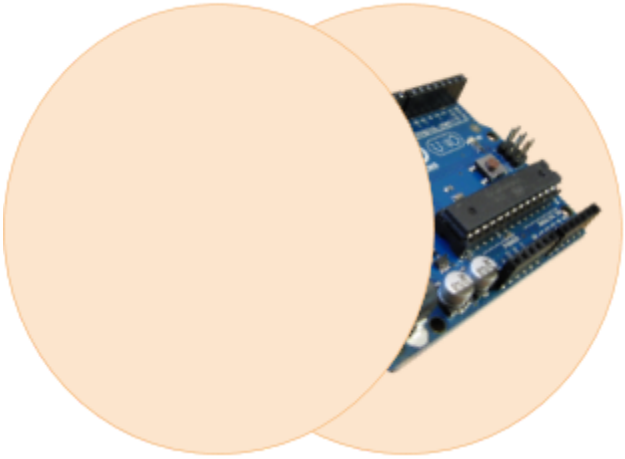
Technische Ontwerp Document

Project Ass



Auteurs : Robin van Wijk, Roy Voetman, Shaquille Louisa en Joey Marthé Behrens

Opdrachtgever : Zeng Ltd.

Documentnummer : TO.0001

Versienummer : 1.0.0

Status : Done

Documentdatum : 28-10-2019

I. Revisiehistorie

Versie	Datum	Omschrijving	Auteurs
0.0.1	28-10-2019	Start Technische ontwerp (layout, inhoud, informatie)	Robin van Wijk, Roy Voetman, Shaquille Louisa en Joey Behrens
1.0.0	31-10-2019	Eind realisatie van document	Robin van Wijk, Roy Voetman, Shaquille Louisa en Joey Behrens

Inhoudsopgave

1 Inleiding

Het technische ontwerp is een document dat de technische eisen en implementaties van het eindproduct van een project beschrijft. In dit document wordt voor elk onderdeel van het domotica project dat Zeng ltd. heeft opgedragen technisch uitgewerkt en worden er randvoorwaarden gesteld. Het doel is een duidelijk inzichtelijk te hebben hoe alles gerealiseerd moet worden op een technische niveau.

Dit technische ontwerp zal alleen informatie bevatten over alle randvoorwaarden en eisen daarnaast wordt er gedocumenteerd hoe alles op een technische niveau gerealiseerd wordt. Dit document speelt in op het eerder geschreven plan van aanpak dat dieper inging op de structuur van het project en de omgeving waarin het project zal worden uitgevoerd. Alle eisen die gesteld zijn in het functioneel ontwerp zullen hier ook op een technische niveau besproken worden. Aangezien Zeng ltd. de opdrachtgever is van het project zal er worden gewerkt vanuit een bestaand concept dat verwerkt is in het technische ontwerp.

Zoals voorheen genoemd bestaat de documentatie voor dit project uit meerdere documenten.Elk document heeft een specifiek doel dat het wilt bereiken en bieden samen een solide basis waar vanuit gewerkt kan worden. Dit document is de laatste in de documenten reeks:

- Het Plan van Aanpak
- Het Functioneel Ontwerp
- Het Technisch Ontwerp (dit document)

Alle documentatie voor dit project is terug te vinden op de Google Drive die is aangemaakt door de projectgroep.

Een kort overzicht van het technische ontwerp ziet er als volgt uit: In hoofdstuk 2 worden de technische vereisten aan het licht gebracht. Daarna gaan we in hoofdstuk 3 in op het protocol dat gebruikt zal worden voor seriële communicatie. Informatie over de werkomgeving van de centrale en bestuurseenheden zijn te vinden in hoofdstuk 4. Hoofdstuk 5 zal vervolgens alle interfaces van het FO technische uitwerken. Hoofdstuk 6 gaat in op de technische realisatie van de bestuurseenheid en tenslotte is er in hoofdstuk 7 een conventie lijst te vinden voor code standaardisering.

2 Technische eisen

2.1 De Centrale

2.1.1 Snelheid

De snelheid van de applicatie is afhankelijk van veel verschillende factoren zoals bijvoorbeeld device hardware. Het is dus lastig een bepaalde tijd te garanderen voor alle hardware combinaties. Er wordt echter gestreefd om user input binnen 100ms af te handelen aangezien alles boven dit aantal resulteert in haperingen.

2.1.2 Hardware

De Centrale moet werken op standaard laptop hardware zodat er door alle developers aan gewerkt kan worden. Om een minimum te stellen aan de hardware vereisten is erg lastig omdat het moeilijk is om de exacte ervaring te bepalen tijdens het uitvoeren van Python. Het volgende wordt echter aanbevolen door de Python Community voor lightweight GUI applicaties: Single Core 1.0 Ghz, 64Mb Videokaart, 1Gb Ram, met een grafisch besturingssysteem zoals Lubuntu (Lightweight Ubuntu)

Daarnaast dient de hardware stack de mogelijkheid te bieden voor seriële communicatie via UART. Dit is namelijk vereist om te communiceren met de Arduino volgens het protocol beschreven in hoofdstuk 3.

2.1.3 Afhankelijkheid

De Centrale dient onafhankelijk te werken. Als er geen devices zijn aangesloten en er volgens een wordt aangesloten dient deze herkent te worden. Hetzelfde geldt andersom als er een device is aangesloten en deze wordt ontkoppeld dient de applicatie te blijven werken.

2.2 Bestuurseenheden

2.2.1 Autonomie

De voornaamste eis van de besturingseenheden is dat zij autonoom kunnen werken. Dit houdt in dat zij zonder invloed van een persoon zelf kunnen bepalen wanneer zij open en dicht gaan. Dit zorgt ervoor dat de eigenaar de rolluiken één keer kan instellen en er daarna niet meer naar om hoeft te kijken. Ook als de verbinding zou verbreken met het dashboard dan zullen de rolluiken nog steeds functioneel zijn.

2.2.2 Communicatie

De besturingseenheden moeten in te stellen zijn via het dashboard. Dit houdt in dat de maximale en minimale threshold moet kunnen worden ingesteld. Daarnaast moeten de besturingseenheden uitgelezen kunnen worden via het dashboard. Dit gaat via hetzelfde protocol. De werking van dit protocol is de vinden in hoofdstuk “3 UART communicatie protocol” De besturingseenheden moeten naast een ultrasone sensor ook een temperatuur of een licht sensor bevatten. Dit houdt dus ook in dat de code generiek geschreven moet worden, en dat tijdens het uploaden van de firmware er aangegeven moet worden welke sensor er gebruikt wordt.

3 UART communicatie protocol

3.1 Serial High Integrity Transmission

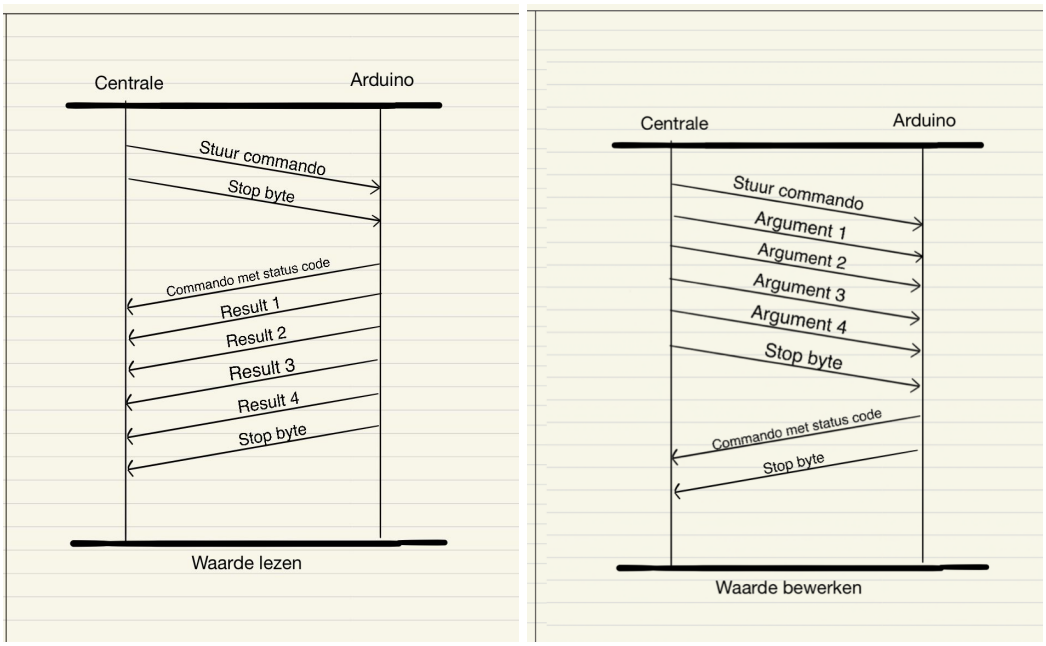
Voor dit project is er een custom communicatieprotocol ontwikkeld. Serial High Integrity Transmission, in het vervolg gerefereerd als SHIT. SHIT is een command based protocol. Er zal eerst een commando verstuurd worden. Elk commando is 5 bits lang met nog 3 bits voor een status code. De status code is alleen van toepassing als het command in uitgevoerd. Als een status code niet van toepassing is zullen deze 3 bits allemaal als “0” worden verstuurd. De combinatie van commando en status code is in zijn geheeld 1 byte.

Elk commando heeft een verwacht aantal argumenten, elk argument is 1 byte in grote. De argumenten dienen verstuurd te worden na het commando zelf. Tot slot wordt er altijd afgesloten met een stop byte. Voor een alle command en status codes wordt gerefereerd naar *3.4 Data organisatie.*

Omdat het commando en de status code in 1 en dezelfde byte zijn verwerkt kan dit protocol aan beide kanten hetzelfde worden geïmplementeerd. Als er geen status code is meegeven moet het commando uitgevoerd worden. Als er wel een status code is meegeven kunnen we het interpreteren als het resultaat van het commando.

3.1.1 Diagrammen

Onderstaand staan de protocol diagrammen die visueel toelichten wat er in "*3.1 Werking van het protocol"* is besproken.



Figuur 3.1.1 *Protocol diagram voor het lezen en bewerken van een waarde.*

3.2 Frame format

Het volgende frame formaat zal aangehouden worden:

Asynchronous, 8 data bits, 1 stop bit, no parity

Als we vervolgens de datasheet van de **Atmega328P MCU** raadplegen om te onderzoeken hoe we dit frame format moeten instellen komen we uit op het volgende Control en Status Register (**UCSRnC**):

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

Figuur 3.2.1 *USART Control en Status Register n C (UCSRnC).*

De volgende subkoppen zullen verder ingaan welke specifieke bits de waarde “1” zullen moeten bevatten. Naast het controle en status register **UCSRnC** is er ook het **UCSRnB** en **UCSRnA** register**.** In dit geval hebben we deze register echter niet nodig om het hierboven genoemde frame format in te stellen.

We hebben het **UCSRnB** register echter wel nodig om de transmitter en receiver modus beide in te schakelen. Hiervoor moeten de bits **TXEN0** en **RXEN0** de waarde “1” bevatten.

Code sample:

```
UCSRnB = _BV(TXEN0) | _BV(RXEN0);
```

3.2.1 Asynchronous

De bits **UMSELn1** en **UMSELn0** in **UCSRnC** selecteren de “mode of operation. Voor dit frame format willen we de mode “Asynchronous USART” instellen. Volgens *figuur 3.2.2* moeten beide bits de waarde bevatten “0”. Aangezien dit volgens *figuur 3.2.1* de initiële waardes zijn is er geen additionele handeling nodig.

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

Figuur 3.2.2 *UMSELn Bits Settings.*

Code sample:

```
UCSRnC = _BV(UCSZn1) | _BV(UCSZn0);
```

3.2.2 Aantal data bits

De bits: **UCSZn1** en **UCSZn0** in **UCSRnC** en de bit **UCSZn2** in **UCSRnB** definieren tesamen het aantal data bits. *Figuur 3.2.3* laat zien dat de bits **UCSZn1** en **UCSZn0** de waarde “1” moeten bevatten om 8 data bits in te stellen.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Figuur 3.2.3 *UCSZn Bits Settings.*

3.2.3 Stop bit(s)

De **USBSn** bit in **UCSRnC** controleert of er 1 or 2 stop bits gebruikt worden. Voor dit frame format willen we de 1-bit instellen. Volgens *figuur 3.2.1* is dit de initiële waarde van **USBSn** dus er geen additionele handeling nodig.

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Figuur 3.2.4 *USBS Bit Settings.*

3.2.4 No parity

De bits **UPMn1** en **UPMn0** in **UCSRnC** selecteren de “Parity Mode”. Voor dit frame format willen we parity mode uitschakelen. Volgens *figuur 3.2.4* moeten beide bits de waarde bevatten “0”. Aangezien dit volgens *figuur 3.2.1* de initiële waardes zijn is er geen additionele handeling nodig.

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Figuur 3.2.5. *UPMn Bits Settings.*

3.3 Baudrate instellingen

De baudrate zal ingesteld worden op **19.2K** omdat dit één van de meeste gebruikte snelheden is voor asynchrone operaties. Deze baudrate kan ook worden gegenereerd met behulp van de UBRRn-instellingen van de Atmega328P zie *figuur 3.2.5*.

Baud Rate (bps)	f_{osc} = 16.0000MHz				f_{osc} = 18.4320MHz				f_{osc} = 20.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max. (1)	1Mbps		2Mbps		1.152Mbps		2.304Mbps		1.25Mbps		2.5Mbps	

Figuur 3.2.5 *Voorbeelden van UBRRn-instellingen voor veelgebruikte oscillatorfrequenties.*

De Arduino Uno beschikt over een oscillator frequentie van 16 MHz (**f_{osc} = 16 Mhz**). Er zal **geen** gebruik gemaakt worden van de **U2X mode**. Volgens *figuur 3.2.5* moet dan aan **UBBRn** de waarde **51** worden toegekend om een baudrate van **19.2K** te configureren.

Code sample:

```
UBRRnL = 51;
```

3.4 Data organisatie

De commando's worden verstuurd in packets van 8 bits. In *Figuur 3.3.1* wordt uitgelegd hoe de commando packets worden opgebouwd en welke functies deze hebben.

B7	B6	B5	B4	B3	B2	B1	B0	Functie	Omschrijving
1	1	1	1	1	1	1	1	Stop	Einde van een packet sequentie
0	-	-	-	-	-	-	-	Functie	Lees een waarde
1	-	-	-	-	-	-	-		Schrijf een waarde
-	0	0	-	-	-	-	-	Modus	Huidige waarde
-	0	1	-	-	-	-	-		Drempelwaarde - Min
-	1	0	-	-	-	-	-		Drempelwaarde - Max
-	0	0	0	0	-	-	-	ID	Status
-	-	-	0	1	-	-	-		Afstand
-	-	-	1	0	-	-	-		Temperatuur/licht
-	0	0	1	1	-	-	-		UUID/Type

Figuur 3.3.1 *Seriële communicatie commando's.*

3.4 Packet sequence

Om een commando te kunnen versturen is er een bepaalde volgorde van packets nodig. Als eerste moet er een packet verstuurd worden met het commando zelf. Deze packet bevat een commando lezen of schrijven, de waarde die gelezen of geschreven moet worden en welke modus er gebruikt moet worden. Nadat het schrijf commando is verstuurd volgen alle packets met de waarde en wordt deze afgesloten met een stop packet. In *Figuur 3.4.1* en *Figuur 3.4.2* wordt weergegeven hoe de volgorde van de packets wordt opgebouwd.

1	**2..n-1**	**n**
Commando	**Waarde** (0..n)	**Stop**

Figuur 3.4.1 *Sequentie voor een schrijf commando.*

1	**2**
Commando	**Stop**

Figuur 3.4.2 *Sequentie voor een lees commando.*

3.5 Packet validatie

Wanneer er een bepaalde sequentie van packets verstuurd wordt zullen deze eerst gevalideerd worden. Wanneer er een commando verstuurd wordt zal de ontvangende kant wachten tot er een stop packet verstuurd wordt. Daarna zal de eerste packet gecontroleerd worden op welk commando er verstuurd wordt, daarna welke modus je wilt gebruiken en als laatst welke waarde er gebruikt moet worden. Alle commando's hebben een verwacht aantal packets die verstuurd worden. Voor een floating point waarde is dit namelijk het commando packet, de 4 packets voor de waarde zelf en als laatst de stop packet.

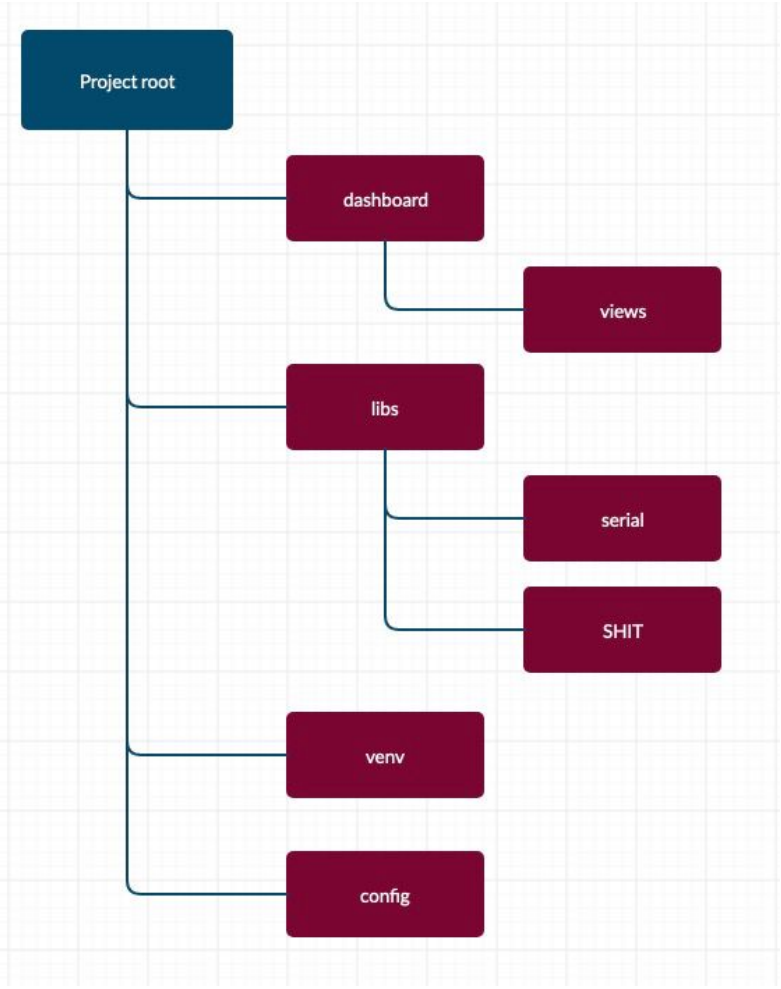
B7	B6	B5	B4	B3	B2	B1	B0	Functie	Omschrijving
-	-	-	-	-	0	0	0	Error	Goed
-	-	-	-	-	0	0	1		Fout
-	-	-	-	-	0	1	1		Data verlies
-	-	-	-	-	1	0	1		Onverwachte hoeveelheid packets
-	-	-	-	-	1	1	1		Invalid command

Figuur 3.5.1 *Seriële communicatie error commando's.*

4 Werkomgeving

4.1 Bestandsstructuur

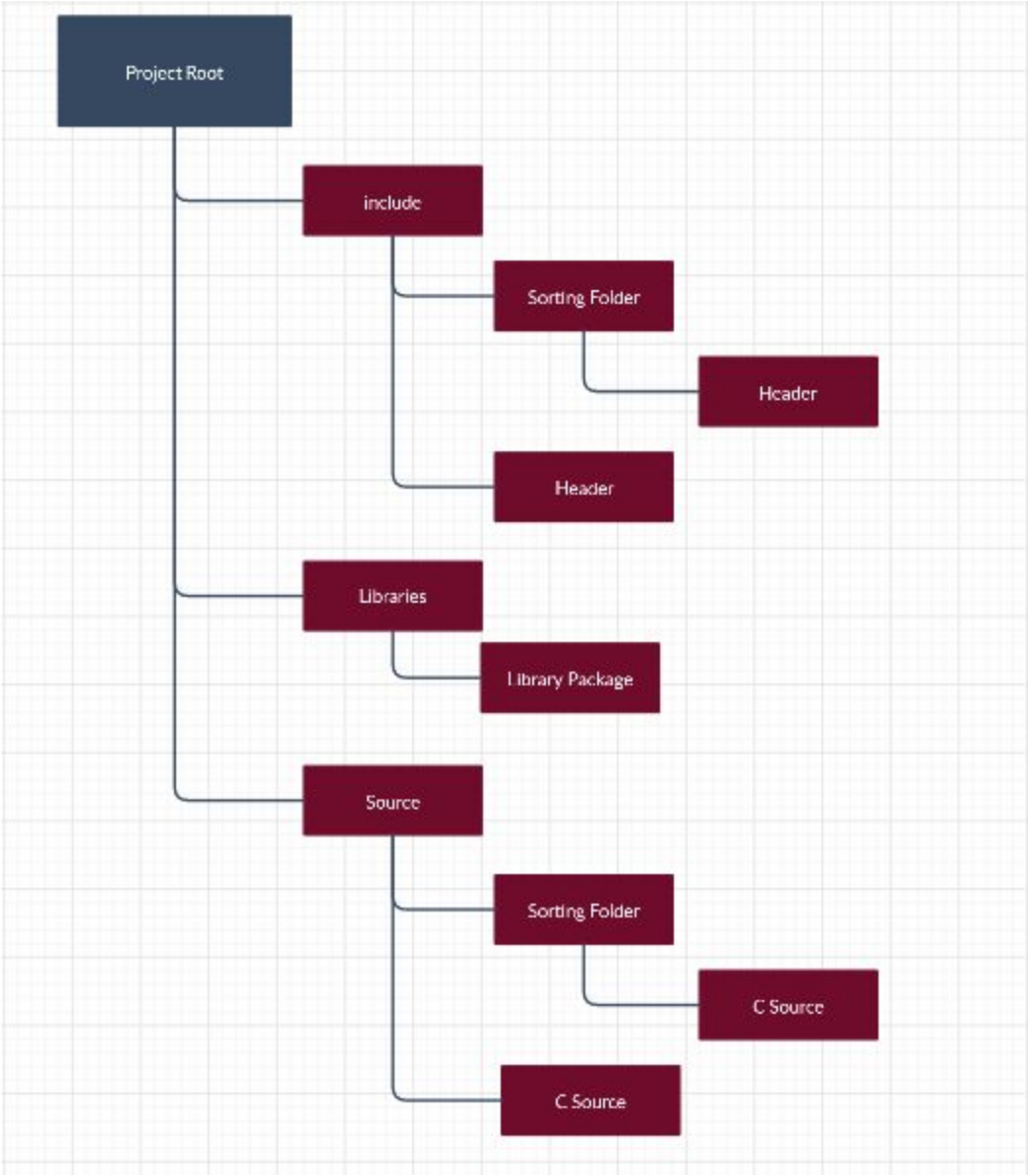
4.1.1 De Centrale



Figuur 4.1.2.1 *Bestanden structuur voor De Centrale.*

Het Python programma voor de besturingseenheden gebruikt de folder structuur die hierboven is afgebeeld. Het project bestaat uit drie top level folders. Dashboard is de equivalent van een “src” folder, hier zal dus alle application logic komen. In de libs folder zullen onze eigen modules komen waar onder seriële communicatie en een implementatie van het SHIT protocol. De venv folder is voor de virtual environment (lees: *4.3.1.2 Virtual environment*). De config folder dient voor het opslaan van de json files.

4.1.2 De besturingseenheden



Figuur 4.1.2.1 *Bestanden structuur voor C programma.*

Het C programma voor de besturingseenheden gebruikt de folder structuur die hierboven is afgebeeld. Het project bestaat uit 3 folders: een include folder voor alle headers, een libraries folder voor eventuele externe library bestanden en een source folder waar alle C source files in komen te staan. Als de folders te druk en onoverzichtelijk worden zullen er subfolders toegevoegd worden die helpen met het sorteren van de code bestanden.

4.2 Programmeertalen

4.2.1 De Centrale

De GUI applicatie voor het dashboard van de Centrale zal geschreven worden in Python 3. Python is ontwikkeld onder een OSI-approved open source-licentie, waardoor het vrij bruikbaar en verspreidbaar is, zelfs voor commercieel gebruik. In samenwerking met deze taal gebruiken we ook de Python Package Index (PyPI). PyPi host duizenden modules van derden voor Python.

4.2.2 De besturingseenheden

Alle code die draait op de besturingseenheden wordt geschreven in AVR-C. Dit is de programmeertaal C die de compiler van AVR gebruikt. Naast deze compiler maken we gebruik van PlatformIO, een extensie voor Visual Studio Code die extra functionaliteit toevoegt specifiek voor embedded systemen. Met deze extensie kunnen we vanuit Visual Studio Code de Arduino flashen, monitoren, debuggen en compileren met de avr-gcc compiler.

4.3 Libraries en modulen

4.3.1 De Centrale

4.3.1.1 PIP

Voor het beheren van libraries en module zal gebruikt gemaakt worden van **PIP** (Package Installer for Python). PIP kan gebruikt worden om libraries uit de **Python Package Index** en andere indexen te installeren.

PIP heeft ook ondersteuning voor een requirements bestand. In dit bestand worden alle nodige libraries genoteerd met de versienummers. De conventie is om dit bestand **requirements.txt** te noemen. Zie *figuur 4.3.1.1* voor alle nodige libraries en modules.

Het volgende commando kan vervolgens gebruikt worden om alle libraries en modules te installeren via PIP:

pip install -r requirements.txt

4.3.1.2 Virtual environment

Om versie problemen tussen libraries en python te voorkomen zal er gebruik gemaakt worden van een virtual environment. Een zelfstandige mapstructuur met een Python-installatie voor een bepaalde versie van Python, plus een aantal project specifieke libraries en modules.

Windows en Unix system werken echter iets anders als het gaat om het instellen van deze omgeving. Om deze reden zal er een **serve.sh**, voor UNIX, en een **serve.bat**, voor Windows, aangemaakt worden. Deze shell scripts zullen ervoor zorgen dat de virtuele omgeving wordt ingesteld met de juiste libraries en python versie.

4.3.1.3 Libraries en versies

Hieronder volgt een overzicht van alle libraries die gebruikt zullen worden van 3de partijen. Daarnaast zijn ook de meest belangrijke, maar niet alle, libraries uit de Python Standard Library hier gedocumenteerd.

Library	**Versie**
3de partij	
PySerial	3.4.0
matplotlib	3.1.1
ttkthemes	2.3.0
Python Standard Library	
Tkinter (GUI library)	*Meest recente versie in Python Standard Library*
collections	
os	
json	
random	
string	
math	

Figuur 4.3.1.1 *Gebruikte libraries en versies.*

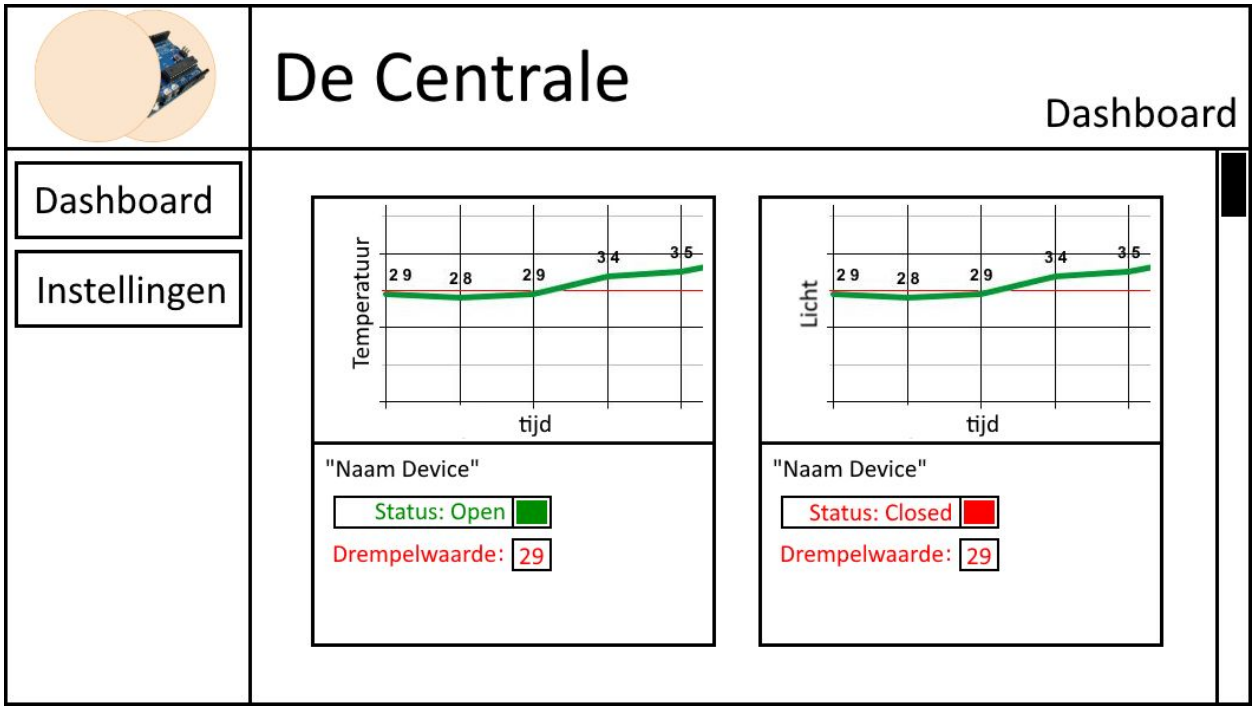
4.3.2 De besturingseenheden

De broncode van de besturingseenheden zal gebruik maken van de AVR standaard libraries. Dit zijn standaard header files die kunnen worden gebruikt die inbegrepen zijn bij de AVR-C Compiler. Daarnaast wordt de standaard header gebruikt die is gemaakt door de Hanzehogeschool genaamd de AVR_TTC_Scheduler.

5 Interfaces

5.1 Views concretisering

5.1.1 Dashboard



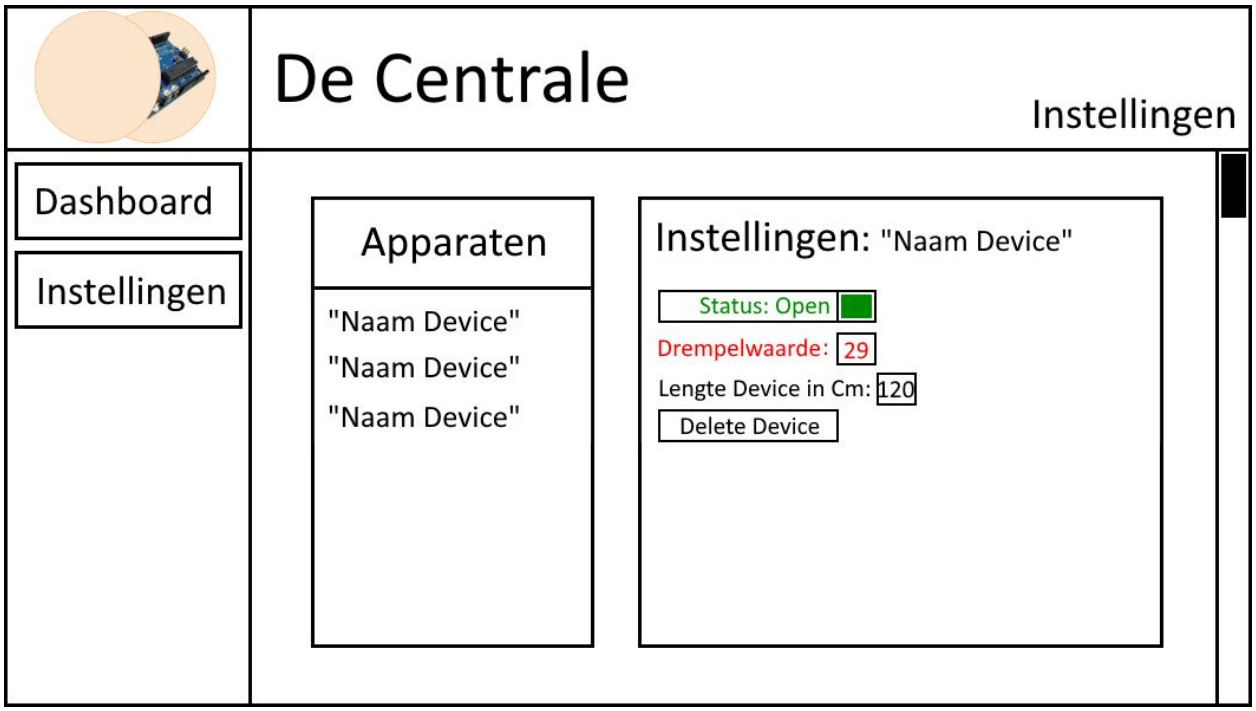
Figuur 5.1.1.1 *Dashboard interface.*

Op deze pagina worden alle aangesloten bestuurseenheden weergeven met een grafiek van de desbetreffende sensor waarde. De rode lijn geeft de ingestelde drempelwaarde aan.

Als er een nieuwe device wordt aangesloten komt er een popup met de tekst “naam van het device” en een invoerveld. De naam die hier wordt ingevoerd zal vervolgens op de plek “Naam device” worden ingevuld.

Het drempelwaarde invoerveld doelt hier op de drempelwaarde voor de temperatuur of licht. De grafiek werkt realtime, dus als er een device wordt aangesloten zal de grafiek zich op dat moment gaan vullen. De max tijd om terug te kijken in de grafiek is een half uur.

5.1.2 Instellingen



Figuur 5.2.1.1 *Instellingen interface.*

In de instellingen pagina is het mogelijk om de drempelwaarde van beide sensoren aan te passen, temperatuur\licht en afstand. Hier staan ook alle device die op dit moment niet aangesloten zijn. Tot slot is het hier ook nog mogelijk niet aangesloten device te vergeten. Dat wil zeggen de opgeslagen data en instellingen van het device worden verwijderd uit het json bestand. De verschillende statussen die een device kan hebben zijn: Disconnected, Closed, Open en Transitioning.

5.2 Class diagram de centrale

5.2.1 Application

De Application klasse is de hoofdklasse van de applicatie, hier draait de main system loop. Deze klasse is ook verantwoordelijk voor het initialiseren van de views klassen, dashboard en settings.

5.2.2 Settings

De settings klasse regelt het renderen van de gebruikers interfaces die bij het settings scherm hooren. Deze klasse vraagt de devices configuratie json file op voor het weergeven en instellen van de device instellingen. Deze klasse biedt ook een interface om waardes te bewerken.

5.2.3 Dashboard

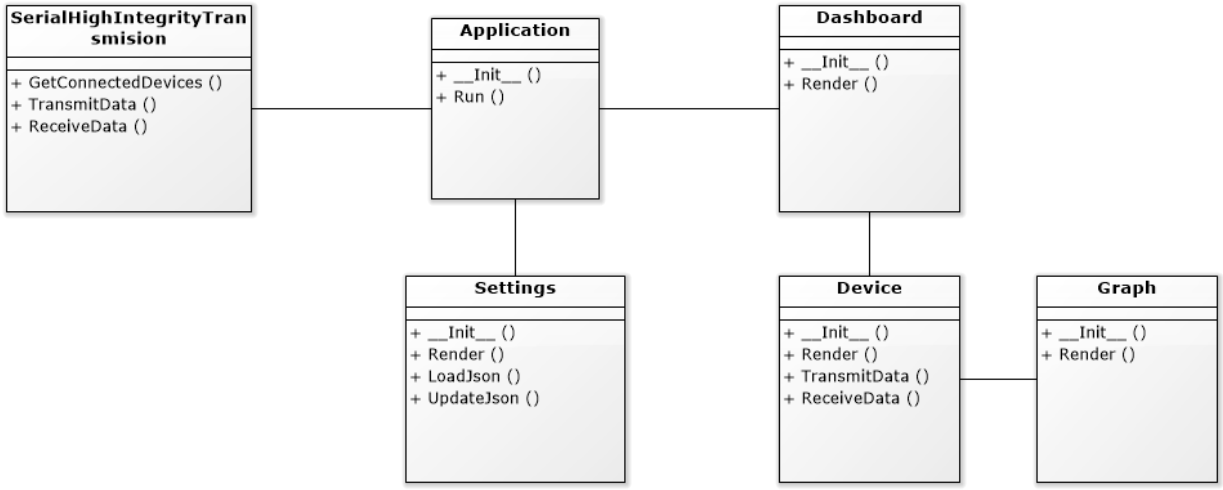
De dashboard klasse regelt het renderen van de gebruikers interfaces voor de bestuurseenheden die in het dashboard scherm getoont dienen te worden. In dit scherm worden de aangesloten devices weergegeven. Voor elk aangesloten device wordt een Device klasse aangemaakt. Elke device heeft ook een grafiek die de nodige metadata weergeeft.

5.2.4 Device

De device klasse is verantwoordelijk voor het renderen van de gebruikersinterface voor een device card in het dashboard. De device klasse representeert altijd een aangesloten bestuurseenheid. Deze klasse biedt vervolgens ook de mogelijkheid om te communiceren met het device via het communicatie protocol uit *hoofdstuk 3*.
De device klas zal regelmatig data opvragen aan het device om de betreffende grafiek te bewerken.

5.2.5 Graph

De graph klasse maakt een graph voor het bijbehorende device. De informatie die hiervoor nodig is wordt meegegeven in de initialisatie.

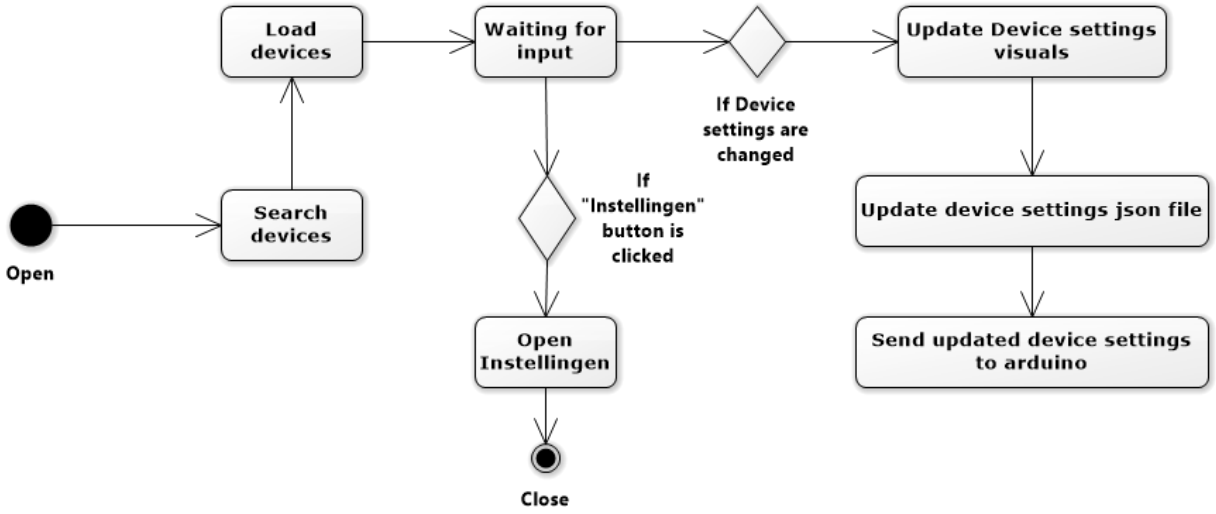


Figuur 5.2.1 *UML Diagram voor Centrale applicatie*

5.3 Dashboard scherm

Dit is de state machine diagram voor het dashboard scherm. De eerste stap die het dashboard uitvoert is dat zodra dit scherm geopend wordt is het zoeken naar aangesloten devices. Vervolgens worden device visualisaties ingeladen waarin de instellingen worden weergeven.

Vanaf hier dicteert de gebruiker de volgende stappen. De gebruiker kan er nu voor kiezen om de “instellingen” knop aan te klikken wat het instellingen scherm opent en het dashboard scherm sluit. Of de gebruiker kan de instellingen van een van de aangesloten devices aanpassen.

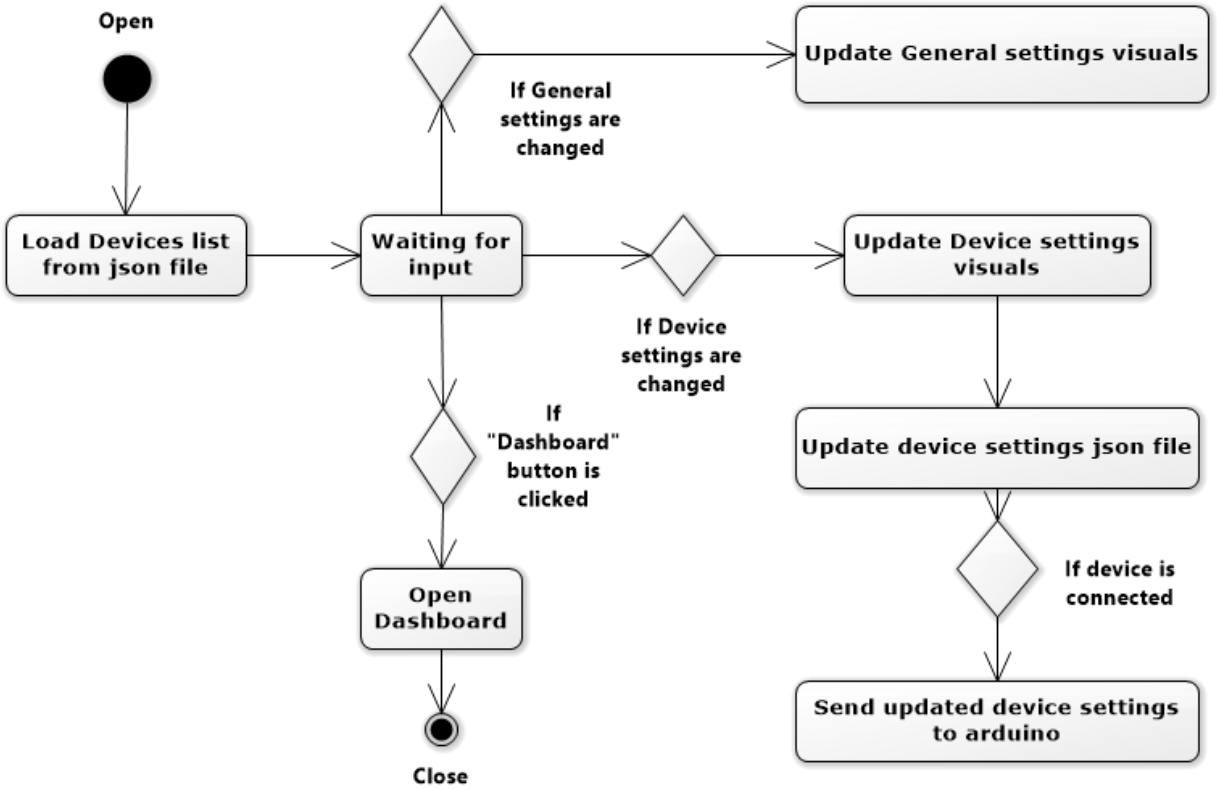


Figuur 5.3.1 *Dashboard state machine diagram.*

5.4 Instellingen scherm

Dit is de state machine diagram voor het instellingenscherm. De eerste stap die dit scherm uitvoert is het inladen van de json file die de device settings opslaat. Vanaf hier dicteert de gebruiker de volgende stappen. De gebruiker heeft de optie om op de “dashboard” knop te klikken om het dashboard te openen, wat vervolgens het instellingen scherm opent.

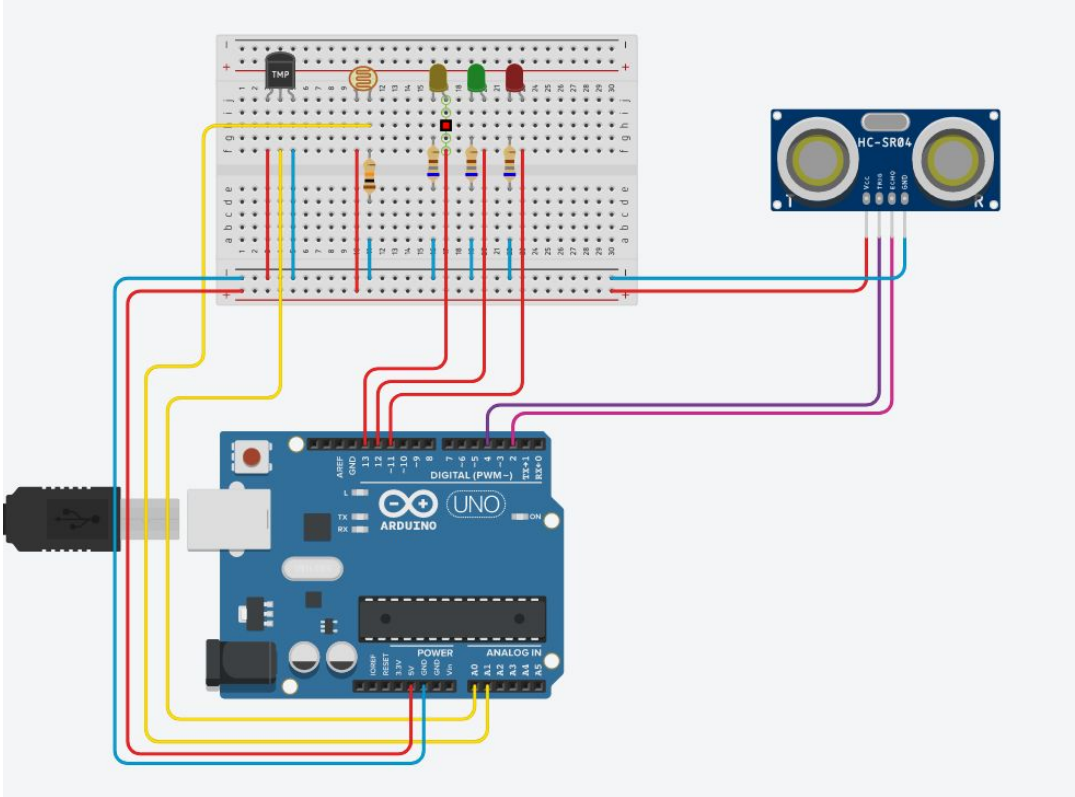
Op het instellingenscherm heeft de gebruiker ook de mogelijkheid om globale device settings aan te passen en individuele device settings aan te passen, het updaten van globale device settings zorgt ervoor dat elk nieuw aangesloten device gebruik maakt van de globale settings. Device settings kunnen op dit scherm net als op het dashboard scherm ook individueel worden ingesteld.



Figuur 5.4.1 *Instellingen state machine diagram.*

6 Besturingseenheden

6.1 Circuit



Figuur 6.1.1 *Visualisatie aansluitingen Arduino Uno.*

Het circuit voor de besturingseenheden bestaan uit een Arduino Uno, 3 LED lampjes, een ultrasoon sensor en een trigger sensor. Er zijn twee variaties van de besturingseenheden de eerste heeft een trigger sensor dat de lokale temperatuur meet en de tweede meet de intensiteit van het licht. De Atmega328p chip van de Arduino Uno wordt gebruikt om het systeem aan te sturen en door middel van een usb kabel verbinden we de systemen met de Centrale.

In **Figuur 6.1.1** is een visualisatie van het circuit voor het project te zien. Hierin zijn beide trigger sensoren aangesloten op een Arduino Uno. In het uiteindelijke eindproduct zal dit niet het geval zijn en zullen beide trigger sensoren de A0 pin gebruiken.

6.2 Analog to Digital Convertor

Voor de analoge sensoren hebben we de **ADC** van de Arduino Uno nodig. Voordat we deze kunnen gebruiken moet als eerste de **ADMUX** en **ADCSRA** Registers ingesteld worden. Nadat deze registers zijn ingesteld kunnen er analoge sensoren aangesloten worden op de pinnen waar de ADC aan vast zit. De ADC kan zo met een resolutie van 10 bits, oftewel 1024 stappen de waarde converteren naar een digitale representatie.

6.2.1 ADMUX Register

Bit	7	6	5	4	3	2	1	0
	REFS1	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Value	0	1	0	0	0	0	0	0

Figuur 6.2.2.1 *ADMUX Register*

In de **ADMUX** Register zetten we de 7e bit om aan te geven dat we een voltage reference voor de **ADC** gebruiken dat overeenkomt met de **AVCC** met een eventuele externe capacitor. De volgende tabel laat de verschillende voltage referenties zien die ingesteld kunnen worden. Voor de toepassing van dit project wordt alleen **REFS0** hoog gezet.

REFS1	**REFS0**	**Voltage Reference Selection**
0	0	AREF, internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V voltage Reference with external capacitor at AREF pin

Figuur 6.2.2.2 *Voltage Referenties voor ADC van de Arduino Uno.*

6.2.1 ADCSRA Register

Bit	7	6	5	4	3	2	1	0
	ADEN	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
Read/ Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial	1	0	0	0	0	1	1	1

Figuur 6.2.3 *ADCSRA Register*

De **ADCSRA** register houdt bij wat de status is van de **ADC**. Hierin wordt **ADEN**, **ADPS2**, **ADPS1** en **ADPS0** hoog gezet. Met de **ADEN** bit kunnen we aangeven of de ADC aan of uit moet. Als hier een 0 naar geschreven wordt, werkt de conversie niet. De drie **ADPSn** registers geven de division factor van de prescaler van de **ADC** aan. Voor dit project wordt een division factor van 128 gebruikt door ze alle drie hoog te zetten. Na deze initiële setup kunnen door middel van de **ADSC** bit hoog te zetten een conversie starten. Het resultaat van de conversie kan opgehaald worden uit de **ADC**.

ADPS2	**ADPS1**	**ADPS0**	**Division Factor**
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Figuur 6.2.4 *ADCSRA prescaler division factor*

6.3 Sensoren

6.3.1 Temperatuur en lichtsensor

De temperatuur en lichtsensoren zijn de trigger sensoren voor de twee variaties van de besturingseenheden. Beide trigger sensoren zijn analoge sensoren en worden ten alle tijden op 5V aangesloten.

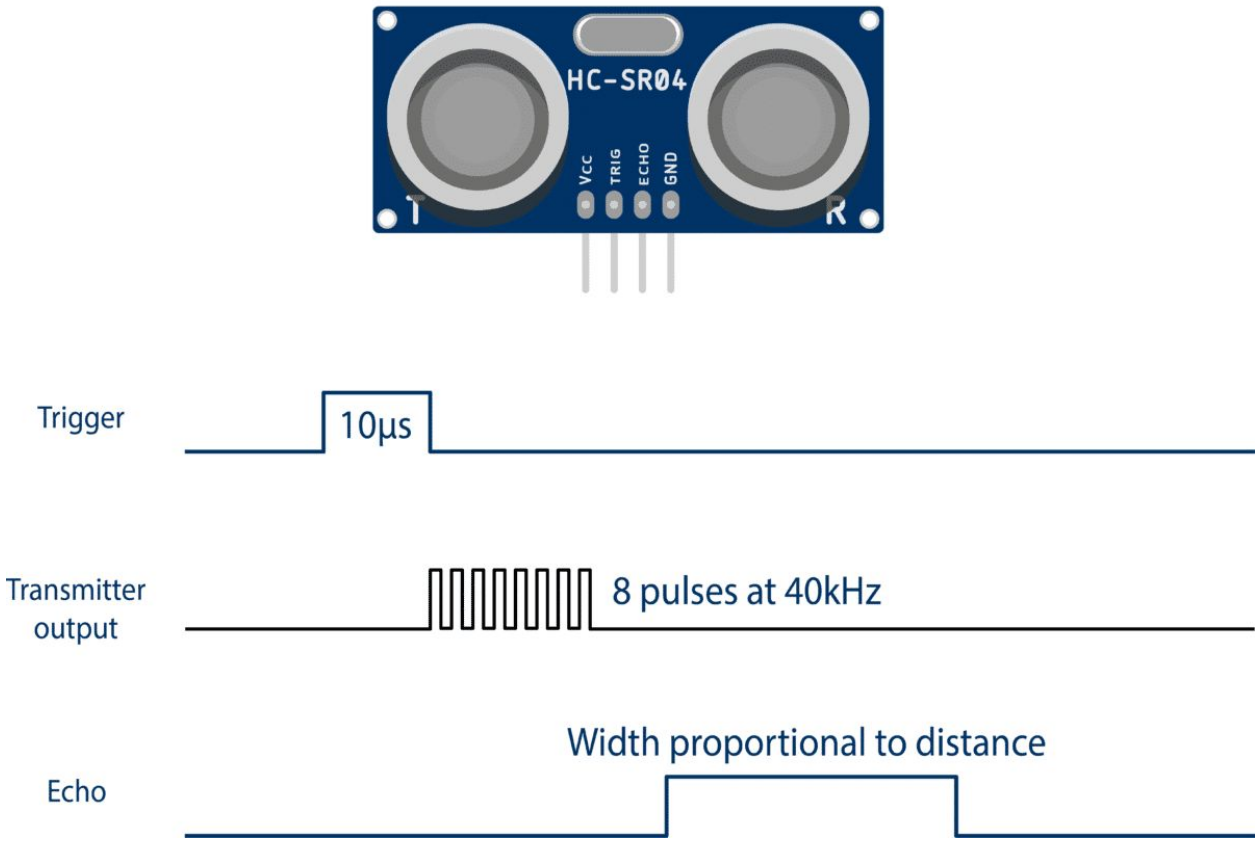
De temperatuursensor heeft een output pin specifiek voor het uitlezen van de waarde die de sensor teruggeeft. Deze pin wordt daarom aangesloten op de A0 pin van de Arduino. Deze Pin is direct aangesloten op de ADC van de Arduino Uno waardoor we de waarde van de sensor kunnen uitlezen met een resolutie van 1024 stappen. Met de volgende berekening kan dan de temperatuur in Celsius uitgerekend worden: *((ADCValue * 5 / 1024) - 0.5) * 100 = Temp in Celsius*. Om de Temperatuur in Fahrenheit te krijgen kunnen we deze waarde nemen en hem in de volgende formule stoppen: *(TempInCelsius * 9.0 / 5.0) + 32 = Temp in Fahrenheit.*

De lichtsensor heeft niet een output pin zoals de temperatuursensor. Dit komt omdat de temperatuursensor een transistor is en de lichtsensor werkt als een weerstand. Om de waarde te lezen van de sensor moet de lichtsensor aangesloten worden op 5V met een 10 kilo ohm resistor aan de kant van GND. Als dit is aangesloten kan de waarde van de lichtsensor gelezen worden door een kabel aan te sluiten tussen de weerstand en de lichtsensor. Deze kabel wordt dan net zoals de temperatuursensor aangesloten op de A0 pin van de Arduino Uno.

Voor het ontwikkelproces worden beide sensoren tegelijk aansluiten als volgt: Temperatuursensor op pin A0 en de Lichtsensor op pin A1.

6.3.2 Ultrasone sensor

Net als de analoge sensoren wordt de ultrasone sensor ook ten alle tijden op stroom gezet. De ultrasone sensor is echter geen analoge maar digitale sensor. Naast VCC en GND heeft de ultrasone sensor een trigger pin en een echo pin. Om de ultrasone sensor te starten moet er een 10 microseconde lange signaal gestuurd worden naar de sensor. Zodra de sensor actief is, stuurt het een aantal ultrasone pulsen en vangt deze weer op. Tenslotte stuurt de sensor dan over de echo pin een signaal waarvan de lengte direct afhankelijk is van de afstand dat is gemeten door de ultrasone sensor.



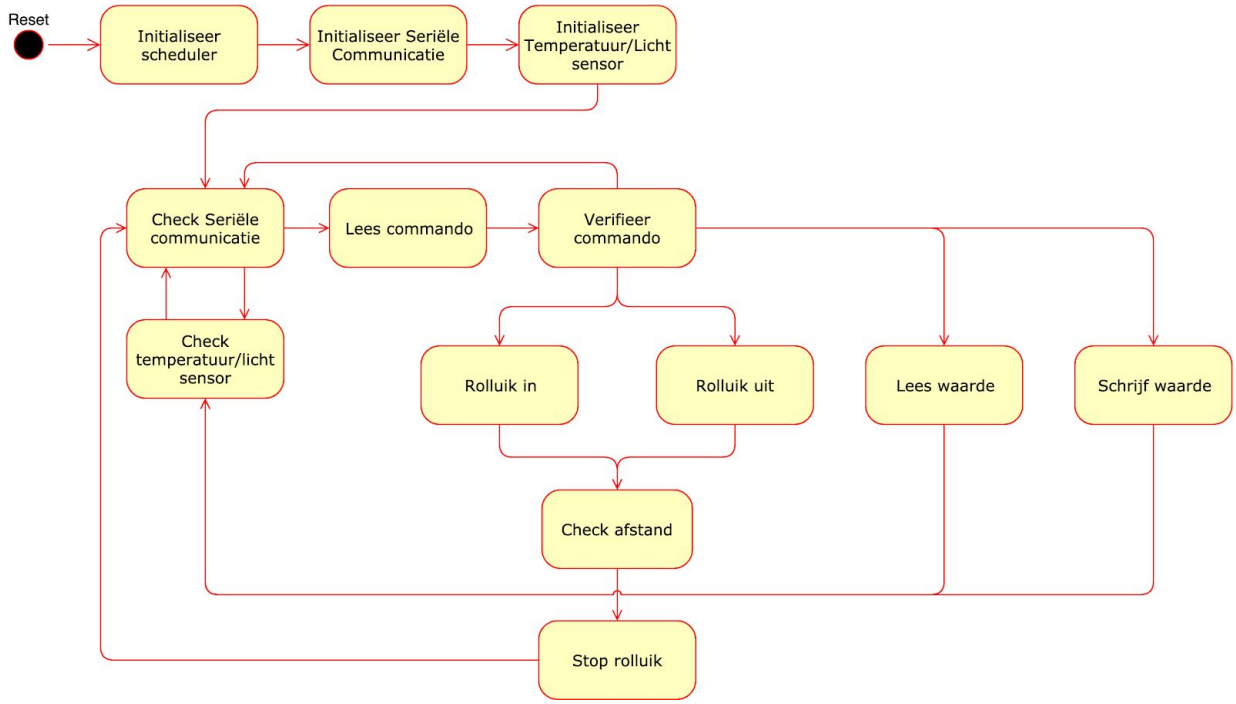
Figuur 6.3.1 *Tijd diagram van de hc-sr04 ultrasone sensor*

In **Figuur 6.3.1** hierboven is de tijd diagram van de ultrasone sensor die gebruikt wordt voor dit project te zien. Bovenaan is de trigger pulse te zien van 10 microseconden, daarna de 8 ultrasone pulsen op een frequentie van 40 kHz en de echo waarvan de lengte afhankelijk is van de afstand die is gemeten.

6.4 Functionaliteit

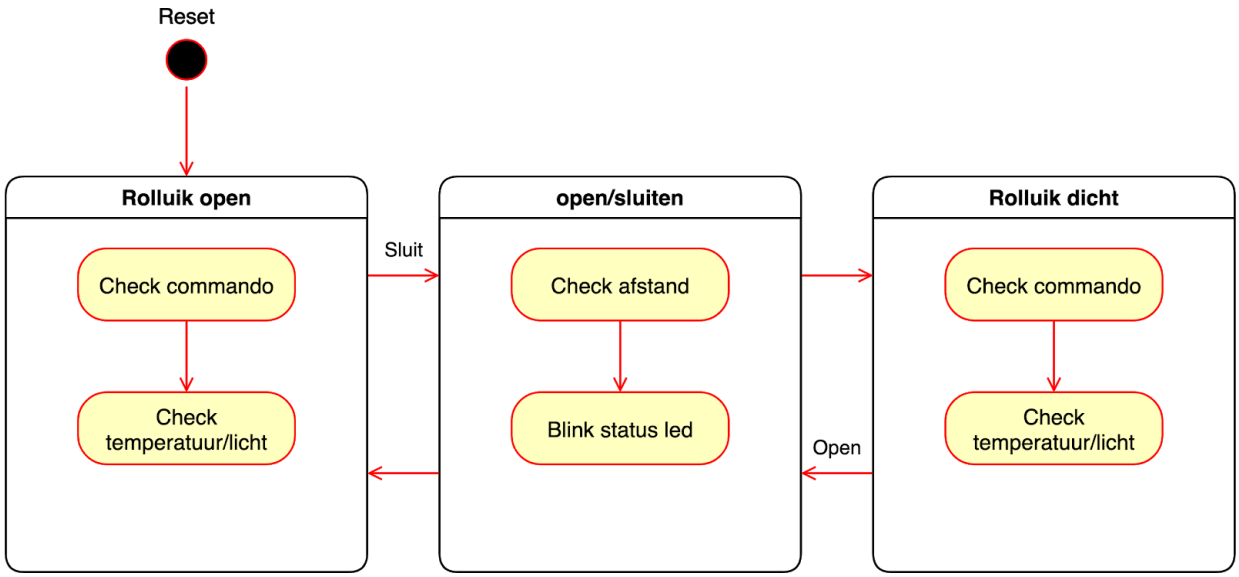
6.4.1 Program flow

In *Figuur 6.4.1* is de flow van de besturingseenheid te zien. Hierin staan alle belangrijke taken die de besturingseenheid heeft met betrekking tot het opstarten checken en uitvoeren van taken. De flow zal altijd opnieuw beginnen vanaf het reset commando die als eerst gegeven wordt aan de besturingseenheid. Hierna zal als eerst de scheduler geïnitialiseerd worden. Daarna zal met behulp van de scheduler de seriële communicatie geïnitialiseerd worden en hierna de sensoren. De zogenaamde cyclus van het programma zal altijd beginnen met het checken op een commando via de seriële verbinding. In het geval dat er geen commando is verstuurd van het dashboard zal de besturingseenheid checken of de huidige waarde van de sensoren groter of kleiner is dan de maximale of minimale waarde. Dit is afhankelijk van de waarde die is ingesteld op de besturingseenheid. Wanneer er wel een commando is verstuurd vanaf het dashboard, zal de besturingseenheid wachten totdat het commando compleet is verstuurd. Daarna zal de besturingseenheid het commando verifiëren op zijn integriteit. Wanneer de integriteit niet goed is zal de besturingseenheid een error code terug sturen naar het dashboard. Is de integriteit van het commando wel goed, dan zal het commando uitgevoerd worden. Wanneer het commando volledig is uitgevoerd dan zal de besturingseenheid weer verder gaan met het checken van de licht en temperatuur sensor. Waarna de cyclus weer opnieuw begint.



Figuur 6.4.1 *Program flow diagram.*

6.4.2 State machine



Figuur 6.4.2.1 *State machine diagram.*

6.4.3 States

Er zijn 3 states die de besturingseenheid kan hebben. Dit zijn open, gesloten en transitie. De status open zal eerst kijken of er een commando is verstuurd. Daarna wordt de temperatuur of licht sensor gecheckt. Als de sensor een waarde geeft die buiten de threshold ligt, dan zal de status veranderen naar de transitie. Op basis van de voorgaande status zal de transitie het rolluik naar boven of naar beneden rollen. Tijdens de transitie status zal er gekeken worden wat de afstand is van het rolluik ten opzicht van het einde. Daarna worden ledjes goed gezet. Deze cyclus wordt herhaald totdat het rolluik open is of dicht is. Daarna zal de status verder gaan vanuit de open of gesloten status. Hierna herhaalt de gehele cyclus zich weer.

7 Conventies

C code convention

```
// Data types.
int a = 0;
float b = 0.0f;
double c = 0.0;
```

```
// Binary operations
a = (1 << 2);
a &= ~(1 << 2);
a |= (1 << 2);
```

```
// Every header needs to
include a #define guard.
#ifndef __NAME_H
#define __NAME_H
...
#endif // __NAME_H
```

```
// Functions
void test()
{
    // Statements.
}

void test(float x, char* string)
{
    // Statements.
}
```

```
// Control statements.
If (x < y) {
    // Statements.
}

if (x < y) {
    // Statements.
}
else if (y < x) {
    // Statements.
}
else {
    // Statements.
}

If (x < y)
    // Ommit braces incase of oneliner.

do {
    // Statements.
}
while();

while(1) {
    return 0;
}
```

```
// .h files
// Header guard.
#ifndef __NAME_H
#define __NAME_H

// Includes.
#include <avr/io.h>

// Defines.
#define X 3.0

// Function declarations.
void function();

#endif // __NAME_H

// .c files.
// Includes.
#include <avr/io.h>

// Function implementations.
void function()
{
    // Statements.
}

void long_name_function()
{
    // Statements.
}
```

Figuur 7.1 *Overzicht van de C code conventie.*

Python code convention

```
# Comments.
# Variables.
a = 0
b = “hello”
c = ClassName()
long_name = [1, 2]
```

```
# Classes.
class MyClass():
    def __init__(self):
        pass

class MyClass(ParentClass):
    def __init__():
        pass
```

```
# Functions.
def my_function():
    return 0

def function():
    pass
```

```
# Control statements.
if(x < y):
    pass

while True:
    pass

for x in y:
    pass
```

Figuur 7.2 *Overzicht van de Python code conventie.*