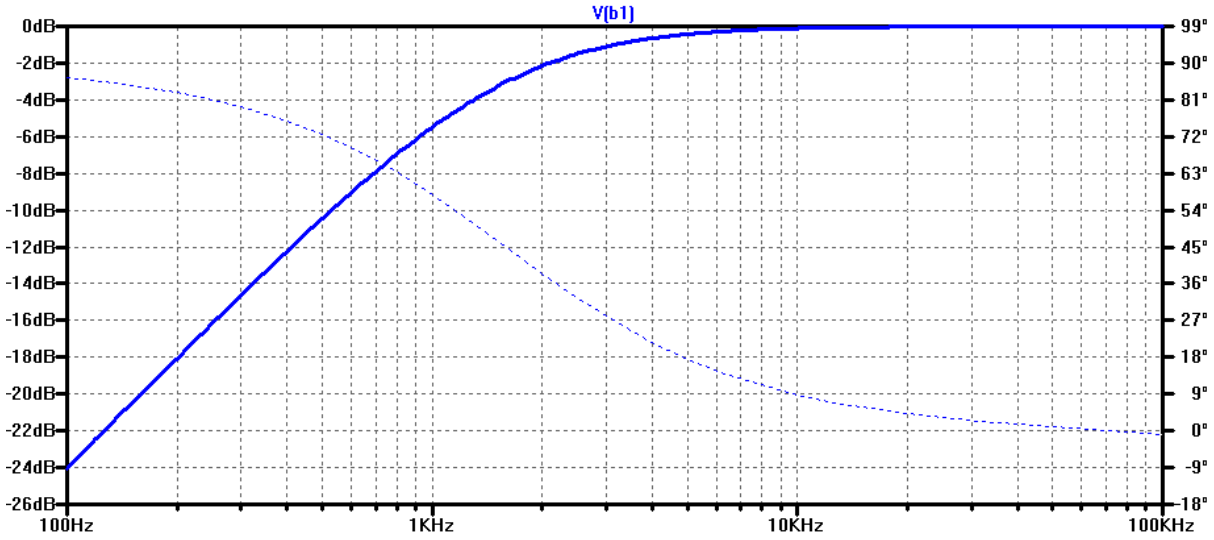
Aangezien we met een frequentie rond de 10kHz verzenden moeten de frequenties onder de 10kHz (zoals daglicht, lampen,...) weg gefilterd worden. Hiervoor maken we gebruiken van een hoogdoorlaat filter.

3.4.3.1 Schakeling



Figuur 3-4-2 buffer met hoog doorlaat filter

Omdat we enkel het spanningsniveau nodig hebben van de sensor brengen het signaal eerst naar een buffer. Deze zorgt er ook voor dat de sensor geen andere belastingen heeft en dus ook geen stroom moet genereren. Na de buffer wordt het signaal gefilterd met een CR filter met figuur 3-4-3 als bodediagram:



Figuur 3-4-3 Bodediagram hoog doorlaat filter

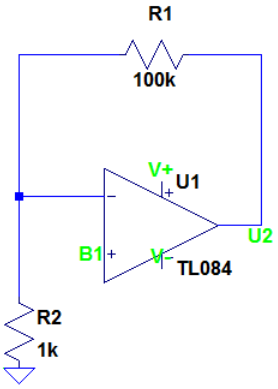
In figuur 3-4-3 zien we dat de frequenties onder 9kHz-10kHz worden onderdrukt. Het DC component van de filter wordt dus zo goed als volledig weg gefilterd en enkel het nodige signaal blijft over.

3.4.4 Versterken

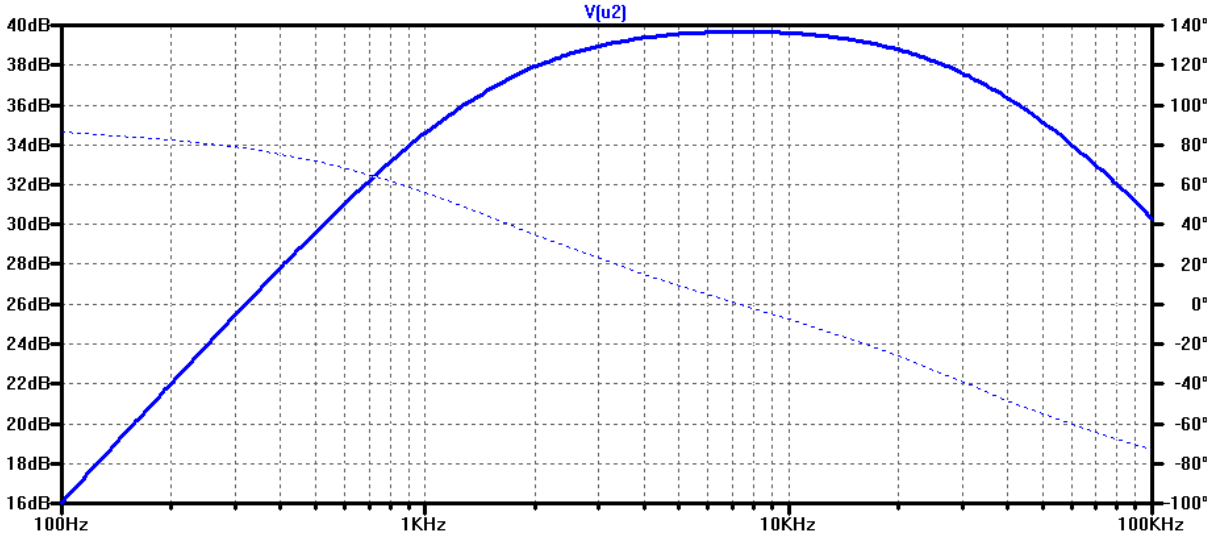
De spanning na de HDF is een grootorde van mV's. Om dit signaal bruikbaar te maken verstreken we dit met een niet-inverterende versterker.

3.4.4.1 Schakeling

R1 = 100kΩ
R2 = 1kΩ
Versterking: $1 + \frac{100k\Omega}{1k\Omega}$ = 101



Figuur 3-4-4 Niet-inverterende versterker



Figuur 3-4-5 Bodeplot niet-inverterende versterker

We zien duidelijk in figuur 3-4-5 dat de frequenties rond 10kHz worden volledig versterkt.

3.4.5 Omzetten naar een digitaal signaal

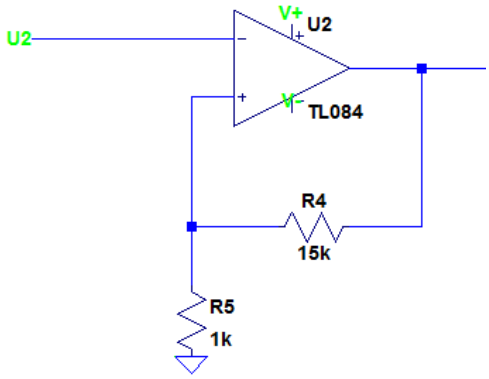
Na de versterker zijn er geen steile flanken (door de rise- en fall time van de sensor) dus maken we gebruik van een Schmitt trigger om deze te creëren.

3.4.5.1 Schakeling

R_4= 15kΩ
R_5 = 1kΩ

V_+ = 0,56V
V_- = -0,56V

Aangezien we met een grote versterking werken bij de niet-inverterende versterker is de waarde van 0.56V en -0.56V snel bereikt waardoor we een snelle reactie krijgen op de uitgang van de Schmitt trigger.



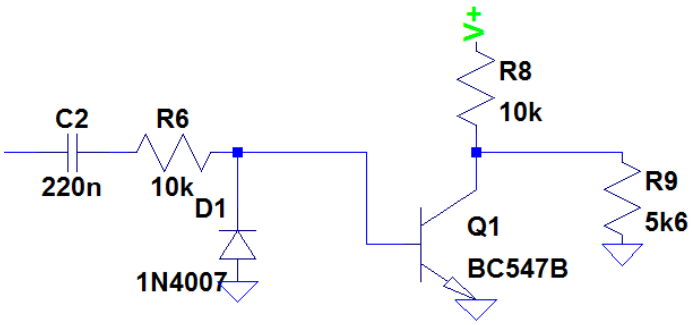
Figuur 3-4-6 Schmitt trigger

3.4.6 Leesbaar maken voor de mbed

Na de schmitt trigger krijgen we $\pm 9V$. Aangezien de mbed op een digitale ingang maar 0 of 3,3V kan lezen moeten we deze waarden nog omzetten.

3.4.6.1 Schakeling

Wanneer de uitgang +9V wordt de transistor in geleiding gebracht waardoor de uitgang laag getrokken wordt (0v). Als de uitgang -9V wordt spert de transistor en komt er 3,3V op de uitgang.



Figuur 3-4-7 transistor schakeling

3.4.7 Mogelijke verbeteringen

3.4.7.1 Meerdere sensoren

Het gebruik van meerdere sensoren (bijvoorbeeld onder een verschillende hoek) kunnen het bereik van ontvangst vergroten. Ook aan weerskanten sensoren plaatsen zodat wanneer de connectie op één van de sensoren wegvalt, men weet dat de wagen bijna uit bereik komt.

3.4.7.2 Asymmetrisch voeden

Nu hebben we twee 9V batterijen nodig om ±9V te verkrijgen om de opamps te voeden. Het zou beter zijn om deze asymmetrisch te voeden bijvoorbeeld tussen 0 en 9V zodat we enkel één 9V batterij nodig hebben.

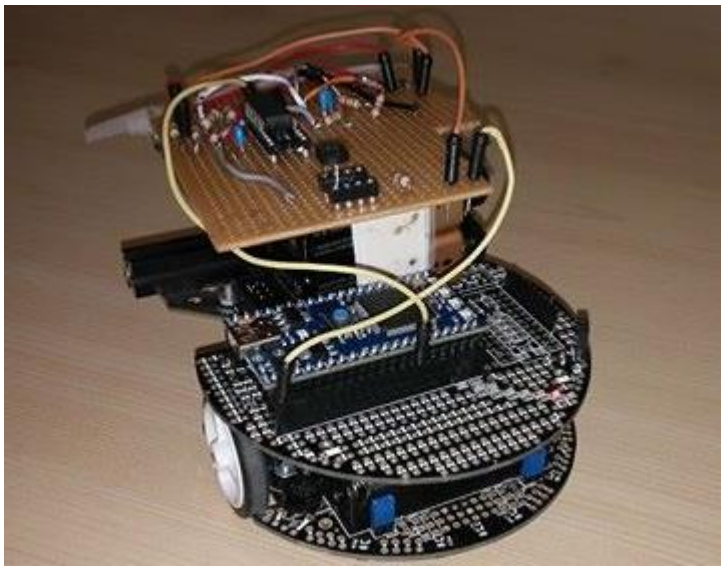
3.4.7.3 Andere sensor met betere rise- en fall time

Met de VTB8440BH zijn we beperkt tot 10kHz, als we hoger gaan kunnen we het signaal niet meer omzetten naar een bruikbaar signaal. Met een andere sensor die sneller detecteert zou dit wel mogelijk zijn.

3.5 mbed Polulu M3PI

Kristof T'Jonck

De data die via het licht ontvangen wordt moet opgevangen worden door de mbed op de M3PI maar er moeten ook tevens motoren aangestuurd worden. Een foto van de M3PI is afgebeeld op figuur 3-5-1. Om dit te doen zijn er 2 thread die constant lopen. In de volgende secties worden deze threads uitgelegd.



Figuur 3-5-6 Foto van de polulu M3PI

3.5.1 Thread voor de aansturing van motoren

De eerste thread dient om de M3PI aan te sturen. Dit door het aansturen van de linker- en rechtermotor. Via de m3pi package is het mogelijk om deze motoren aan te sturen met een waarde van -1.0 float tot 1.0 float. Dit via de waarden die we in de andere thread ontvangen via het licht. Aangezien we met gedeelde variabelen werken wordt er gebruik gemaakt van mutexen bij het inlezen van de variabelen.

3.5.2 Thread voor licht ontvangst

De data die via het licht werd doorgezonden moet worden opgevangen. Deze data is serieel of via NEC doorgezonden. Beide methoden van ontvangen zijn geïmplementeerd maar momenteel wordt de seriële transmissie gebruikt. De bytes die de linker- en rechtermotor bepalen zijn integers, deze moeten omgezet worden naar een float. Een volgende formule wordt voor de conversie naar een float:

$$\left(\frac{value}{100}\right) * SPEED = float\ value$$

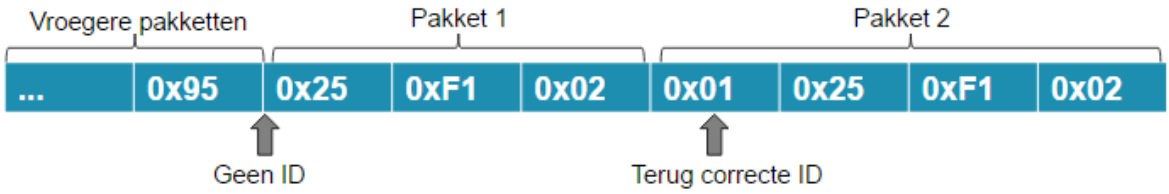
Hierin is de "value" de linker en rechtermotor snelheid als integer. Door deze te delen door 100 zal deze een float van de gewenste grootte worden. We vermenigvuldigen deze nog met een speed modifier, deze zorgt ervoor dat de snelheid beperkt wordt. Ons test vlak is namelijk een klein oppervlak van een tafel waarop gewerkt moet worden, de snelheid mag dus niet te hoog zijn zodat hij niet van tafel rijdt.

De verkregen data moet worden gecheckt op fouten. Als er zich fouten voordoen moet het pakket weggegooid worden. Er zijn 2 checks die gebeuren:

- Checken op ID
- Checken van de checkbyte

De data wordt byte per byte verwerkt. Als het ontvangen ID overeenkomt met het ID van de Polulu M3PI dan zullen de volgende 3 bytes ingelezen worden. Waarna we dus 4 bytes gekregen hebben. Uit de eerste 3 bytes wordt de CRC-6 checksum bepaald en vergeleken met de 4^{e} byte. Als deze klopt is de data correct. De serial interface werkt met een buffer zodat karakter per karakter uit deze buffer kan uitgelezen worden.

Figuur 3-5-2 geeft een voorbeeld van hoe data ontvangen kan worden. Als de ID bijvoorbeeld niet leesbaar toegekomen is zullen er dus maar 3 karakters ontvangen zijn in de buffer. Daarna zal er karakter per karakter gekeken worden tot er terug een correct ID is, in dit voorbeeld dus 0x01. Als deze klopt wordt dus van de eerste 3 bytes de CRC-checksum bepaald (bv. crc van 0x0125F1 = 0x02). Deze zal vergeleken worden met de 4^{e} byte. Als deze gelijk zijn is de data dus correct doorgestuurd en wordt deze als rechtse en linkse variabele opgeslagen. Deze zijn dezelfde variabelen die gebruikt worden in de thread voor het aansturen. Ook hier wordt dus een mutex gebruikt.



Figuur 3-5-7 Voorbeeld van ontvangen data

3.5.3 Verbeteringen

3.5.3.1 Error detectie/correctie

Zoals in 3.2.6.2 vermeld kan door Hamming encoding een error detectie doen van meerdere bits, en ook een enkele bitfout kan gecorrigeerd worden. Hierdoor wordt de dataoverdracht verbeterd.

3.5.3.2 Energiebesparing

Momenteel werken de threads simultaan naast elkaar en blijven runnen. In principe moet er enkel een thread runnen bij ontvangst van data waarna deze verwerkt moet worden in de auto. Dit door een eventuele interrupt bij ontvangst in de seriële interface.

4 RESULTATEN

KRISTOF T'JONCK

De data wordt correct van de Androidapplicatie naar de mbed-server verzonden. Die data wordt op zijn beurt dan correct omgezet en verzonden via licht communicatie. Men kan nog zien als de data verstuurd wordt door een flikkering in het licht, die flikkering zou nog weggewerkt moeten kunnen worden. De data wordt meestal correct ontvangen bij de auto.

Op deze manier werkten alle onderdelen van ons project dus allemaal samen naar behoren. In zijn geheel kon men dus de auto besturen aan de hand van de Androidapplicatie.

5 VERBETERINGEN

KRISTOF T'JONCK

Om het project te verbeteren kunnen verschillende aanpassingen en aanvullingen gebeuren. Bij de secties in hoofdstuk 3 werd telkens vermeld waar er verbeteringen of aanvullingen kon gebeuren. Deze waren vooral voor het verbeteren van:

- Energieverbruik
- Snelheid
- Foutcorrectie
- Gebruiksvriendelijkheid

Ook zou het rijden met meerdere wagens mogelijk moeten zijn. Echter moet voor elke wagen een PCB voorzien worden.

6 CONCLUSIE

KRISTOF T'JONCK

We kunnen concluderen dat dit een geslaagd project was.

Samenwerken in groep is niet altijd even simpel, soms moet de een wachten op een stuk van de ander om te kunnen testen. Maar door elkaar te helpen met problemen van een ander komt er vaak een beter eindresultaat.

Vaak is het ook zo dat problemen maar gemerkt worden bij het testen van de onderdelen samen, terwijl ze apart wel werkten. De vraag is meestal als het aan de software of aan de hardware ligt. Zo was het vaak dat bij het testen de hardware meestal op de software moest ingesteld worden en andersom.

Github is ook niet makkelijk te gebruiken als er code op bijvoorbeeld mbed te schrijven. Deze werkt namelijk met een ander versiebeheer systeem namelijk via Mercurial. Deze exporten naar een andere IDE om daar te compileren was niet mogelijk, het bewerken in een IDE en daarna pushen naar de repository was wel mogelijk. Er moest dus telkens naar beiden gepushed worden.

REFERENTIES

(n.d.). Retrieved from http://www.mouser.com/ds/2/196/irlb8721pbf-938011.pdf

International Rectifier. (n.d.). *dgd*. Retrieved from Infinion: http://www.infineon.com/dgdl/irlr8721pbf.pdf?fileId=5546d462533600a40153566e27f326ec

Nakamura, S. (2010, Augustus 17). *RemoteIR*. Retrieved from MBED: https://developer.mbed.org/users/shintamainjp/code/RemoteIR/

Vishay. (n.d.). *docs*. Retrieved from vishay: http://www.vishay.com/docs/91328/91328.pdf