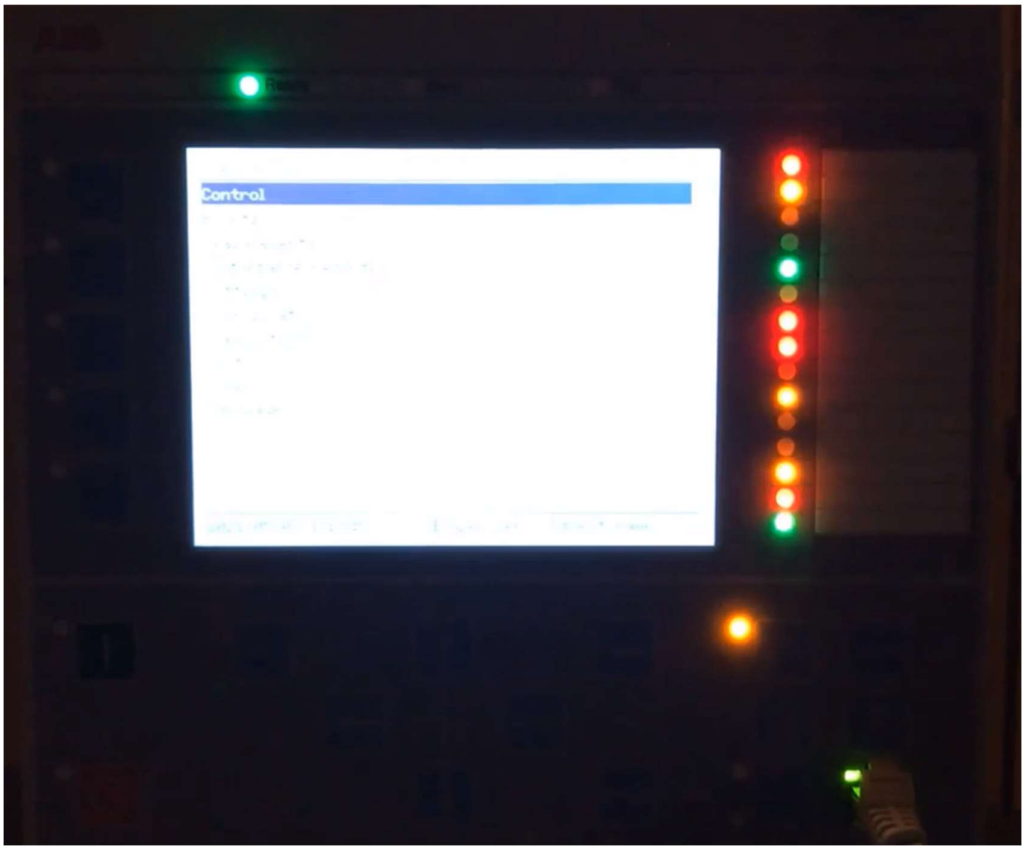
Detektion av lysdioder



Nils Borg

Kevin Klarin

Jesper Jansson

Victor Fagerström

ABB-Industrigymnasium 2020/2021

Uppdragsgivare: Kenneth Saxin, Anders Wårdenius Lindqvist, Hans Jernberg

Sammanfattning

Detta är en teknisk rapport som beskriver en enhet som skapades för ett gymnasiearbete vid ABB Industrigymnasium på uppdrag av Hitachi ABB Power Grids som går ut på att detektera lysdioder på en reläskyddsterminal. Rapporten behandlar enhetens konstruktion, detektions- och kommunikationsmetoder.

ABB Hitachi Power Grids önskade automatisera tester av reläskydd. Manuell avläsning var mycket krävande och långsamt. Därför skapades enheten denna rapport beskriver. Enheten består av ett kamerakort, fäst i en hållare som ger optimal positionering och ljusförhållanden för så träffsäker detektion som möjligt. Kamerakortet kopplas i sin tur till en ytterligare enhet som för över resultatet till en delad textfil.

Resultatet blev en enhet som för det mesta klarade av kraven den ställdes inför. Detektionen fungerade med stor träffsäkerhet vid slutet av projektet.

Abstract

This is a technical report that describes a device, developed for a culminating project at ABB Industrial Gymnasium at the request of ABB Hitachi Power Grids that aims to detect LED lights on a terminal for a protective relay. The report describes the device's construction, detection methods and communication methods used.

ABB Hitachi Power Grids wished to automate the testing process of their relays. Manually reading the results was very demanding and slow. Therefore, the device this report covers was created and lighting. It consists of a camera card, attached to a container for optimal camera positioning and lighting for maximum accuracy. The camera card is in turn connected to another device that transfers the data to a shared text file.

The product turned out to be a device that met the requirements placed on it. The detection worked well with good accuracy towards the end of the project.

Förord

Rapporten är skriven för det genomförda projektet åt uppdragsgivaren Hitachi ABB Power Grids som även stått för projektets kostnader.

Stort tack till uppdragsgivarna och handledarna Kenneth Saxin, Anders Wårdenius Lindqvist och Hans Jernberg!

Tack till lärarna Andreas Jillram, Daniel Åkerlund, Joakim Flink och Jeton Mustini på ABB Industrigymnasium för er handledning och input till projektet.

Innehållsförteckning

Sammanfattning 2
Abstract 2
Förord 2
1. Syfte 6
2. Bakgrund och teori 6
3. Metodik och Material 7
3.1 Metodik 7
3.1 Material 7
4. Resultat 8
4.1 Systembeskrivning 8
4.2 Representation av data 8
4.3 Detektion 8
4.3.1 Bildbehandling med mjukvara 9
4.3.2 Maixpys inbygga färgdetektering 11
4.4 Kommunikation och strömförsörjning 12
4.4.1 Seriell kommunikation 13
4.4.2 Fildelning 13
4.4.3 Integrering 14
4.4.4 Strömförsörjning 16
4.5 Hållare 16
4.5.1 Konstruktion 16
4.5.2 Filtret 17
5. Diskussion 17
5.1 Kommunikation 17
5.2 Representation av data 19
5.3 Detektion 19
5.3.1 Maixpys inbyggda färgdetektering 19
5.3.2 Filter 20
5.4 Val av hårdvara 20
5.5 Hållare 21
5.5.1 Förundersökning 21
5.5.2 Prototyp 1 och 2 22
5.5.3 Filtret och avdelaren 22

5.5.4 Prototyp 3 24
5.6 Integrering 24
6. Slutsats 25
Referenser 25

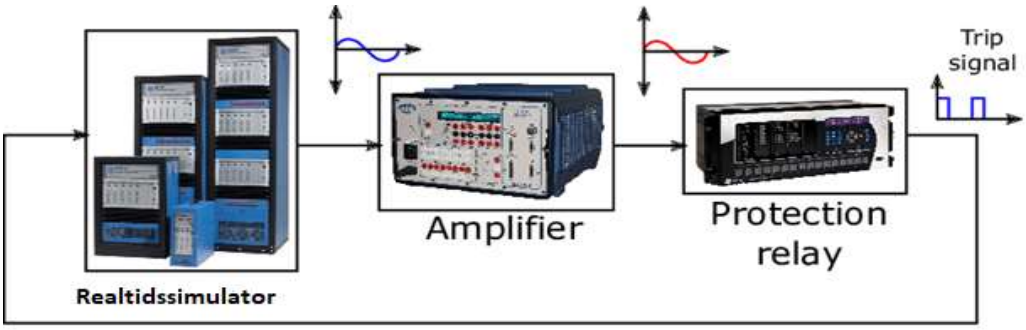
1. Syfte

Syftet med projektet är att med hjälp av bildigenkänning kunna detektera vilka av de 15 lysdioderna som är tända. Vilken färg de tända dioderna har ska också kunna detekteras. Om analysen misslyckas ska ett felmeddelande skickas ut. Resultatet ska kunna nås från uppdragsgivarnas befintliga simulatorsystem. Mätningarna ska även kunna tas på kommando, så att en tagning kan startas antingen manuellt eller triggas igång automatiskt av ett script. Analysen görs i dagsläget genom manuell avläsning som är tids- och resurskrävande. Processen behöver därför automatiseras.

2. Bakgrund och teori

Hitachi Power Grids tillverkar skydds- och automationsprodukter för växelströmsnät. På Grid automation i Västerås jobbar ungefär 140 anställda och man tillverkar reläskydd. På uppdragsgivarnas avdelning utförs tester i en realtidssimulator för att efterlikna verkligheten. Realtidssimulatorn är en mycket kraftfull dator som simulerar kraftnät i realtid i samma hastighet som i verkligheten, vilket leder till att hårdvaran tror att den verkar i ett riktigt elnät.

Figur 1 visar ett exempel på en uppställning av ett testsystem. Realtidssimulatorn ger ut mycket små analoga signaler som i sin tur förstärks innan de når reläskyddet (Protection relay i figur 1).



Figur 1. Exempelbild av ett testsystem (bilden visar inte ABB Hitachis specifika utrustning).

Resultatet från reläskyddet visas med 15 lysdioder. Merparten av dessa reläskyddsterminaler kan mata ut resultatet digitalt för vidare användning av data, medan några ej har denna funktionalitet. Se figur 2 för beskrivning av processen utan automatiserad detektering.

Figur 2. Blockdiagrammet visar en beskrivning av processen utan automatisering.

Ett automatiserat system behövs som läser av lysdioderna och kan lagra resultatet på en disk med åtkomst över nätverket.

Lösningar för avläsning av färgdata finns utbrett i form av färgsensorer, men dessa är inte syftade till att använda för så pass små och närliggande dioder som i detta fall. Liknande problem har lösts med Computer Vision, genom att använda sensorisk input från en kamera och detektera områden med färgdata. Detta används exempelvis för detektering av trafiksignaler i självkörande bilar[1]. Flera tekniker från denna lösning används i detta projekt.

3. Metodik och Material

3.1 Metodik

Resultatet uppnåddes med hjälp av testning av olika metoder och konstruktioner tills ett korrekt resultat uppmättes i helhetssystemet. Olika kamerainställningar testades i kamerakortets user interface-miljö tills detektionen överensstämde med verkligheten. Olika konstruktioner testades även, tills en hållbar modul som gav kameran möjlighet till korrekt positionering hittades.

3.1 Material

För arbetet har följande material använts:

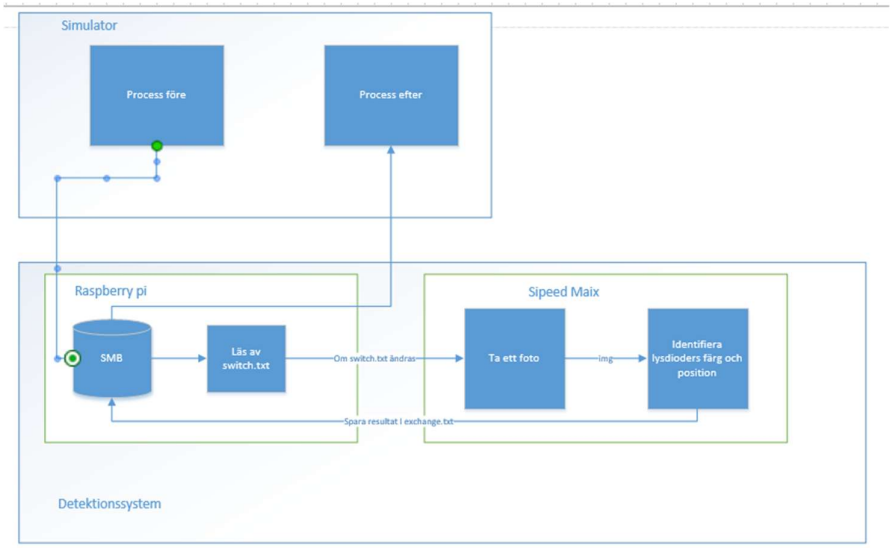
- Maix M1 dock
- Raspberry pi 3b+
- PLA-plast
- Reläskyddsterminal samt labbdator för tester

[1] Alkiek, K. (2018), *Traffic Light Recognition – A Visual Guide*, https://medium.com/@kenan.r.alkiek/https-medium-com-kenan-r-alkiek-traffic-light-recognition-505d6ab913b1

4. Resultat

4.1 Systembeskrivning

I figur 3 visas ett flödesschema över det fullständiga system som skapats och hur det interagerar med uppdragsgivarnas system. Visionssystemet triggas igång av att en textfil i en delad mapp ändras. Mappen delas av en Raspberry Pi som lyssnar efter ändringar. När systemet triggats igång skickas en signal till kamerakortet som gör en tagning och svarar med resultatet i form av en textsträng. Denna läggs in i resultatfiler som simulatorn läser av.



Figur 3. Blockdiagrammet visar en systembeskrivning över den automatiserade processen.

4.2 Representation av data

Utdatan i detta projekt representeras som en textsträng som i sin tur består av 15 tecken. Denna sträng beskriver ordningen i vilken lamporna är tända på reläskyddet och använder r för röd, g för grön, y för gul och o för släckt diod. Dessa tecken separeras av kommatecken.

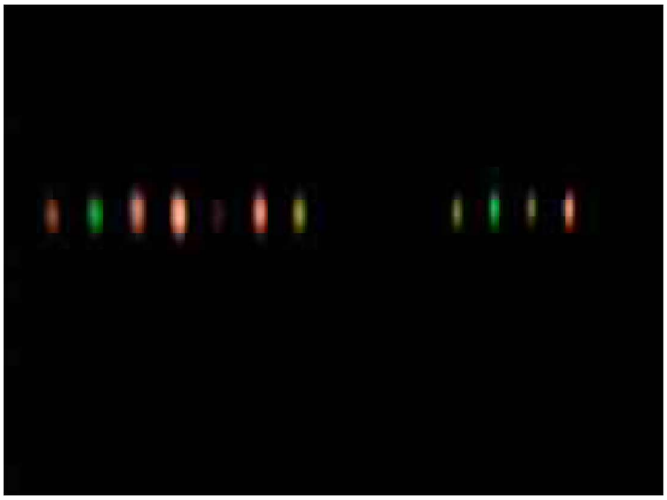
4.3 Detektion

För detekteringen används Sipeed Maix M1 dock som är hårdvara som integrerar enkel bildbehandling med hjälp av ett eget bibliotek byggt ovanpå micropython[2] [3]. Denna är kopplad till en kamera för att kunna ta bilder på reläskyddet samt visa det som programmet har upptäckt för felsökningsmässiga anledningar.

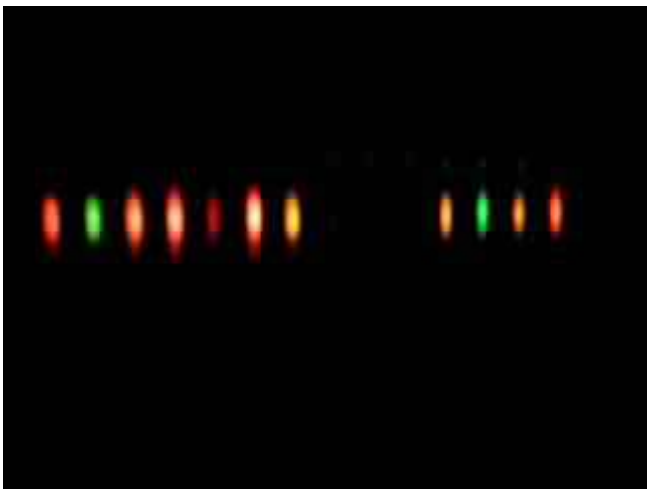
[2] Sipeed. (2020), *Specifications*, https://cn.dl.sipeed.com/shareURL/MAIX/HDK/Sipeed-M1&M1W/Specifications
[3] Sipeed wiki. (2021), *Introduction to MaixPy documentation*, https://wiki.sipeed.com/soft/maixpy/en/

4.3.1 Bildbehandling med mjukvara

Flera olika tekniker används för att bearbeta den bildinformation som kameran fångar in. Till att börja med är två standardinställningar för kameran ändrade. Kamerafrekvensen är halverad för att minska effekten av diodernas blinkande som uppstår på grund av deras pulsbreddsmodulering. En lägre kamerafrekvens ger i detta fall ett mer stabilt ljusflöde på bilderna som tas. Den andra inställningen är vitbalans som med standardinställning aktivt modulerar vitbalansen efter vad kameran exponeras för. Denna är avstängd för att återge färginformation konsekvent för varje tagning. Figur 4 återger bilden innan inställningarna ändrats och figur 5 efter ändringarna.

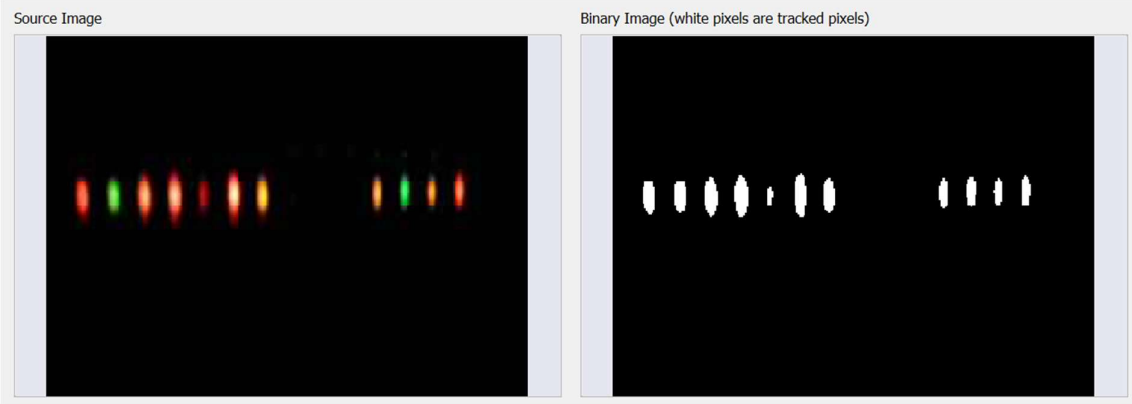


Figur 4. Bilden tagen med standardinställningar.

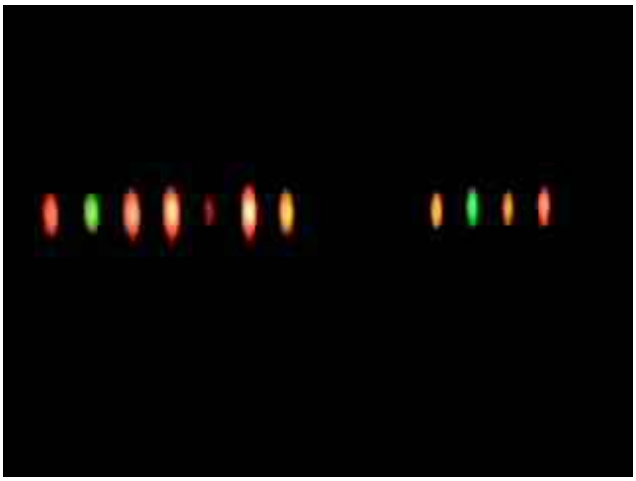


Figur 5. Bild tagen med ändrade inställningar

För att filtrera ut enbart lysdioderna för senare analys används först ett binärt filter. Detta jämför ljusstyrkan på varje pixel med ett visst tröskelvärde som bestäms av prövning i *MaixPy IDE* miljöns inbyggda *Threshold Editor* som visar vilka pixlar som hamnar under respektive över ett visst tröskelvärde likt i figur 6. De vita pixlarna i bilden till höger är de som ligger över tröskelvärdet. Den inbyggda metoden *binary()* maskerar bort alla pixlar i den ursprungliga bilden som i analysen hamnade under tröskelvärdet. De pixlar som återstår tillhör det ljus som dioderna utsänder vilket visas i figur 7.

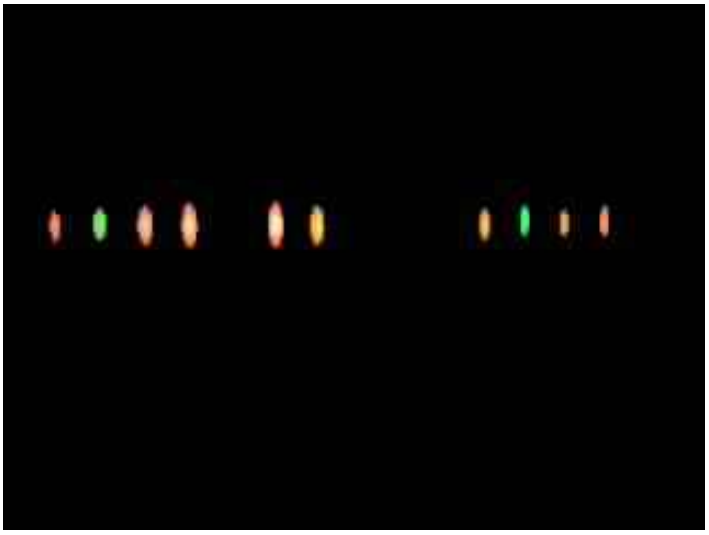


Figur 6. Bilden visar Maixpy IDEs inbyggda *Threshold Editor*.



Figur 7. Bild tagen med ett binärt filter applicerat.

Sedan används ett morfologiskt filter som finns inbyggt i *MaixPy-biblioteket* som heter *top_hat()* som filtrerar stora skillnader i ljusstyrka så att ljusstarka områden förstärks och mörkare områden blir mörkare.[4] Detta leder till att de återstående färgområdena har en jämn färg likt i figur 8.



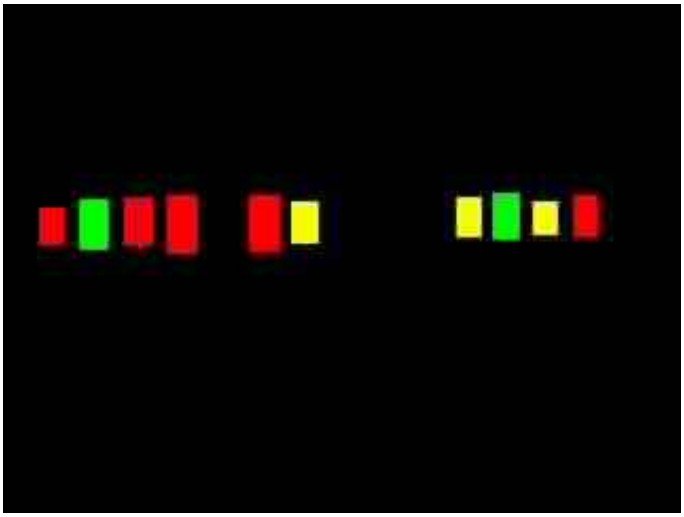
Figur 8. Bild tagen med *top hat.*

4.3.2 Maixpys inbygga färgdetektering

För att slutligen hitta lysdioderna används *find_blob-funktionen*, som är en inbyggd metod[5] i *MaixPy-biblioteket*. Denna metod letar efter färger i bilden som kameran tagit och returnerar en lista med alla "blobs" som den hittade. En *blob* är i sin tur en *array* med information som bland annat innehåller området inom vilket färgen hittades. Vidare, så accepterar *find_blob* bland annat färggränserna i ett *LAB* färgområde, minimala arean av *blob* och "*merge*", som bestämmer om närliggande *blobs* ska läggas ihop eller inte. Metoden körs två gånger med olika färggränser, som bestäms med hjälp av testning i *MaixPy IDE* miljöns *Threshold Editor*[6], för att kunna detektera de tre olika färgerna som dioderna kan anta. För gröna dioder görs en detektering med ett grönt färgspann. För röda och gula dioder görs en separat detektion som letar efter områden med någon av då två färgspannen. Dessa klassificeras som antingen röda och gula. Eftersom de röda och gula dioder som sitter på reläskyddsterminalen har ett överlappande färgspann i *LAB* formatet kan det därför hittas både röda och gula områden på en röd diod. Genom att köra metoden med två färgspann görs en

[4] Sipeed wiki. (2021), *image (machine vision),* https://wiki.sipeed.com/soft/maixpy/en/api_reference/machine_vision/image/image.html
[5] ibid
[6] Sipeed wiki. (2021), *MaixPy IDE installation and use*, https://wiki.sipeed.com/soft/maixpy/en/get_started/env_maixpyide.html

merge på tätt liggande detekterade *blobs*. Detta leder till att det enbart kan detekteras en *blob* per diod. Resultatet med de detekterade lysdioderna som ifyllda rektanglar visas i figur 9.



Figur 9. Bild med detekterade dioder som representeras av ifyllda rektanglar.

Efter *blob-detektionen* omvandlar programmet listan med *blobs* till en sträng innehållande de 15 tecken som ska representera de 15 lysdioderna på reläskyddet. Detta utförs genom att programmet sorterar *blobs* efter deras x-position där de som kommer först på skärmen hamnar först i listan. Därefter lägger den till de rätta färgerna i strängen i samma ordning som de ligger i listan. Om avståndet mellan två *blobs* blir för stort kommer en symbol som representerar en släckt diod läggas till i listan och utgångspositionen för nästa sökning blir framflyttad med ett uppmätt medelavstånd mellan dioder. Om avståndet istället blir för litet kommer mätningen ignoreras. Mätningen repeteras 10 gånger och ett medelvärde tas ut på var och en av positionerna. Denna detektion sker varje gång kortet får en signal från en Raspberry Pi.

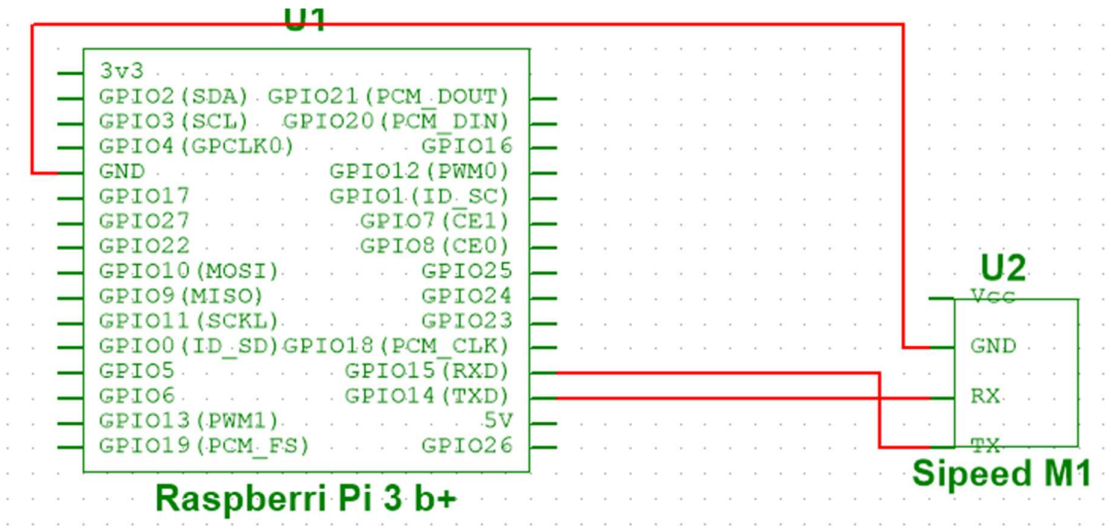
Genom testning med slumpmässiga kombinationer av tända dioder fungerar detektionen i nästan alla fall. Av 90 manuella testningar, var det fyra kombinationer som misslyckades. Samtliga av dessa fyra detektioner ogiltigförklarades av programmet och lades inte till i resultatet.

4.4 Kommunikation och strömförsörjning

Detektionsprogrammet beskriven i del 4.3 genererar output i form av en textsträng. För att kunna veta när en tagning ska göras och för att transportera resultatet används digital kommunikation. Dels kommunicerar huvudkortet, en Raspberry Pi, med simulatorsystemet, dels kommunicerar det med kamerakortet. De två tekniker som används för detta, samt hur de är integrerade genom ett *python-script* beskrivs i denna del. I del *4.4.3 Integrering* finns även en sammanställning över olika möjliga felmeddelanden och förslag på lösningar åt dessa.

4.4.1 Seriell kommunikation

För kommunikation används *UART* vilket är en datahårdvara som används för att kommunicera mellan två enheter.[7] *UART* använder sig av seriell kommunikation och överför data bit för bit från en avsändare till en mottagare. Enheter med stöd för *UART* har en ingång och en utgång för data. Ingången kallas för *RX* (Recieve) och *TX* (Transmit). I Figur 10 framgår hur enheterna är kopplade till varandra för kommunikationen. Strömförsörjningen visas ej i av figuren utan är beskriven i del *4.4.4 Strömförsörjning.*



Figur 10. Kopplingsschemat visar hur kopplingen för *UART-kommunikation* är gjord.

4.4.2 Fildelning

Ethernet är en typ av nätverksteknologi via kabel. Uppdragsgivarna kör simulatorn i ett lokalt nätverk med möjlighet för att koppla in enheten via Ethernet. För att kunna kommunicera med simulatorn används en Raspberry Pi 3 B+ som har en inbyggd fysisk Ethernet-port[8]. Fildelningen mellan Raspberry Pi och simulatordatorn sker med en *Sambaserver* som befinner sig på Raspberry Pi-enheten. Servern är uppsatt enligt en artikel från MagPi[9]. Den tillåter fjärråtkomst till nödvändiga filer över Ethernet och möjliggör sändning av en startsignal till programmet. *Sambaservern* har en mapp vid namn *share* uppsatt med fullständiga rättigheter, vilket möjliggör att vilken användare som helst kan läsa och modifiera de delade filerna. *Sambaservern* har också en användare uppsatt som

[7] Wikipedia. (2013), *Uart*, https://sv.wikipedia.org/wiki/Uart
[8] Raspberry Pi Foundation. *Raspberry Pi 3 Model B+,* https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
[9] Barnes R. (2017), *Samba: Set up a Raspberry Pi as a File Server for your local network*, https://magpi.raspberrypi.org/articles/samba-file-server

man måste logga in med på enheten som ska ha åtkomst till nätverksmappen. Filerna i mappen *share* beskrivs i tabell 1.

Tabell 1 Tabell över filer i den delade mappen *share*.

Namn	Typ av fil	Beskrivning
serialsave.py	script	Script som startas vid uppstart av launcher.sh, beskriven i 4.4.3.
launcher.sh	Bash-script	Script som startar serialsave.py automatisk vid uppstart, beskriven i 4.4.3.
exchange.txt	textfil	Fil som innehåller resultatet från den senaste tagningen.
exchange_history.txt	textfil	Fil som sparar resultatet från alla tagningar.
switch.txt	textfil	Innehåller som standard ”0”. Ett annat nummer agerar som startkod.
logs.txt	textfil	Innehåller eventuella felloggar.

4.4.3 Integrering

Avsnitt 4.4.1 beskriver den fysiska kopplingen av enheten och standarden *UART*. Avsnitt 4.4.2 beskriver i sin tur hur fildelningen mellan enheten och simulatorn. Integreringen av dessa görs av ett *python-script* vid namn serialsave.py. Scriptet lyssnar med korta mellanrum på ändringar i filen *switch.txt.* Om den innehåller något annat nummer än ”0” tolkas det som en startsignal. Filen *switch.txt* måste innehålla ett heltal, då scriptet försöker konvertera innehållet till ett värde av typen *int* (integer). När en startsignal tagits emot skickas signalen ”1” till kamerakortet över *UART-Interfacet*. Scriptet väntar sedan i upp till 20 sekunder på att få ett svar tillbaka. När den fått det läggs resultatet, en tidsstämpel och det nummer/id som angavs som startvärde i *switch.txt* i filerna *exchange.txt och exchange_history.txt*. *Exchangefilen* skrivs över med resultatet som är representerat i *json-format.* I *exchange_history.txt* läggs istället resultatet in på en ny rad. Programmet har två försök på sig att uppdatera filerna ifall filen skulle vara upptagen av simulatorn vid första försöket.

Om något oväntat sker under processen fortsätter programmet att köra. Flera olika tänkbara fel täcks av programmet som fortsätter att köra, men lägger då in ett felmeddelande med tidsstämpel och id i filen *logs.txt*. På detta sätt kan grundläggande felsökning göras utan att behöva ansluta till det grafiska interfacet på *Raspberry Pi:en*. Dessa felmeddelanden är beskrivna i tabell 2.

Tabell 2 Beskrivning av felmeddelanden som kan genereras i loggfilen *logs.txt.*

Meddelande	**Orsak**	**Lösning**
Error in UART communication. Following error message was generated: <Error>	Något gick fel antingen när en signal skickades till kamerakortet, eller när svaret skickades tillbaka	Kontrollera att kablarna är korrekt inkopplade. Starta om enheten genom att dra ut strömadaptern.
Could not write the following object to exchange.txt: {<data>}	Misslyckades att skriva resultatet till exchange.txt.	Ett till försök görs automatiskt efter 2 sekunder.
Received string from camera board is incorrectly formatted. Either the detection failed, or the cables are disconnected. Following string was received: <result>	Om strängen som medföljer meddelandet ser konstig ut kan ett fel skett i UART kommunikationen. Om strängen är tom har antingen detektionen misslyckats, eller så har signalen inte nått till kamerakortet.	Testa med en annan uppsättning lysdioder. Kontrollera att hållaren är korrekt placerad. Om problemet kvarstår, kontrollera att alla kablar är korrekt inkopplade. Starta om enheten.
Failed reading switch.txt. File is incorrectly formatted.	Programmet kunde öppna filen men inte läsa innehållet. Detta kan bero på fel textformat.	Kontrollera att texten som matas in har rätt format.
Failed to open switch.txt.	Programmet kunde inte öppna filen. Antingen är den upptagen eller så saknas rättigheter. Om filen skulle tas bort och läggas till från en annan dator kan rättigheter saknas.	Programmet gör automatisk ett nytt försök efter 1 sekund. Om det inte fungerar, kontrollera att filen har fulla läs- och skrivrättigheter.

Scriptet *serialsave.py* startar automatiskt vid uppstart av uppstartsscriptet *launcher.sh*. Detta innebär att det inte krävs någon inloggning på enheten för att starta scriptet. Om det mot förmodan skulle krascha, om enheten varken svarar eller genererar ett felmeddelande, räcker det därmed att starta om enheten.

4.4.4 Strömförsörjning

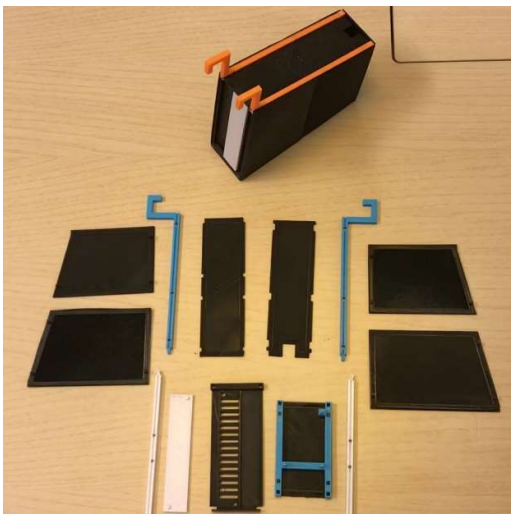
Enheten försörjs av en strömadapter inkopplad till *Raspberry Pi:en* som kan leverera upp till 2.5A vilket är fullt tillräckligt för både en *Raspberry Pi* och kamerakortet. Kamerakortet, *Maix M1 dock*, drivs i sin tur via USB inkopplad till *Raspberry Pi:en*.

4.5 Hållare

Syftet med hållaren var att på ett lätt och smidigt sätt kunna fästa maixpy-kortet vid reläskyddsterminalen och mörklägga kameran för att göra det så lätt som möjligt att se lysdioderna, samt att underlätta detektionen av lysdioderna med hjälp av ett fysiskt filter. För att bygga hållarna har 3d-printing använts. Alla delar är egendesignade i CAD-programmet Solidworks och är utskrivna i PLA-plast. Två kompletta hållare har byggts.

4.5.1 Konstruktion

Hållaren är modulärt byggd, vilket innebär att den består av flera mindre bitar som fästs i varandra med piggar och hål eller skenor. Det visas i figur 11. För att montera hållaren på reläskyddsterminalerna används två krokar som hakas i terminalens överkant.



Figur 11. Hållaren och dess beståndsdelar

Hållaren består av en grundstomme som håller ihop konstruktionen och paneler som klär in hållaren för att stänga ute ljus. Det finns även ett filter som ligger emot lysdioderna. Alla paneler som inte ska släppa in ljus är gjorda i svart plast då det är den färg som släpper igenom minst ljus. I hållarens bortre ände, den som är rakt emot dioderna, fästs kamerakortet med tre piggar på sin egen plattform. Detta ger lätt åtkomst till kamerakortet. Ovanpå kamerakortet monteras sedan en plastbit vars syfte är att trycka ner kameran för att se till att kamerans position alltid förblir densamma.

4.5.2 Filtret

Filtret består av två delar: En solid vit plastbit, som dämpar diodernas ljus, samt en svart avdelare som det vita filtret monteras på, som hjälper till att avskilja dioderna från varandra och på så vis underlättar detektionen.

5. Diskussion

Under projektarbetets tid har flera problem uppkommit som tvingat projektet in i en ny riktning. I denna del motiveras varför projektet utförts på det givna sättet från resultatdelen. Dessutom värderas resultatet.

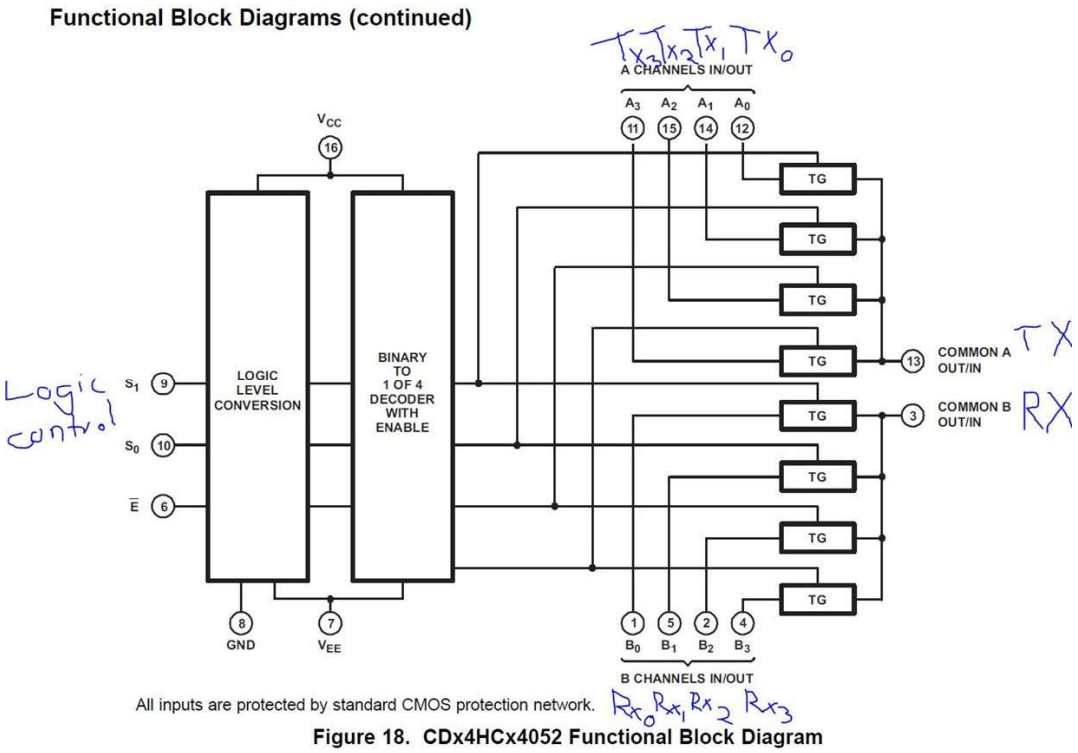
5.1 Kommunikation

Projektets val av kommunikation stod mellan de möjliga metoderna Wi-Fi eller Ethernet för kommunikation med uppdragsgivarnas system och seriell kommunikation med något av protokollen UART eller I2C mellan de interna enheterna. Anledningen till att Ethernetöverföring valdes istället för Wi-Fi är att cybersäkerheten i uppdragsgivarnas omgivning är mycket hög och en Wi-Fi router som sätts upp på platsen skulle kunna användas för att bryta sig in på deras nätverk av vem som helst. Resultatet av denna restriktion är att vi tvingades använda oss av en Ethernetkompatibel enhet. På så vis kan vi koppla upp oss på ett litet lokalt nätverk enbart för simulatorerna. Här fanns det en modul som tillhörde ett av de kort som vi valde mellan vid namnet "*Arduino Ethernet Shield*"[10]. Den var mycket stor och tung och det skulle behövas en för varje kort. Dessutom är det tillhörande kortet vid namn *Maixduino* mycket större i sitt grundutförande än *Maix M1 Dock* som blev det slutgiltiga valet.[11] Därför bestämde vi oss för att istället använda en *Raspberry Pi 3 B+* för möjligheten att koppla in flera enheter och ge de tillgång till kommunikation över nätverket.

För att kommunicera med Maix M1 korten används UART. Detta gränssnitt är egentligen inte optimalt för flera enheter, men det används i brist på att *Maix M1 dock* enheterna inte fungerar att använda som slavar för I2C eller SPI i praktiken trots att dokumentationen visar stöd för det.[12] [13]

[10] Arduino.cc. *Arduino Ethernet Shield V1,*
https://www.arduino.cc/en/Main/ArduinoEthernetShieldV1
[11] Sipeed. (2019), *Sipeed Maixduino Datasheet v.1.0,*
https://www.mantech.co.za/Datasheets/Products/MAIXDUINO-20100410A.pdf
[12] Sipeed wiki. (2021), *machine.I2C*,
https://wiki.sipeed.com/soft/maixpy/en/api_reference/machine/i2c.html
[13] Sipeed wiki. (2021), *machine.SPI*,
https://wiki.sipeed.com/soft/maixpy/en/api_reference/machine/spi.html

För att använda sig av flera enheter finns det flera olika tillvägagångssätt. Ett är att seriekoppla enheterna och att skicka mellan dessa vilket inte kräver någon ytterligare hårdvara. Däremot kräver denna lösning att man skapar ett eget protokoll för att rätt data ska sparas av rätt enhet. En annan lösning är koppla in alla enheter till en analog brytare som i sin tur är inkopplad till *Raspberry Pi:en*. Detta beskrivs av figur 12 där *Common A* och *Common B* är inkopplade till *Raspberry Pi:ens Rx-* och *Tx-linje.* De olika kamerakorten är i sin tur inkopplade till de olika kanalerna. Genom binära ingångar bestäms vilken av kanalerna som ska vara ansluten till huvudenheten. Denna lösning laborerade vi med och testade en analog multiplexer vid namn *CD74HC4052* som visas i figur 12, som har en konfiguration av 1:4 kanaler vilket innebär att fyra kamerakort kan kopplas till en Raspberry Pi.[14] Även denna lösning kräver mycket arbete då ett eget system för att styra vilken enhet som ska använda linjen krävs. Dessutom krävs det att göra ett eget kort med in- och utgångar samt kondensatorer. Att lösningen inte lämnade labbstadiet har att göra med att multiplexern inte lyckades leverera alla bitar i strängen korrekt. En vidareutveckling är att testa med andra multiplexrar eller någon annan typ av grind. Detta fick avgränsas på grund av tidsbrist.



Figur 12. Diagram över den testade multiplexern. Källa: https://e2e.ti.com/support/switches-multiplexers/f/388/t/527646.

[14]Texas Instruments. (1997, revised 2019), *CDx4HC405x, CDx4HCT405x High-Speed CMOS Logic Analog Multiplexers and Demultiplexers*, https://www.ti.com/lit/ds/symlink/cd74hc4052.pdf?ts=1620974511563&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCD74HC4052

5.2 Representation av data

Anledningen till att vi i detta projekt bestämde oss för att representera data som en textsträng är dels att vi med den kommunikationsmetod som vi använder kommer behöva överföra data med *UART*. Denna metod har i tidigare fall producerat felaktiga resultat med större mängde data på grund av den seriella överföringen. Därför ville vi hitta den kortaste möjliga metoden för att överföra data och bestämde oss då för att en karaktär per lampa fyllde kravet på bästa sätt. Vid senare testning visade sig detta inte vara ett problem för kommunikationen. Kommatecken lades till för läsbarhetens skull. Detta förenklade även arbetet för uppdragsgivarna då de lättare kunde läsa in resultatet med det valda formatet.

5.3 Detektion

Resultatet från detektionen är i huvudsak den avgörande faktorn för projektets framgång. För att kunna använda detektionsenheten krävs det att detektionen är tillräckligt tillförlitlig. Detektionens delar har arbetats fram under tid och detta avsnitt motiverar varför detektionsprogrammet fungerar på det sätt som det gör samt filtrets betydelse för att kameran över huvud taget kan ta upp relevant bildinformation.

5.3.1 Maixpys inbyggda färgdetektering

Anledningen till att *find_blob* metoden används är för att den specifikt är designad för att detektera färger inom ett område. Vid rätt förhållanden gör metoden ett mycket bra jobb med att separera färgerna åt och hittar alltid upplysta områden. Problemet med denna metod är däremot att de färgerna som används till detektionen är statiska så utifall ljusförhållanden förändras kommer färgområdet bli felaktigt. En möjlig lösning på detta problem är att använda en konstant diod som referenspunkt och anpassa ljusnivåerna efter den. Denna metod skulle däremot sannolikt bli mycket komplex och ingen lättillgänglig sådan diod finns på det reläskydd arbetet sker emot.

Det framkom efter en tids arbete att find_blob metoden inte räckte på egen hand. Eftersom resultatet blev så varierande undersöktes olika orsaker till detta. Något som under en stor del av projektet förbisetts var kamerans olika inställningar. Den varierande ljusstyrkan på bilden berodde på en för hög kamerafrekvens. Dessutom uppträdde ett svårförklarligt fenomen trots att vi lagt till ett binärt filter och det morfologiska filtret *top hat*: dioderna ändrade färg beroende på vilka andra dioder som lyste. Detta fenomen kunde härledas till standardinställningen för vitbalans som ändrades automatiskt beroende på vilka dioder som var tända. Detta löstes genom att stänga av den automatiskt anpassade vitbalansen.

En sak som orsakat problem vid detektering genomgående under projektet är de röda och gula diodernas överlappande i färgområdet *LAB*. Med det menas att delar av både röda och gula dioder har en mycket liknande färg. Att de gula dioderna i grunden har en orange nyans är en orsak, en annan att de lyser olika starkt och att det fysiska och programmatiska filtret får dessa att likna varandra ännu mer. Detta ledde till att röda dioder i vissa fall detekterades som både röda och gula. För att lösa detta fanns redan tillgänglig funktionalitet i ett av biblioteken tillhörande *Maixpy*. Genom att köra metoden *find_blobs()* och specificera två färgspann letas *blobs* precis som innan. Skillnaden är att om en stor röd *blob* och en liten gul *blob* hittas nära varandra slås de ihop till en röd *blob.* Detta gör att metoden i första hand tar hänsyn till diodens huvudsakliga färg och eliminerar eventuella feldetekteringar.

I slutskedet av projektet har detekteringen presterat bra. Av de tester som gjorts manuellt av oss har detekteringen varit tillförlitlig och säker, utan variation mellan olika tagningar. I vissa fall har mätningarna ogiltigförklarats av anledningarna förklarade i 4.3.2. Målet var att vid detta stadie gjort mer omfattande tester, men eftersom integreringen dröjt och problem uppstått med detta har inga fullständiga tester av detektionen i sitt slutskede kunnat göras.

Programmets placering av de detekterade dioderna fungerar mycket bra, speciellt när antalet tända dioder närmar sig 15 stycken. Detta eftersom konstruktionen alltid har samma position i det som på skärmen representeras i x-led och lysdioderna då får samma mönster. Repetitionen sker eftersom det ger möjligheten att utesluta enstaka felaktiga mätningar vilket ökar robustheten på systemet. Slutligen så körs detektionen endast på kommando eftersom denna enhet ska ha möjlighet att vara påslagen utan att detektera.

5.3.2 Filter

Till kameran krävdes ett filter på grund av den ljusstyrka lysdioderna hade. Den höga ljusstyrkan hade två oönskade effekter. Dels det att diodernas färg blev mycket vit vilket gjorde det svårare att detektera dem, dels det att färgerna hade stor påverkan på varandra med sitt sken vilket skapade olika färgnyanser på närliggande dioder. Dessa effekter gjorde det omöjligt att använda färgdetekteringsverktygen i *Maixpy* med universella tröskelnivåer för färgskillnad. Filtrets öppningar är relativt avlånga eftersom produkten ska kunna monteras med horisontell marginal och fortfarande fungera. På så sätt elimineras nödvändigheten av ytterligare onödiga fästningsmoduler. Tjockleken på filtret bestämdes genom testning.

5.4 Val av hårdvara

Valet av hårdvara bygger mycket på kraven på kommunikation som finns detaljerat beskrivet i kommunikationsdelen. Valet av Maix-enheter för detektering beror av flera faktorer. Den främsta

fördelen är storleken. Mikrodatorn är tillräckligt kraftfull för projektets krav och samtidigt väldigt kompakt. I jämförelse med en Raspberry pi tar kortet mycket mindre anspråk i utrymme och går således enkelt att montera i en kompakt konstruktion.

En annan anledning till detta val av hårdvara är de färdiga bibliotek som finns för Computer Vision i maixpys variant av OpenMV. Ett av dessa bibliotek innehåller färdig funktionalitet för den bildbehandling som projektet kräver.

5.5 Hållare

5.5.1 Förundersökning

Det första vi behövde ta reda på när det kom till hållaren var hur kameran och nödvändiga kort skulle sitta för att ta så lite plats som möjligt, samtidigt som kameran ser allt det den behöver se. Vi behövde därför testa vart kameran skulle sitta för att precis få med alla lysdioder i sitt synfält. Den första 3D-printade delen vi skrev ut var en enkel pinne som man kunde fästa i reläskyddsterminalen, och som hade ett fäste för kameran. Detta gjorde vi för att vi ville vara säkra på att kameran skulle sitta på rätt ställe. När vi visste var kameran skulle sitta var nästa problem hur själva hållaren skulle se ut. Vi visste att vi skulle behöva skärma in kameran för att alltid ha samma ljusförhållanden och vart kameran skulle sitta. Det vi inte visste var hur vi skulle fästa hållaren i reläskyddsterminalen och hur vi skulle skriva ut hållaren. Ganska snart visste vi att vi skulle behöva göra hållaren modulär. Dels skulle inte hela hållaren få plats i 3D-skrivaren, dels skulle det bli näst intill en omöjlig modell att skriva ut.

Nu visste vi ungefär hur den skulle se ut men inte hur vi skulle göra för att dela upp den i mindre komponenter. Dels behövde vi lista ut ett sätt att fästa de olika komponenterna i varandra, dels hur stora komponenterna skulle vara samt hur de olika komponenterna skulle se ut. Redan tidigt kom vi fram till att hålla komponenterna små och det var främst av en anledning: Om något inte passade eller gick sönder skulle det vara enkelt och snabbt att skriva ut en ny. Vi funderade på olika lösningar för att kunna fästa de olika delarna i varandra. I början var vi inne på någon slags låsmekanism. Hur den skulle fungera var inte riktigt bestämt. Det vi kom fram till var dock en lösning med piggar och hål, där piggarna precis skulle passa i hålen. Har vi då flera hål, alternativt piggar i bitarna skulle bitarna sitta på plats men ändå vara möjliga att sära på. För att få det att fungera behövde vi hitta mått på hålen respektive piggarna så att de skulle passa i varandra och vara lagom stora men samtidigt tåliga nog för att hålla när man stoppar in och tar ur de flera gånger. För att ta reda på ett passade mått skrev vi ut små test med lite olika mått vi trodde kunde vara lagom. Vi testade sedan dessa med varandra för att hitta det mått vi sökte. Vi valde att göra piggarna och hålen fyrkantiga, för att undvika att delarna skulle kunna snurra runt när de fästs i varandra.

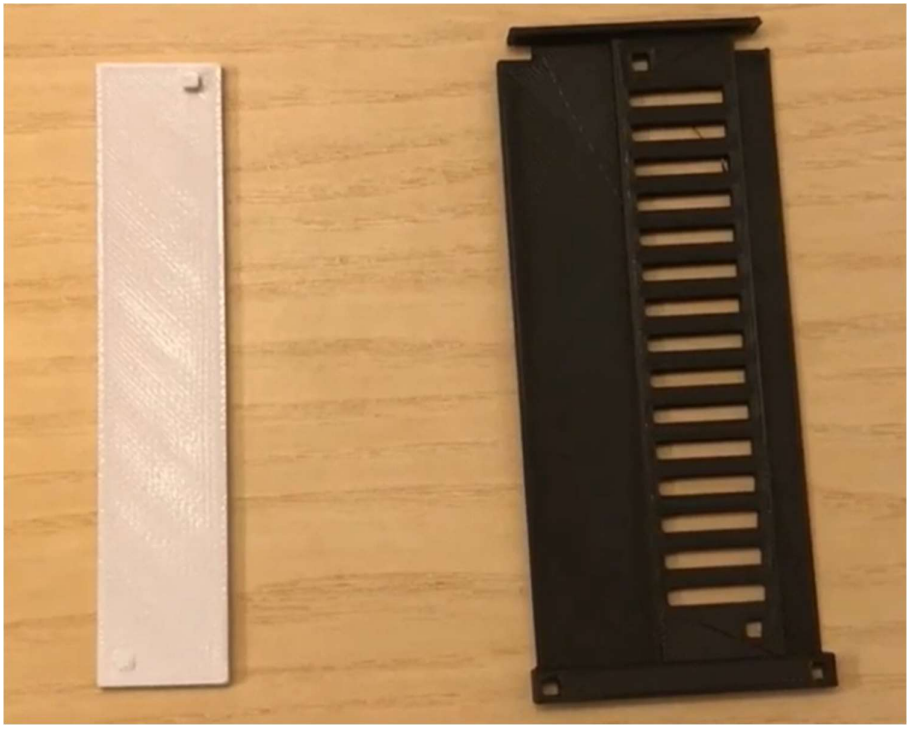
När vi var nöjda med hålen och piggarnas dimensioner gjordes den första prototypen. Prototypen var väldigt fyrkantig med bara räta vinklar. Anledningen var att det är lättast att 3d-printa sådana delar. För bästa resultat vill man ha så simpla former som möjligt vilket underlättar för skrivarna. Den första prototypen var inte så stabil, vilket vi hade räknat med. För att dra ner på materialåtgången valde vi att göra stommen minimalistisk då vi visste att panelerna som skulle komma senare skulle bidra till strukturens hållfasthet.

5.5.2 Prototyp 1 och 2

De första prototyperna skrev vi ut med plast i olika färger. Färgerna hade ingen påverkan på det vi ville testa då, vilket var den övergripande designen. När vi gjorde prototyper med paneler för att kunna börja testa detektionen ordentligt märkte vi dock att färgen spelade stor roll. Plast av exempelvis grå färg stängde inte ute tillräckligt mycket ljus och färgad plast färgade av ljusets och förvrängde lysdiodernas färger. Det gjorde att ljusförhållanden inne i hållaren kunde påverkas av yttre faktorer, såsom datorskärmar, tid på dygnet eller taklampor. Lösningen på det problemet var att skriva ut panelerna i svart plast, den färg som släpper igenom minst ljus. Tjockleken på panelerna spelade till en gräns inte en betydande roll för ljusinsläppet, då svart plast stänger ute nästan allt ljus även vid tunna tjocklekar.

5.5.3 Filtret och avdelaren

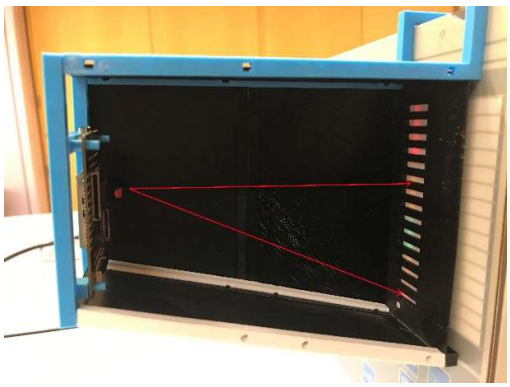
Det motsatta gällde för filtret vi använde för att dämpa lysdiodernas ljus. Det behövde dämpa ljuset men inte stänga ut det helt. Det fick inte heller påverka lysdiodernas färg. Därför behövde den skrivas ut helt i vit plast. Tjockleken på filtret spelar en viktig roll för hur mycket ljus från lysdioderna som lyser igenom. Rätt tjocklek testade vi genom att skriva ut små vita plastrektanglar i olika tjocklekar och testade de på lysdioderna. Det var en svår balansgång, då de röda och gröna dioderna lyser starkare än de gula. För tjockt filter, och de gula syns inte. För tunt, och de röda och gröna blir för starka. Med ett filter på plats stötte vi på ett problem med detektionen. Vissa färgkombinationer kunde “flyta ihop” eller störa varandra när de var nära varandra, vilket antingen kunde förvränga lysdiodernas färger för kameran eller helt störa ut andra dioder så att de inte gick att detektera. Detta löste vi genom att skapa en avdelare. Vi gjorde den som en panel med rader lite mindre än lysdiodernas diameter för att skilja de från varandra, som visas i figur 13.



Figur 13. Bild på plastfilter (vänster) och avdelare (höger). Källa: Egengjord film.

Filtret och avdelaren var något som vi testade ett flertal varianter utav. Den vi kom fram till i slutändan var en avdelare som fästs i hållaren, och ett filter som monteras på avdelaren. Problemet som uppstår med denna design är att ljus från dioderna kan spridas i filtret, vilket gör att man inte kan garantera att de olika dioderna är helt opåverkade av varandra. Det som hade varit att föredra är en design där varje diod har en egen vit filterpanel, som skiljs åt från resten med den en mer omfattande svart avdelare. Vi experimenterade lite med denna design, men beslutade att det skulle bli för komplext och att nyttan inte var tillräcklig för att vara värd den tiden som skulle behövts.

Ytterligare ett problem som vi inte hade tänkt på var kamerans position. På grund av kamerans position blir avståndet från de övre och lägre till kameran större än till de dioderna som är rakt framför kameran, se figur 14. Detta hade två effekter: Dels så blev avståndet mellan dioderna glesare ju längre ut åt sidan de befann sig, dels så lyste de yttre dioderna svagare då deras ljus delvis blockerades av avdelaren. Avståndet mellan dioderna gick att lösa i genom att modifiera koden i kamerakortet, men de låga ljusnivåerna var desto svårare. Dioderna lyste så svagt att det inte var



Figur 14. Pilarna visar de olika långa avstånden till dioderna

möjligt för kameran att detektera dem, så en modifiering av avdelaren krävdes. Lösningen var att göra spalterna längst upp och längst ner aningen större än de i mitten, vilket löste problemet och möjliggjorde detektionen av alla dioder, oavsett position.

5.5.4 Prototyp 3

Vid det här laget hade vi fått fram en duglig prototyp, men vi såg fortfarande ett flertal utvecklingsmöjligheter. En av de främsta var att göra botten av hållaren sned. Ytan under kameran spelar ingen väsentlig roll då det är utanför kamerans synfält, så för att dra ned på materialåtgången och minska hållarens storlek valde vi att byta ut den räta vinkeln mot en trubbig. Det var även i detta stadie som vi satte ihop flera bitar till en för att skapa kamerans plattform, vilket underlättade mycket när man skulle ta av eller sätta på kameran. Vi lade ned mycket tid på att få alla mått och passformer bra med denna prototyp, för att se till att hållaren skulle bli så praktisk och lätthanterlig som möjligt.

5.6 Integrering

Relativt sent i projektets skede påbörjades integreringen av enheten med simulatorsystemet. Det första steget som togs var att sätta upp ett delat filsystem och besluta om formatet på filerna som delas. På detta sätt kunde uppdragsgivarna förbereda sig för att arbeta med filer av dessa format.

Ett första problem i början av integreringen var att överhuvudtaget kunna kommunicera med enheten. För att göra detta behövdes en statisk IP-adress sättas på enheten, vilket enkelt kunde göras i användargränssnittet. Trots att enheten var synlig på nätverket och gick att pinga till kunde den delade mappen inte kommas åt av simulatordatorerna. Uppdragsgivarna arbetade på sin sida med att möjliggöra att enheten skulle bli synlig.

Ett annat problem var att kunna komma åt filerna. För att köra automatiserade tester behöver uppdragsgivarna från sin programvara kunna komma åt filinnehållet. Detta problem visade sig vara svårare än väntat och överväganden gjordes kring att byta fildelningssystem till NFS istället för Samba eller att sätta upp ett fildelningssystem på någon av datorerna istället för på enheten. Efter en lång tids felsökande av uppdragsgivarna slutade det med att det ändå fungerade med Samba-servern som behölls.

De flesta ändringar i detektion och av filter gjordes i samband med att enheten kopplades in i den verkliga miljön som hade starkare ljusförhållanden än miljön på skolan. Detta ledde till att enheten var tvungen att bli mer ljusisolerad. Även detektionen visade sig fungera varierande länge, och var framförallt beroende av placeringen av hållaren. De förändringar som gjorts av filter och detekteringskod har till stor del rättat till detta, men inga omfattande tester har ännu kunnat göras med den slutgiltiga lösningen.

6. Slutsats

Automatisering av arbetsprocesser sker idag för fullt och datorseende har börjat användas för att kunna utföra mer komplexa uppgifter av datorer. Detektion av lysdioder är i grunden ett relativt enkelt detektionsproblem, men för att lösa det krävs flera delar: datorseende, hårdvara, kommunikation samt konstruktion av hållare. Framförallt har det framkommit vikten av kunskap om ljus och kameror, då filtrering och separering av ljus samt kamerainställningar varit viktiga för en fungerande detektion. Även vikten av att tidigt påbörja integration med befintliga system har blivit en slutsats av projektet, då integrationen inte är helt klar i skrivande stund. Trots de olika problemen som uppstod har en fungerande enhet skapats som uppfyller projektets syfte, att automatisera avläsningen av lysdioder på reläskydd.

Referenser

Alkiek, K. (2018), *Traffic Light Recognition – A Visual Guide*, https://medium.com/@kenan.r.alkiek/https-medium-com-kenan-r-alkiek-traffic-light-recognition-505d6ab913b1

Sipeed. (2020), *Specifications*, https://cn.dl.sipeed.com/shareURL/MAIX/HDK/Sipeed-M1&M1W/Specifications

Sipeed wiki. (2021), *Introduction to MaixPy documentation*, https://wiki.sipeed.com/soft/maixpy/en/

Sipeed wiki. (2021), *image (machine vision),* https://wiki.sipeed.com/soft/maixpy/en/api_reference/machine_vision/image/image.html

Sipeed wiki. (2021), *MaixPy IDE installation and use*, https://wiki.sipeed.com/soft/maixpy/en/get_started/env_maixpyide.html

Wikipedia. (2013), *Uart*, https://sv.wikipedia.org/wiki/Uart

Raspberry Pi Foundation. *Raspberry Pi 3 Model B+,* https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

Barnes R. (2017), *Samba: Set up a Raspberry Pi as a File Server for your local network*, https://magpi.raspberrypi.org/articles/samba-file-server

Arduino.cc. *Arduino Ethernet Shield V1,* https://www.arduino.cc/en/Main/ArduinoEthernetShieldV1

Sipeed. (2019), *Sipeed Maixduino Datasheet v.1.0,* https://www.mantech.co.za/Datasheets/Products/MAIXDUINO-20100410A.pdf

Sipeed wiki. (2021), *machine.I2C*, https://wiki.sipeed.com/soft/maixpy/en/api_reference/machine/i2c.html

Sipeed wiki. (2021), *machine.SPI*, https://wiki.sipeed.com/soft/maixpy/en/api_reference/machine/spi.html

Texas Instruments. (1997, revised 2019), *CDx4HC405x, CDx4HCT405x High-Speed CMOS Logic Analog Multiplexers and Demultiplexers*, https://www.ti.com/lit/ds/symlink/cd74hc4052.pdf?ts=1620974511563&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCD74HC4052