

FACULTEIT INDUSTRIELE

INGENIEURSWETENSCHAPPEN

**TECHNOLOGIECAMPUS OOSTENDE**

Academiejaar 2016-2017

Projectlab bachelor electronica-ICT

Besturen van een auto met behulp van zichtbaar licht communicatie

Pieter BERTELOOT

Pascal BARBARY

Pieter DEWACHTER

Kristof T’JONCK

Inhoud

[Inhoud ii](#_Toc483174805)

[1 Algemeen 1](#_Toc483174806)

[1.1 Doel 1](#_Toc483174807)

[1.2 Uitwerking & overzicht 1](#_Toc483174808)

[2 Android applicatie 2](#_Toc483174809)

[2.1 Screenshots, uitleg, verbeteringen mogelijk?, Custom slider klasse & custom bullet voor size 2](#_Toc483174810)

[3 MBED (server) 3](#_Toc483174811)

[3.1 Controller 3](#_Toc483174812)

[3.2 Communication 3](#_Toc483174813)

[3.2.1 Connectie met ethernet 3](#_Toc483174814)

[3.2.2 Verbinden met een socket 4](#_Toc483174815)

[3.3 JSONParser 4](#_Toc483174816)

[3.3.1 Parsen van de data 4](#_Toc483174817)

[3.3.2 Berekenen checkbyte 5](#_Toc483174818)

[3.4 LCD 5](#_Toc483174819)

[3.5 Lightcommunication 6](#_Toc483174820)

[3.5.1 Verzenden via NEC 6](#_Toc483174821)

[3.5.2 Serieel verzenden (UART) 6](#_Toc483174822)

[4 Lichtcommunicatie (zender) 8](#_Toc483174823)

[4.1 Keuzes componenten, schema’s, scoopbeelden, problemen & eventuele verbeteringen 8](#_Toc483174824)

[5 Lichtcommunicatie (Ontvanger) 9](#_Toc483174825)

[5.1 Keuzes componenten, schema’s, scoopbeelden, problemen & eventuele verbeteringen 9](#_Toc483174826)

[6 MBED Polulu M3PI 10](#_Toc483174827)

[6.1 Thread voor het aansturing van motoren 10](#_Toc483174828)

[6.2 Thread voor licht ontvangst 10](#_Toc483174829)

[7 Algemeen besluit 12](#_Toc483174830)

[7.1 12](#_Toc483174831)

[Referenties 13](#_Toc483174832)

[Bijlagen 14](#_Toc483174833)

[Bijlage A Detailtekeningen van de proefopstelling 1](#_Toc483174834)

# Algemeen Kristof T’Jonck

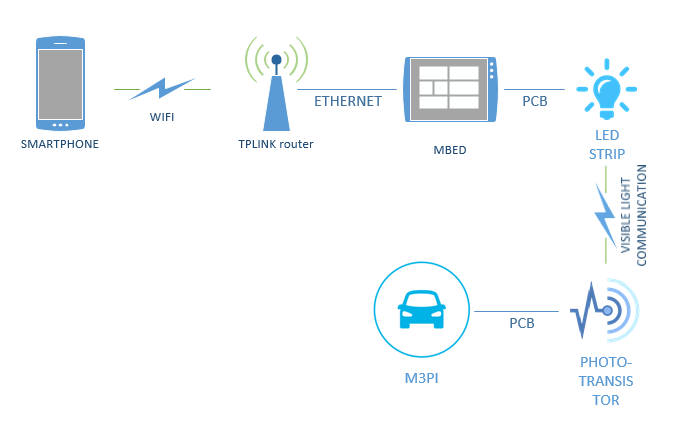
## Doel

Een applicatie maken dat digitale communicatie met zichtbaar licht demonstreert. Deze applicatie moet minimum 1 microcontroller hebben, communicatie tussen een stand-alone, smartphone app of een webapplicatie op een PC/smartphone en de microcontroller.

## Uitwerking & overzicht

Wij hebben gekozen om een auto te besturen met behulp van de zichtbaar licht communicatie. Op figuur 1-1 kunnen we een algemeen overzicht zien over hoe het programma in elkaar zit. Een smartphone app zendt via wifi commando’s door die de auto moeten besturen naar een MBED server via wifi en een router. Deze data wordt vervolgens omgezet in de MBED server en verstuurt deze data via een led strip met licht communicatie. Via een phototransistor wordt de data op de MBED van de Polulu M3PI ontvangen. Deze zal de data dan gebruiken om zijn motoren aan te sturen.

Dit verslag bespreekt hoe deze onderdelen opgebouwd zijn.



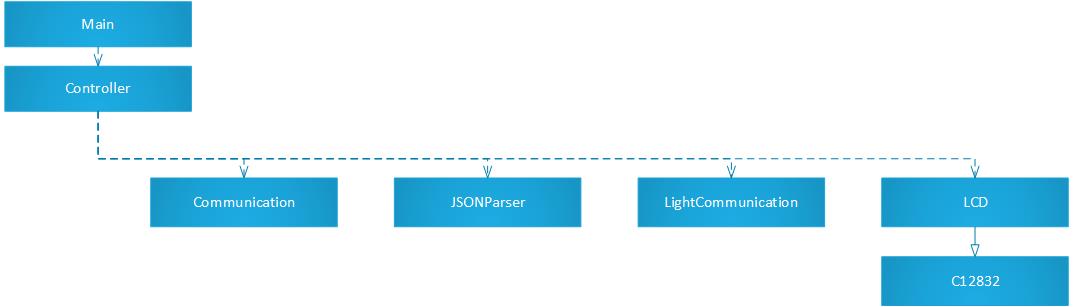
Figuur ‑ Algemeen overzicht van het project

# Android applicatie

## Screenshots, uitleg, verbeteringen mogelijk?, Custom slider klasse & custom bullet voor size

# MBED (server) Kristof T’Jonck

Een programma op de MBED zorgt er voor 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 dit er uit ziet.



Figuur 3‑‑ Klassendiagram van de MBED server

## Controller

De controller klasse zorgt voor de samenhang van het gehele programma. De run functie van deze klasse zal eigenlijk de hoofdmethode zijn die het gehele programma bepaalt 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.

## Communication

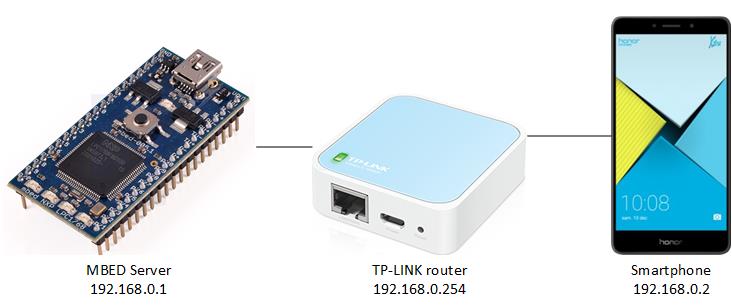
De communication klasse zorgt voor een ethernetverbinding via de “EthernetInterface” package. Om zo dan via een socket informatie van een client kan verkrijgen.

### Connectie met ethernet

Deze verbinding wordt gemaakt om zo via TCP/IP data te krijgen van de Android-applicatie. De ethernetinterface zal als volgt geconfigureerd worden:

* IP address: 192.168.0.1
* Subnet mask: 255.255.255.0
* Default gateway 192.168.0.1

Dit wil zeggen dat de Android-applicatie ook een ip address moet hebben in dezelfde range. Dus een IP adres dat tussen 192.168.0.2 en 192.168.0.255 gelegen ligt. Aangezien we gebruik maken van een eigen router die niet aan het net hangt moeten we deze statisch instellen. 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. Door op de Smartphone een statisch IP adres in te stellen, alsook op de mbed, kan er dus een communicatie voorzien worden. 

Figuur 3‑‑ Overzicht IP adressen van de verschillende apparaten

### Verbinden met een socket

De “EthernetInterface” library bevat tevens ook methodes waarmee een socket server opgezet kan worden. De server zal luisteren op poort 4000 om zo inkomende data van de Android applicatie op te vangen. Dit door een client via een TCPSocketConnection te ontvangen als er data komt. Het ontvangen is volledig “blocking”, de MBED zal dus wachten tot er data binnen gekomen is vooraleer er ontvangen wordt.

## 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.

### 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:

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 4.5.2). De data zal dus 4 bytes lang zijn. Éen 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 3-1 staat een oversicht over hoe de byte array er uit ziet.

|  |  |  |  |
| --- | --- | --- | --- |
| **int8\_t[0]** | **int8\_t[1]** | **int8\_t[2]** | **int8\_t[3]** |
| client | left | right | check byte |

Tabel 3‑ Overzicht van de byte array

### Berekenen checkbyte

De checkbyte wordt berekend met de volgende formule:

Dit zorgt er voor dat er kan gechecked worden als de left en right 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **0x01** | **0x37** | **0xF1** | **0x02** |
| 1 | 37 | -15 | 2 |

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 dan ook gelijk kan zijn. Dit komt niet veel voor maar er zijn betere alternatieven.

Table 3‑ Voorbeeld pakket voor lichtcommunicatie

Omdat de checkbyte methode niet goed was, is de checkbyte veranderd in een CRC-6 checksum, 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 0x02 geven. In de volgende tabel 3-2 zien we hoe het uiteindelijke pakket er uit ziet voor ons voorbeeld. Meer info over de calculatie staat in de cpp file op github.[[1]](#footnote-1)

## 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.

## Lightcommunication

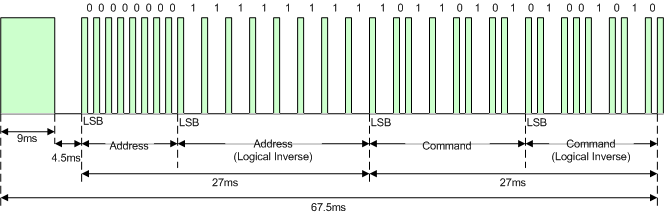
Twee methodes van lichtcommunicatie zijn uitgewerkt:

* 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.

### Verzenden via NEC

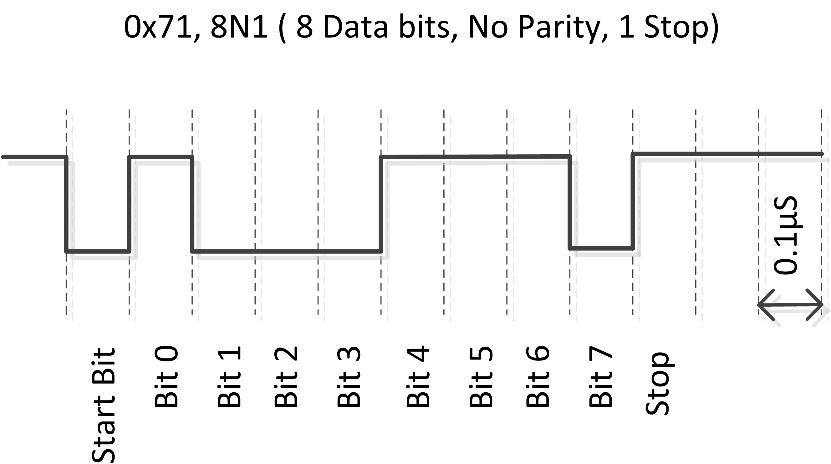
De Lightcommunication klasse zorgt voor het verzenden van de data via licht communicatie. De eerste methode dat we gebruikten om te verzenden was via NEC. Figuur 3-3 geeft een voorbeeld hoe een NEC-pakket er uit ziet. Om dit te verzenden gebruiken we de “RemoteIR” package, deze wordt normaal gebruikt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



Figuur 3‑‑ Voorbeeld van een NEC-pakket

### 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 zijn.



Figuur 3‑‑ 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 3-4. Aangezien er vier bytes zijn zullen er dus 40 bits doorgestuurd moeten worden.

Bij een verhoging van baud rate daalt het aantal correct verkregen bits, 9600 was ideaal om mee door te sturen.

# Lichtcommunicatie (zender)

## Keuzes componenten, schema’s, scoopbeelden, problemen & eventuele verbeteringen

# Lichtcommunicatie (Ontvanger)

## Keuzes componenten, schema’s, scoopbeelden, problemen & eventuele verbeteringen

# 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 6-1. Om dit te doen zijn er 2 thread die constant lopen. In de volgende secties worden deze threads uitgelegd.



Figure 6‑‑ Foto van de polulu M3PI

## Thread voor het aansturing van motoren

De eerste thread dient om de de M3PI aan te sturen. Dit door het aansturen van de linker- en rechtermotor. Via de m3pi package is het mogelijk om deze 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.

## 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 rechter motor bepalen zijn integers, deze moeten omgezet worden naar een float. Een volgende formule wordt voor de conversie naar een float:

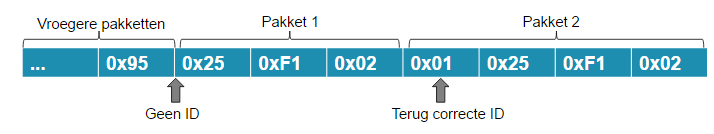
Hierin is de “value” de linker en rechtermotor snelheid als integer. Door deze te delen door 100 zal deze een float worden. We vermenigvuldigen deze nog met een speed modifier, deze zorgt er voor dat de snelheid beperkt wordt. Ons test vlak is namelijk een klein oppervlak waarop gewerkt moet worden, 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 4e 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 6-2 geeft een voorbeeld van hoe data ontvangen kan worden. Als de ID bijvoorbeeld niet leesbaar toegekomen is zullen er dus maar 3 karakters zitten in de buffer vanuit dit pakket. 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 crc(0x0125F1) = 0x02. Deze zal vergeleken worden met de 4e byte. Als deze klopt is de data dus correct en wordt deze als rechtse en linkse variabele opgeslaan. Deze zijn dezelfde variabelen die gebruikt worden in de thread voor het aansturen. Ook hier wordt dus een mutex gebruikt.



Figuur 6‑‑ Voorbeeld van ontvangen data

# Algemeen besluit

## 

Referenties

*Hier komt de volledige referentielijst in de gekozen stijl APA of IEEE.*

Bijlagen

Bijlagen worden bij voorkeur enkel elektronisch ter beschikking gesteld. Indien essentieel kunnen in overleg met de promotor bijlagen in de scriptie opgenomen worden of als apart boekdeel voorzien worden.

Er wordt wel steeds een lijst met vermelding van alle bijlagen opgenomen in de scriptie. Bijlagen worden genummerd het een drukletter A, B, C, …

Bijlage A Detailtekeningen van de proefopstelling

Bijlage B Meetgegevens (op USB)

1. Detailtekeningen van de proefopstelling

1. https://github.com/GimoHD/Project2/blob/master/project2\_server/JSONParser.cpp [↑](#footnote-ref-1)