NTNU

Kunnskap for en bedre verden

Bacheloroppgave

IR303612 Bacheloroppgave

3D-tilpasset støtteskinne ved håndleddsbrudd

10016, 10029

Totalt antall sider inkludert forsiden: 89

Ålesund, 20.05.2019

NTNU

Kunnskap for en bedre verden

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.**	☒
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver**	☒
5.	**Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31**	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☒

Publiseringsavtale

Studiepoeng: 20

Veiledere: Webjørn Rekdalsbakken og Paul Steffen Kleppe

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven §2).
Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage HiM med forfatter(ne)s godkjennelse.
Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU i Ålesund en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	☒**ja**	☐**nei**
Er oppgaven båndlagt (konfidensiell)? (Båndleggingsavtale må fylles ut) - Hvis ja: **Kan oppgaven publiseres når båndleggingsperioden er over?**	☐**ja** ☐**ja**	☒**nei** ☐**nei**
Er oppgaven unntatt offentlighet? (inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13)	☐**ja**	☒**nei**

Dato: 20.05.2019

FORORD

Denne rapporten er en besvarelse på det avsluttende prosjektet i studiet automatiseringsteknikk ved NTNU (Campus) Ålesund. Oppgaven var å forbedre dagens behandlingsmetoder for håndleddsbrudd. Dette var en videreføring av en tidligere bacheloroppgave som tok i bruk en skanner for å fremstille en 3D-modell av et håndledd for så å behandle modellen i CAD. Det viste seg at denne måten var for tidkrevende og i tillegg var avhengig av mye redigering i CAD for å fremstille en grei replikasjon. Det ble derfor bestemt at denne oppgaven skulle videreføres, og det skulle undersøkes hvilke alternativer som var aktuelle.

Denne rapporten presenterer et automatisert system som fremstiller 3D-modeller klare for 3D-print ved hjelp av fotogrammetri og 3D-modellering.

Vi vil takke Webjørn Rekdalsbakken og Paul Steffen Kleppe ved NTNU Ålesund som rådgivere og faglige kontaktpersoner for veiledning. I tillegg vil vi takke Anders Sætersmoen og Øyvind Andre Hanken for assistanse med bestilling og råd til oppsett av utstyr.

Takk til NTNU Ålesund for lokale, utstyr og materialer som har gjort det mulig å gjennomføre denne oppgaven.

INNHOLD

SAMMENDRAG

Denne rapporten omhandler bacheloroppgaven: “3D-tilpasset støtteskinne ved håndleddsbrudd”. Oppgaven tar for seg dagens behandlingsmetode, problematikken rundt denne og hvordan den kan forbedres. Hensikten er å bevise at det er mulig å automatisere behandlingen ved hjelp av teknologi som fotogrammetri og 3D-printing.
Vi vil ta for oss alternativene og vurderingene vi gjorde underveis før vi valgte hvilken retning vi ville ta prosjektet. Videre ser vi på hvordan vi realiserte prototypen, hvilke resultater den har gitt, hva som kunne vært gjort annerledes og problematikken rundt denne prototypen.

Resultat viser at en kan oppnå rette geometriske forhold mellom objekt og 3D-modell ved bruk av Raspberry Pi og tilhørende kameramodul. Sett i sammenheng med oppgaven tilfredsstiller ikke modellen kravene for å kunne 3D-printes.

Rapporten inneholder nødvendig bakgrunnsteori, hvilke metoder vi har brukt for å optimalisere 3D-modellen, endelig resultat, drøfting av resultat og en konklusjon på prosjektet. Vedlagt ligger også referanser og kildekoden vi har brukt.

TERMINOLOGI

Begreper

Switch	En enhet som kobler sammen flere enheter over et datanettverk
Socket	Et endepunkt for levering/mottakelse av data over datanettverk
Pixel	En pixel er et av mange små punkter som til sammen utgjør en representasjon av et bilde.

Forkortelser

2D	Todimensjonal
3D	Tredimensjonal
AC	Alternating Current
CAD	Computer Aided Design
DC	Direct Current
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
IDE	Integrated Development Environment
IP	Internet Protocol
LED	Lysemitterende diode
PLA	Polyactic Acid
PWM	Pulse Width Modulation
RGB	Red Green Blue
SD card	Secure Digital card
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus

Figurer

Tabeller

1 INNLEDNING

1.1 Bakgrunn for oppgaven

Håndleddsbrudd er den vanligste formen for brudd i verden, og bare i Norge forekommer det over 15 000 tilfeller årlig. **Feil! Fant ikke referansekilden.** Det overordnede målet for b acheloroppgaven, og den gitt tidligere, er å forbedre måten håndleddsbrudd behandles på ved å automatisere gipsprosessen. Dagens behandlingsmåte er både tid- og ressurskrevende, og er sådan utdatert sett i sammenheng med dagens trender innenfor automatisering og 3D-teknologi.

1.2 Oppgaven

"Ved armbrudd trengs det støttegips. Dette er en tidkrevende prosess. Prosessen kan automatiseres ved å "skanne" pasientens arm, lage en 3D-modell av armen med støtteskinne og deretter 3D-printe støtteskinnen. Dette skal kunne gjøres i en sammenhengende operasjon."
-Forslag til bacheloroppgave 2018

Spesifikt er målet å utvikle en prototype av en maskin som tar bilder av et håndleddsbrudd og deretter produserer en 3D-modell av en støtteskinne innen 15-20 minutter som er klar til bruk.

Oppgaven ble delt på to grupper, hvor vår gruppe hadde ansvar for å sette opp alle komponentene (inklusiv elektronikk), programvarer (server/klient, fotogrammetri) og testing av forskjellige parameter for å optimalisere 3D-modellen. Den andre gruppen bestod av tre POD (produkt og systemdesign) studenter som hadde ansvar for å bygge den fysiske delen av prototypen, samt bruke 3D-geometrien de fikk av oss til å lage 3D-modell av en støtteskinne som kunne skrives ut umiddelbart.

1.3 Dagens metoder

Går en til legen med brudd i dag kartlegges bruddet ved røntgen. Legen vurderer så bildet og gjennomfører en klinisk undersøkelse. Lokalisering av brudd, type, grad av feilstilling og helse er faktorer som avgjør hvilken behandlingstype, operativ eller konservativ, som brukes. Operativ, som navnet foreslår, involverer inngrep for å rette på bruddet og konservativ lar bruddet gro naturlig. De fleste brudd behandles konservativt med gips. Det er hovedsakelig to måter som brukes til å gipse ved brudd, tradisjonell kalkgips og moderne plastgips.

1.3.1 **Kalkgips**

Første steg er å legge en tynn strømpe over armen, som vist i figur 1.1, så et tynt lag med vatt vist i figur 1.2. Dette gjøres for å hindre direkte kontakt mellom gips og hud, samt for å gjøre det mer komfortabelt. Videre fuktes kalk-kluter i vann og påføres området som skal beskyttes, vist i figur 1.3. Dette kan være relativt krevende på grunn av at kalk-klutene er ømfintlige og kan lett formes, noe som kan føre til ubehag for pasienten hvis de ikke formes i forhold til bruddet. Figur 1.4 viser siste steg, som er å bandasjere og innen 24 timer størkner gipsen.

Figur 1.1: Strømpe

Figur 1.2: Vatt

Figur 1.3: Kalk-klut

Figur 1.4: Bandasje

1.3.2 **Plastgips**

Plastgips brukes som regel etter en hevelse har gått ned, og sjelden direkte etter brudd. Plastgipsen er nyere enn kalkgips, og som følge er det færre leger som vet hvordan den brukes. Prosedyren er svært lik den med kalkgipsen. Det brukes først en strømpe, deretter en plastbandasje over hele armen som størkner i løpet av 24 timer, vist i figur 1.5. Det legges også ekstra støtte i området rundt bruddet som vist i figur 1.6.

Figur 1.5: Plastbandasje

Figur 1.6: Ekstra støtte

1.4 Kravspesifikasjoner

Prototypen skal kunne produsere en pålitelig støtteskinne med like gode eller bedre egenskaper enn dagens gips. Prosessen skal ta 15-20 minutter.

Attributtene til dagens gips er:

Passform og komfort
Kvaliteten på en gips kan i stor grad variere avhengig av hvem som påfører gipsen. Om kvaliteten på gipsen er dårlig kan det medføre trykksår i huden, smerter og i verste tilfelle muskel- og nerveskade. Andre irritasjonsmomenter ved gips er kløe, lukt, tyngde, upraktisk og man kan ikke dusje.

Styrke og fleksibilitet
Det viktigste poenget med å gipse er å låse hånden mot bevegelse. Både kalk- og plastgips møter disse kriteriene, men låser samtidig hånden totalt over lang tid.

Ressurser
Håndleddsbrudd er det vanligste bruddet i Norge, og selv om materialkostnaden er lav er behandlingen tidkrevende og høy personell kostnad blir en følge av dette.

Krav som stilles til støtteskinnen:

Passform og komfort
En 3D-printet støtteskinne skal passe håndleddet slik at det i tilstrekkelig grad beskytter og låser håndleddet for bevegelse, samtidig skal den være komfortabel nok til at den ikke irriterer

Styrke og fleksibilitet
Den må opprettholde styrken og fleksibiliteten til tradisjonell gips, men også kunne tas av og på ved behov.

Ressurser

Materialkostnadene må være på lik linje med gips.

Krav som stilles til prototypen av maskinen:

15-20 minutter behandlingstid, fra bildene blir tatt til støtteskinnen er klar. Skal dette være en reell konkurrent til dagens metode må behandlingstiden være kortere. Selv om dette er en autonom prosess er det rimelig å anta at det vil være personell tilstede under hele prosessen for å kontrollere og overvåke at resultatet blir godt nok.
Komfortabel for pasient under behandling. Prototypen skal ikke sette pasienten i en ukomfortabel posisjon, og i tilfellet over en veldig begrenset tidsperiode.
Brukervennlig programvare. Det er viktig at programvaren, gjennom GUI, er så enkel at alt personell kan ta den i bruk.

2 TEORETISK GRUNNLAG

2.1 Programvare

2.1.1 Python

Python er et objektorientert programmeringsspråk startet av Guido van Rossum i 1989. Det er et tolket språk, som betyr at man ikke trenger å kompilere det for å kjøre programmet. Dette betyr at en kan forandre kode og teste det umiddelbart, men at språket er tregere enn kompilerte språk som eksempelvis C. Python har en lettlest og klar syntaks som er svært brukervennlig.

2.1.2 Arduino IDE

Arduino IDE er et java-basert programmeringsverktøy for programmering og opplasting av kode til Arduino mikrokontrollere. IDE'en kjører på Windows, MAC og Linux.

2.1.3 Thonny IDE

Thonny er et utviklingsmiljø for Python som er integrert i Raspberry Pi sin programvare.

2.1.4 Notepad++

Notepad++ er et åpent, fritt og gratis redigeringsverktøy for tekst- og kildekode laget til bruk med Microsoft Windows operativsystem.

2.1.5 Nettverksprogrammering

Nettverksprogrammering er en type programmering som involverer kommunikasjon over et nettverk. Ofte er dette kommunikasjon mellom enheter koblet til en switch. De fleste programmeringsspråk støtter denne type kommunikasjon ved hjelp av egne bibliotek som tar i bruk "sockets" for kommunikasjon. Typisk foregår dette i form av en server/klient konstellasjon.

2.1.6 Fritzing

Fritzing er et fritt og gratis program for å tegne kretsskjema for prototyper laget med Arduino. Det kan også brukes til å illustrere kretser med Raspberry Pi.

2.1.7 Fotogrammetri programvare

Fotogrammetri programvare tar i bruk prinsippene bak fotogrammetri (se avsnitt 2.3.2 fotogrammetri) for å fremstille en 3D-modell fra en samling av bilder. I tillegg til Autodesk ReCap Photo og Agisoft PhotoScan Pro har 3DF Zephyr og Regard 3D blitt tatt i bruk. De to sistnevnte er mindre kompliserte fotogrammetriprogrammer.

2.1.7.1 Autodesk ReCap Photo

Autodesk sin programvare ReCap Photo lager 3D-modeller fra bildeserier tatt av objekter både på nært hold og med flyfoto. Programmets algoritmer og funksjoner er gjemt, og gir bruker veldig lite mulighet til å påvirke resultatet. Bilder blir lastet inn i programmet, deretter lastet opp til en sky hvor prosesseringen foregår. Videre kan en ferdig 3D-modell hentes fra skyen, og en kan justere overflate-gitteret i ReCap Pro.
Programvaren kan kjøpes med lisens, men det tilbys en gratis versjon for studenter. Gratisversjonen gir samme muligheter som den fulle versjonen, men det går på bekostning av prosesseringstiden. Ved prosessering i skyen blir prosjekter fra brukere uten lisens nedprioritert, og det er derfor varierende hvor lang tid det vil ta.

2.1.7.2 Agisoft PhotoScan Pro

Agisoft PhotoScan Pro er et komplett program for fotogrammetri, spesielt utviklet for undersøking og kartlegging av geografiske områder. Programmet kan automatisk generere tette punktskyer og 3D-modeller fra en serie med stillbilder. Programmet bruker algoritmer utviklet av Agisoft selv og kildekoden er ikke tilgjengelig.
Agisoft PhotoScan prosesserer bilder raskt og presist, med opp til 1mm nøyaktighet på modeller fra bilder tatt på nært hold og 3cm på flyfoto. Agisoft kan håndtere tusenvis av bilder samtidig og alle prosesser foregår lokalt på datamaskinen. Brukergrensesnittet er intuitivt og lite forkunnskap trengs. Likevel tilbys det store valgmuligheter til å skreddersy

prosesseringen av bildene for ønsket utbytte, som passer for viderekomne innen fotogrammetri. Programmet generer også en detaljert rapport etter prosesseringen.
En annen stor fordel med denne programvaren er at den har innebygd Python-script for automatisering av prosesser, dette kan for eksempel være til nytte om programmet skal brukes til å lage mange modeller av samme type. Programmet koster $3,499, men er tilgjengelig med gratis prøvelisens i 30 dager.

2.1.8 **GUI (PyQt)**

GUI, graphical user interface, er brukergrensesnittet som gjør det mulig for bruker å samhandle med en datamaskin ved bruk av tastatur, datamus, skjerm og lignende enheter over grafiske ikoner. Dette kan ses som en motsetning til tekstbasert brukergrensesnitt hvor man taster inn kommandoer. PyQt er programvare laget spesielt for design av GUI til python applikasjoner.

2.2 Fotogrammetri

Fotogrammetri er læren om måling ved hjelp av fotografier. Målingene som blir foretatt er form, størrelse og beliggenhet av objektet som er fotografert. Fotogrammetri er mest brukt for å kartlegge geografiske områder, men kan benyttes til flere formål. For å foreta målinger av et objekt må det ut ifra en bildeserie finnes felles punkt, så rekonstrueres en 3D-modell. Bildeserien må inneholde flere overlappende bilder av objektet eller området som skal måles. For å skape en riktig rekonstruksjon må fotoapparatets indre geometri og ytre orientering være kjent. Med indre geometri menes formen og dimensjonen til apparatet, brennvidde og synsvinkel er viktig. Ytre orientering betyr apparatets plassering i rommet.

Et av bruksområdene til fotogrammetri er rekonstruksjon av objekter i form av digitale 3D-modeller. Det finnes mange forskjellige programmer for akkurat dette, og de fleste bygger på de samme algoritmene. For å lage en 3D-modell i disse programmene trengs en bildeserie med bilder tatt fra alle vinkler av objektet. Hvert område av objektet må være dekke av minst to overlappende bilder. Bildeserien må gjennom en rekke operasjoner før modellen er rekonstruert. Operasjoner som kan bli brukt er: SIFT, buntjustering, tette punktskyer, rekonstruksjon av overflate og kartlegging av tekstur [5].

2.2.1 SIFT

Scale-invariant feature transform (SIFT) er en algoritme brukt for å finne og beskrive kjennetegn i bilder. SIFT er den mest kjente algoritmen, men det finnes flere algoritmer som gir veldig like resultat. SIFT er patentert og det gjør at den ikke kan brukes i kommersielle produkter.

Algoritmen starter med å bestemme nøkkelpunkter i objektet fra et sett med referansebilder. Disse får en unik identifikasjon. Punktene finnes ved å se etter lokale maksimum- og minimumpunkter i Difference of Gaussian (DoG) av et bilde. Vektorer blir brukt til å definere nøkkelpunktene ut ifra nabopunktene.
DoG er resultatet av å trekke en uklar versjon av et bilde fra en mindre uklar versjon av samme bildet. De uklare versjonene av bildet blir skapt ved å bruke et filter som heter Gaussian blur. Filteret bruker en Gaussisk funksjon for å kalkulere ny verdi for hver pixel i bildet. DoG fremhever på denne måten detaljer i bildet.
Gaussisk funksjon i to dimensjoner:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

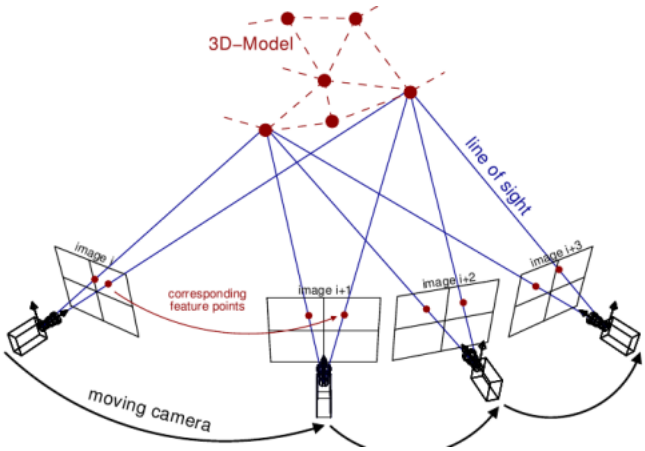
der er standardavviket i den gaussiske fordelingen.

Når nøkkelpunktene er bestemt blir de hentet ut og sammenlignet med kjennetegn og detaljerte områder som ligger i en kjent database med forhåndsgodkjente referansepunkter. Disse er manuelt definerte. Nøkkelpunktene blir brukt til å gjenkjenne et spesifikt punkt i flere bilder og på den måte knytte bildene sammen. Flere nøkkelpunkter gir mer nøyaktig resultat. Den viktigste fordelen med SIFT er at den kjenner igjen nøkkelpunkter selv om det finnes rotasjon, strekking, mindre forvrenginger og lysendringer i en bildeserie.

Denne metoden klarer å gjenkjenne og skille nøkkelpunkter på veldig detaljert nivå, men kan likevel få problemer med mønster i bildet. Hvis mønsteret er repeterende over et større område vil algoritmen gjenkjenne ett nøkkelpunkt flere plasser i samme bilde. Punktene blir ikke unike og dette vil gi feilaktige resultater.

2.2.2 Buntjustering

Dette er en metode innen fotogrammetri som er brukt til å estimere struktur i tre dimensjoner fra bilder av to dimensjoner. Ved å gjenkjenne en enkelt samling nøkkelpunkter i flere bilder kan man finne ut hvilken plassering og retning bildet ble tatt fra. Samlingen av punkter må være todimensjonal, altså inneholde minste tre punkter som ikke ligger på linje. Metoden som brukes for dette ligner på kamerakalibrering. Avstand mellom punktene innad i gruppen, og plassering i forhold til hverandre beregnes i hvert bilde. Disse målingene fra hvert bilde blir sammenlignet for å finne endringer. Endringer i avstand og posisjon kommer av rotasjon, skalering og forvrenging. Ved å bruke denne kunnskapen, samt informasjon om kameraenes egenskaper kan større mengder samlinger av nøkkelpunkter bli brukt til å gjengi kameraenes posisjoner i forhold til hverandre. Figur 2.1 illustrerer dette for bildeserier tatt med et kamera i bevegelse. [6]



Figur 2.1: Buntjustering

2.2.3 Tette punktskyer

En tett punktsky er grunnlaget for den ferdige 3D-modellen av et objekt. Punktskyene er nøkkelpunktene, funnet ved SIFT, plassert i forhold til hverandre i rommet. Plasseringene kan beregnes ut ifra at kameraenes posisjon og orientering er kjent fra buntjustering.

2.2.4 Rekonstruksjon av overflate

Overflaten i 3D-modellen konstrueres ut ifra en tett punktsky. Det finnes mange forskjellige algoritmer for denne prosessen. I hovedsak bygger algoritmene på å bruke punkter som ligger i nærheten av hverandre i skyen til å lage små lapper med overflate. Triangulering er brukt i en del av metodene. Rekonstruksjonen av overflaten blir representert av et nett av masker kalt mesh. Mesh er mange små trekantede flater som henger sammen, nesten som et lappeteppe.

2.2.5 Kartlegging av tekstur

Lappeteppet fra rekonstruksjonen av overflaten er hakkete og uferdig. For å ferdigstille modellen må kantene jevnes ut, dette gjøres ved å kutte de små lappene i mindre biter. Bitene er så små at kantene mellom dem nesten blir usynlige med mindre det er store endringer i overgangene. For at modellen skal være mest mulig lik det opprinnelige objektet blir det lagt på bilder av teksturen. Små deler fra hvert bilde fra bildeserien som har detaljer fra de forskjellige områdene blandes sammen for å gjenskape tekstur og farge fra objektet.

2.2.6 Kriterier til bildeserie

Før bildene som skal brukes til fotogrammetri skal tas er det mange hensyn å ta. For å få et optimalt sluttresultat er det viktig at bildene er detaljerte. SIFT trenger så mye informasjon som mulig for å finne nøkkelpunkter. Skal modellen gjenspeile virkeligheten må også bildene gjøre det. Fordi nøkkelpunkter blir skapt i områder der det er store endringer, ved hjelp av DoG, er bilder med høy kontrast og skarphet godt egnet til fotogrammetri.

Høy oppløsning, altså høy tetthet av pixler, er avgjørende for å få med de minste detaljer som kan være potensielle nøkkelpunkter. Høy oppløsning i bildet hjelper ikke hvis det av andre grunner er uklart. Uklarhet kan komme av bevegelse under fotografering eller av feil fokus. Uklarhet gir ikke bare dårlige detaljer og derfor få nøkkelpunkter, men det kan også gi misvisende informasjon om objektet og dermed være ødeleggende for resultatet.

Alle detaljer i et bilde er like interessante for en fotogrammetri algoritme, ikke bare på objektet, men også i bakgrunnen. Dette er detaljer et menneske gjerne filtrerer ut av fokus automatisk. Er ikke bakgrunnen gjennomtenkt kan denne derfor ta mye av oppmerksomheten

fra algoritmen og gi mange nøkkelpunkter som ikke er interessante for modellen. Mønster, tekstur og lyspunkter kan være kilder til forstyrrende detaljer i bakgrunnen. For å unngå store problemer med bakgrunnen er det derfor fordelaktig å ta bilder på nært hold hvor objektet dekker mest mulig av bildet.

Objektet som skal fotograferes burde ikke ha store, glatte flater som gir gjenskinn fra lyskilder da dette blir sett som et hvitt område uten detaljer til å lage nøkkelpunkter av. Objekter av glatt plast og glass vil være vanskelig å lage modeller av. Det finnes teknikker for å unngå problemet, men da må overflaten av objektet manipuleres før fotografering. Overflaten kan tilføres tekstur som ikke reflekterer lys, som for eksempel å lakkere objektet og deretter lage små prikker med en annen farge.

2.3 Annet teoretisk grunnlag

2.3.1 Power over ethernet (POE)

Power over ethernet er en samlebetegnelse for flere standarder, som tillater strøm å sendes sammen med data over en ethernet kabel.

2.3.2 Additiv tilvirkning

Additiv tilvirkning er en samling teknologier for produksjon av komponenter i ulike størrelser og materialer med utgangspunkt i en digital modell. Teknikkene er bedre kjent som 3D-printing. Additiv tilvirkning kan deles inn i syv hovedkategorier; Vat polymerisation, material extrusion, material jetting, binder jetting, powder bed fusion, direct energy deposition og sheet lamination. Selv om det finnes flere kategorier innen 3D-printing, er det oftest "material extrusion" som menes når man snakker om 3D-printing. Dette er en prosess der maskinen presser ut smeltet materialer fra en tupp, som blekk i en kulepenn, og bygger komponenten lagvis oppover. Det er denne metoden som menes når 3D-printing blir nevnt senere i rapporten.

3 MATERIALER OG METODE

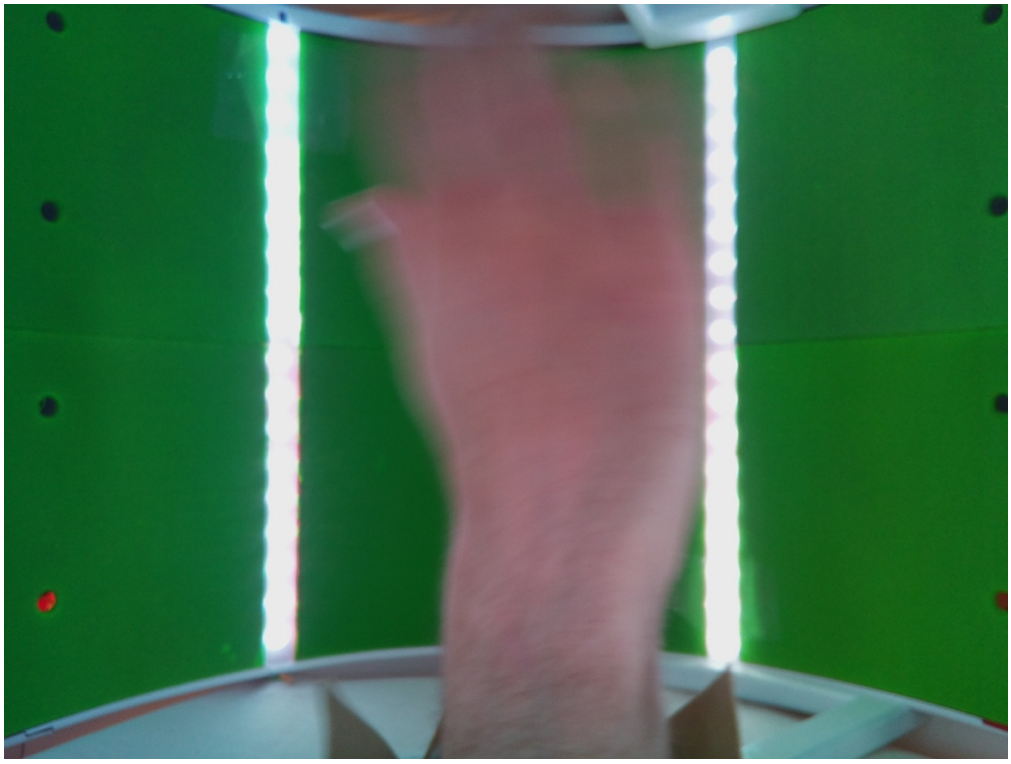
Når vi startet prosjektet opprettet vi en framdriftsplan i form av et ganttdiagram, vedlegg 4, som i grove trekk skulle dekke hele prosjektperioden. Ved å bruke ganttdiagram har vi hele tiden kontroll over hvilke oppgaver vi har gjort, hva som gjenstår og ved hjelp av fargekoding hvor mye de må jobbes med. Oppgavene ble delt inn i hovedmål med tilhørende delmål. Siden vi bare var to stykker ble det ofte en del muntlige diskusjoner rundt planen, og alle detaljer ble ikke nødvendigvis skrevet ned.

Annenhver uke har det vært statusmøter med veileder hvor vi har diskutert status for pågående aktiviteter, avvik i tidsplan og videre fremdrift. Møtene har også vært fordelaktig for tilbakemelding og justering av mål med prosjektet. Referat fra møtene ligger vedlagt i vedlegg 2.

Vi opprettet også et felles web-basert lagringsområde hvor begge lastet opp det de jobbet med. På denne måten fikk begge en bedre oversikt over prosjektet og vi kunne gi konstruktiv kritikk om det var nødvendig. Det ble også kommunisert regelmessig over sosiale medier om prosjektet.

3.1 Data

Den eneste formen for data brukt i denne oppgaven var bildene vi tok. Før vi kunne ta bildene var vi nødt til å stille fokus manuelt på hvert kamera, vi stilte også kontrast og skarphet, via programvaren, på hver enhet før bildene ble tatt. Bildene ble så sendt fra hver klient til server, hvor de ble lastet opp til programvaren for fotogrammetri. Den største usikkerheten her er i øyeblikket bildet tas. Her kan det oppstå ufrivillig bevegelse som gjør at kameraet mister fokus. Spesielt pasienter med håndleddsbrudd kan finne det vanskelig å holde hånden i ro under tiden som kreves for å ta et bilde. Posisjonen hånden holdes i vil også påvirke formen på 3D-modellen. Det vil derfor være viktig at pasienten holder hånden i en naturlig posisjon i det bildet tas.



Figur 3.1: Bilde tatt når håndledd ikke er i ro

3.2 Metode

For å opprettholde høyest mulig kvalitet på bildene er det viktig at de ikke behandles før opplasting. På grunn av at det eksisterer lite forskning rundt fotogrammetri, var det vanskelig å beregne kriterier for oppsettet. Vi brukte derfor en prøve- og feile tilnærming hvor vi testet alle alternativer vi kom på.

Ut fra det vi fant av kilder samlet vi en del grunnleggende teori om fotogrammetri og tidligere resultater. Ingen av kildene hadde testet 3D-skanning med samme kriterier og begrensninger som vi skulle i dette prosjektet. Til sammen ga det teoretiske grunnlaget nok informasjon til å sette opp en plan for utføring av prosjektet.

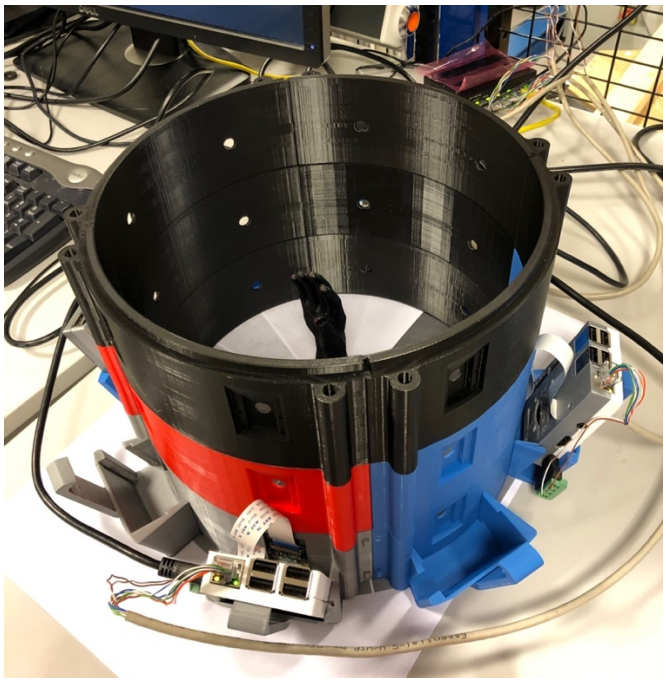
3.2.1 **Testing av programvare**

For å bli kjent med programvaren og eventuelle begrensinger ble det tatt flere bildeserier av små objekter. Bildeseriene ble tatt med frihånd med et mobilkamera. Fra disse bildene ble det forsøkt å lage 3D-modeller av objektene ved hjelp av to forskjellige fotogrammetri-

programmer. Modellene fra de to programmene ble sammenlignet for å bestemme hvilket som egnet seg best. Egenskaper som ble vurdert var prosesseringstid, brukervennlighet og ferdig modell. I disse testene ble det brukt bildeserier med flere bilder enn planlagt i sluttproduktet, dette for å være sikker på å få en modell som gikk an å sammenligne.

3.2.2 Testing av antall kamera

Flere tester ble underveis gjort for å finne ut hvor mange kamera som ville være nødvendig for å lage en tilfredsstillende modell.
I første forsøk ble det brukt det som var tilgjengelig av utstyr for å forsøke å samle inn testbilder. Et utkast av sylinderen til prototypen var produsert, og tillegg var fire Raspberry Pi med kamera tilgjengelig. Sylinderen bestod av tre ringer med plass til åtte kameraer i hver ring. De fire kameraene ble satt inn med lik avstand mellom hverandre. I hver høyde ble det tatt 12 bilder. Dette ble gjort ved å ta bilde med alle kameraene tre ganger, der sylinderen ble rotert 30 grader mellom hver gang. Et bilde av en sirkel oppdelt i like sekvenser ble lagt i bunnen av ringen som referanse for oppstilling og rotering. Totalt ble det tatt 36 bilder i denne testen. Objektene som ble skannet var en 3D-printet hånd i liten målestokk og en drikkekartong. Figur 3.2 viser oppsett ved første test.



Figur 3.2: Første oppsett for testing av antall kamera

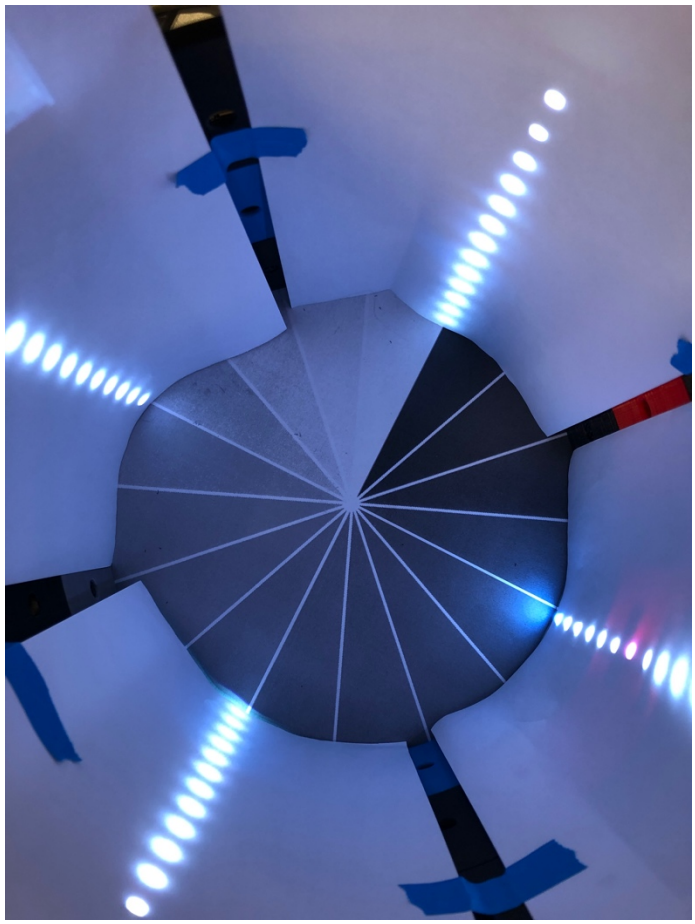
Flere tester ble gjennomført i løpet av prosjektet etter hvert som prototypens utforming endret seg. Tester på en reell hånd ble først gjennomført når skanneren var produsert og satt opp i endelig størrelse og utforming. For å illustrere skanning ved hjelp av flere kameraer enn maskinen hadde ble sylinderen manuelt rotert rundt hånden som stod så stille som mulig under prosessen med fotografering.

3.2.3 Testing av lys

Etter RGB LED-stripene var montert ble de testet med forskjellige innstillinger. Programmet som ble laget for å styre stripene hadde mulighet til å justere farge og lysstyrke. Det var plassert en lysstripe midt mellom hver kolonne med kameraer, på denne måten fikk alle kameraene like lysforhold. For å unngå påvirkning av lyskilder utenfor maskinen ble endene av sylinderen dekket med et tykt stoff for lysskjerming.

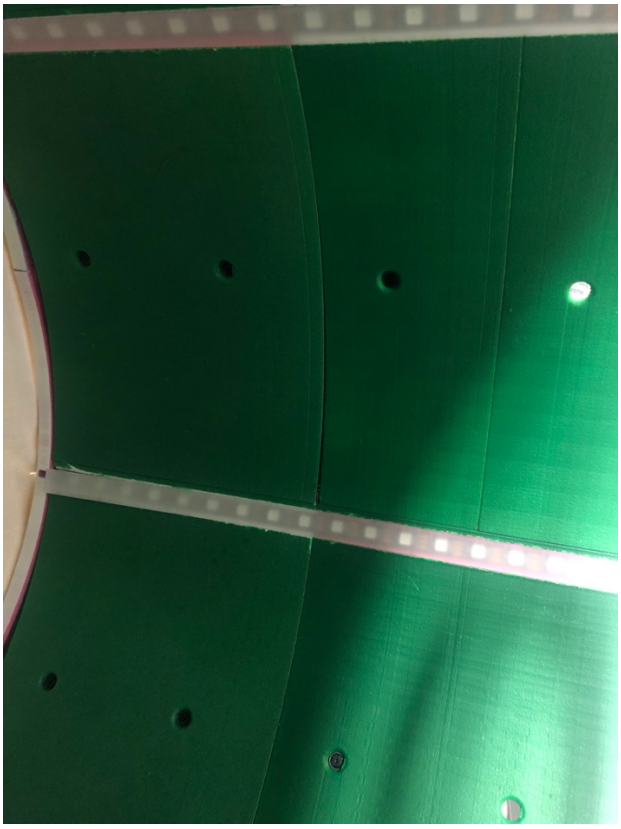
3.2.4 Testing av kameraoppsett

Gjennom prosjektperioden ble det laget flere prototyper. Den første prototypen inneholdt 4 ringer med 8 plasser til kamera i hver ring, noe som ga plass til totalt 32 kameraer. Til sammen ble det en sylinder med diameter på 250mm og høyde lik 300mm. Maskinen ble testet med 4 kameraer som ble flyttet rundt som beskrevet i kapittel 3.2.2. Første utgave hadde ikke lyssetting og ikke behandlet indre vegger.
For å unngå problematikk med mønster i materialet PLA i bakgrunnen ved fotografering, som nevnt i kapittel 2.2.6, ble veggene dekket med kopipapir. LED-stripene ble plassert mellom veggen og papiret, sånn at papiret også fungerte som en dimmer for lyset. Oppsettet er vist i figur 3.3.

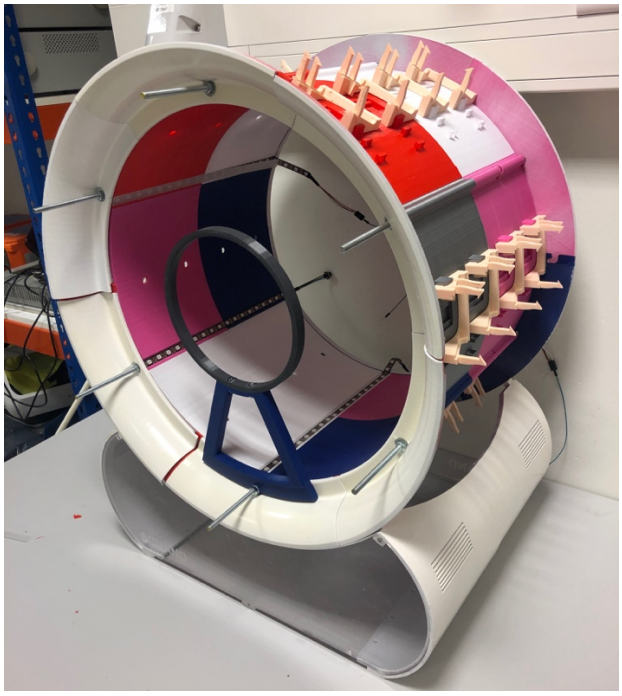


Figur 3.3: Oppsett ved testing

Siste prototype som ble laget hadde en sylinder satt sammen av 4 ringer med plass til 6 kameraer i hver. Prototype før integrering av kameraer er vist i figur 3.4. Til sammen ga det plass til 24 kameraer, men kun 3 av ringene ble fylt opp. Det ble det gjort tester med 18 kameraer. Innsiden av sylinderen hadde vegger dekket med grønn Plasti Dip. I veggene var det laget riss med plass til en LED-stripe mellom hvert kamera. Utenpå LED-stripene var det festet frostet plexiglass. Innsiden av sylinderen er vist i figur 3.5. Indre diameter på ny sylinder var 400mm og høyden var 320mm.



Figur 3.4: Innside av ferdig prototype

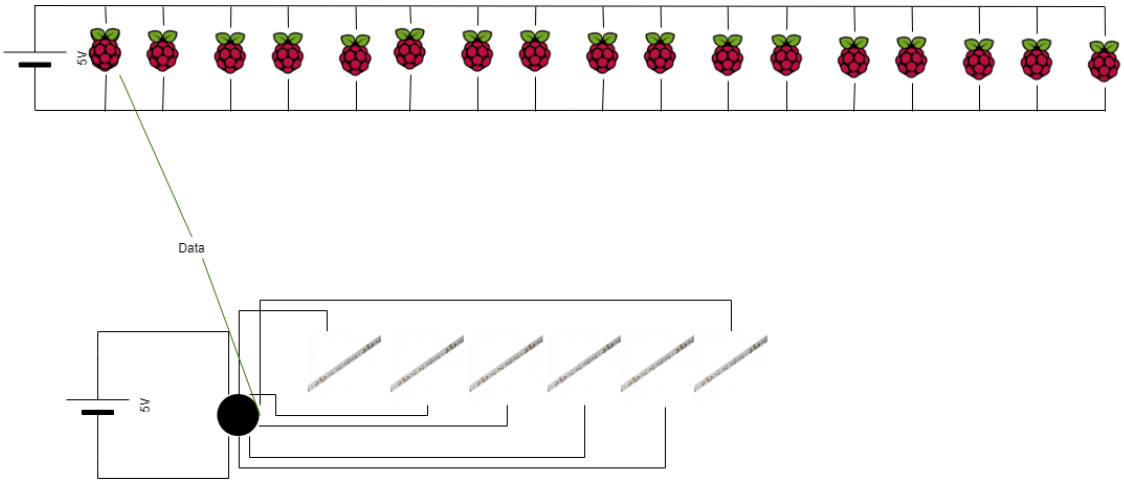


Figur 3.5: Prototype før montering av kamera

3.3 Materialer

3.3.1 Kretsskjema

Kretsen har en strømkilde på 5V/50A DC som gir strøm til 18 Raspberry Pi og 6 LED striper, 22 dioder i hver stripe. Hver Raspberry Pi trekker rundt 1A hver og LED stripene trekker 60mA per diode. Dette trekker totalt 25.92A. Raspberry Pi og LED stripene har hvert sitt uttak fra strømkilden og er koblet i parallell internt som vist i kretsskjemaet.

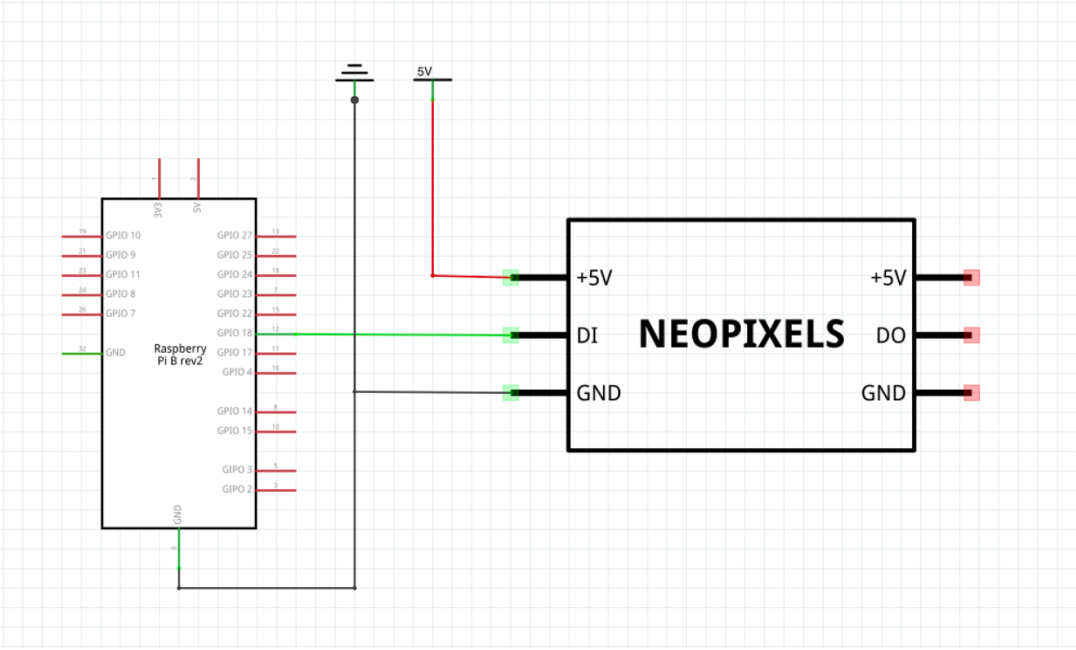


Figur 3.6: Kretsskjema

3.3.2 Lysstripe

Som belysning i prototypen ble det brukt LED-striper av typen WS2812b, også kjent som NeoPixel. WS2812b er striper med programmerbare RGB LED-pakker. Hver LED kan styres uavhengig av andre dioder i samme stripe. Programmet som styrer stripene ble skrevet ved hjelp av biblioteket Neopixel.

En stripe har tre pinner i enden og de er; strøm inn (5V), data inn (DI) og jord (GND). Alle LED-pixelene har tre pinner på hver side; 5V, DI, GND på ene siden, og 5V, DO(data out), GND på den andre. Datasignalet blir sendt gjennom alle diodene via DI/DO-pinnene. Datasignalet må være av typen PWM og sendes gjerne fra en mikrokontroller som for eksempel Arduino eller Raspberry Pi.



Figur 3.7: Kretsskjema for Raspberry Pi og NeoPixel

3.3.3 RGB LED

RGB LED er en lysdiode som inneholder tre dioder, en rød, en grønn og en blå diode. Ved å bruke RGB-fargespekteret kan et bredt utvalg farer dannes. RGB er et fargespekter som benytter en kombinasjon av rød, grønn og blå til å definere farger. For å kombinere fargene justeres intensiteten i de tre lysdiodene. Selv om det egentlig er tre separate dioder i en diode, står de så nærme at menneskeøyet bare oppfatter resultatet av kombinasjonen.

3.3.4 Arduino

Arduino er en plattform for prototyping av elektronikk basert på program- og maskinvare med åpen kildekode. En Arduino kan for eksempel registrere og styre fysiske omgivelser og komponenter ved å brukes sammen med ulike sensorer og aktuatorer. Det standardiserte oppsettet av tilkoblinger på Arduino gjør det mulig å bruke mange forskjellige tilleggsmoduler. Disse modulene er kalt shields. Programmering av Arduino-kretskort kan gjøres via USB, Bluetooth og andre måter.

3.3.5 Raspberry Pi 3 model b

Raspberry Pi er en datamaskin bygget på et enkelt kretskort. Dette kretskortet inneholder 4 USB, 1 ethernet, 1 HDMI, 1 micro USB, 1 kamera inngang og flere GPIO (general purpose input/output) pinner. Raspberry pi er også veldig brukervennlig og på størrelsen med et bankkort som gjør det veldig praktisk å bruke i mange situasjoner.

3.3.6 Raspberry Pi Camera v2

Raspberry Pi kamera modul er et kamera bygget spesifikt til bruk for Raspberry Pi.
Spesifikasjoner er:

Vekt:	3g
Oppløsning:	8 megapixel
Horisontal synsvinkel:	62.2°
Vertikal synsvinkel:	48.8°
Fokallengde:	3.04 mm.

3.3.7 Newlink 16/24 port unmanaged 10/100 ethernet switch

En nettverksswitch er en enhet som kobler sammen flere enheter på et nettverk ved hjelp av "packet switching" for å motta, prosessere og sende data.

3.3.8 Strømforsyning 5V 50A

Strømforsyning er en komponent som leverer energi til en bruker. Hovedfunksjonen til en strømforsyning er å konvertere elektrisk strøm fra en kilde til rett volt, ampere og frekvens til en bruker.

3.3.9 RJ45

RJ45 er en type modulær forbinder for kabler brukt til elektroniske enheter.

3.3.10 Wago rekkeklemmer

Rekkeklemmer brukes som delingspunkt mellom en strømkilde og flere brukere.

3.3.11 Multimeter

Et multimeter er et elektronisk måleinstrument som kombinerer flere målefunksjoner i en enhet. Typisk kan et multimeter måle volt, ampere og motstand. Det finnes både analoge og digitale multimeter, men digitale er mye vanligere og som regel å foretrekke over analoge.

3.3.12 microSD Card

Secure digital card er en type digitalt minnekort som brukes i de fleste apparater som krever eksternt minne.

3.3.13 Plexiglass

Plexiglass er gjennomsiktig termoplast. I vårt tilfelle brukt til å dimme lyset ved å froste det.

3.3.14 PLA

PLA står for polylactic acid, og er en utbredt bioplast med smeltepunkt på 160-200 °C. Den er svært mye brukt innen 3D-printing og er materialet vi brukte til prototypen.

3.3.15 Plasti Dip

Materiale brukt til å beskytte komponenter brukt i behandling av bil. Syntetisk gummi som påføres som spray.

3.3.16 **Kostnader**

Beskrivelse:	Pris stk (NOK):	Antall:	Pris tot (NOK):	Link
raspberry pi 3 model B	279	18	4914	https://no.rs-online.com/web/p/processor-microcontroller-development-kits/8968660/
raspberry pi camera v2	237	18	4266	https://no.rs-online.com/web/p/video-modules/9132664/
kingston sd minnekort 16 GB	122	18	2196	https://no.rs-online.com/web/p/sd-cards/9011289/
micro usb kontakt	58	18	1044	https://no.rs-online.com/web/p/micro-usb-connectors/1792871/
Newlink switch	860	1	860	https://uk.rs-online.com/web/p/network-switches/7947079/
RJ 45	2	36	72	https://no.rs-online.com/web/p/rj45-connectors/3316386/
CAT kabel	9,197 pr m	27m	248,3	https://no.rs-online.com/web/p/cat6-cable/0556392/
LED Strips	225	6	1350	https://no.rs-online.com/web/p/led-strip-lights/7736936/
Power Supply	960	1	960	https://no.rs-online.com/web/p/embedded-switch-mode-power-supplies-smps/1711502/
Pris Totalt			**15662**	

Tabell 3.1: Kostnader for materialet brukt i prosjektet

4 RESULTATER

4.1 Programvare

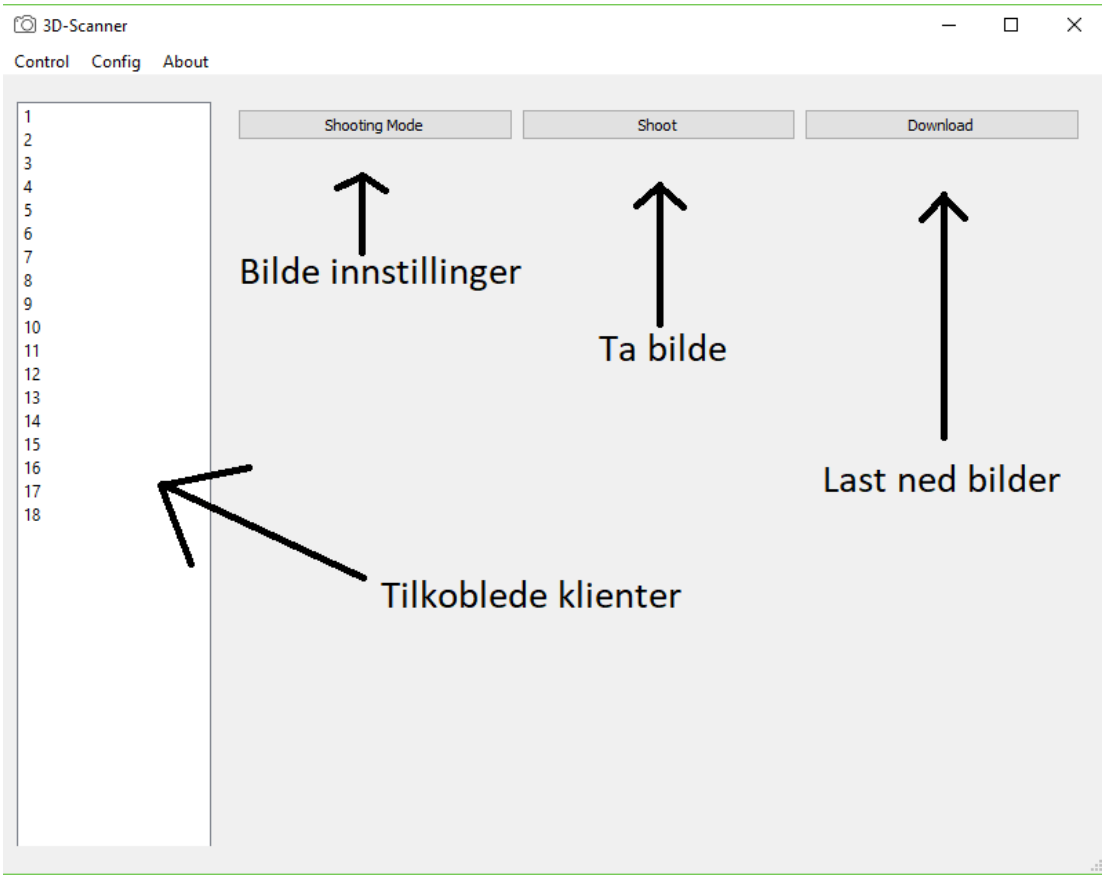
Programvaren vi utviklet var en typisk server/klient konstellasjon hvor hver Raspberry Pi representerte en klient, og en hvilken som helst datamaskin kan brukes som server. Kommunikasjonen foregår over et lokalt nettverk ved hjelp av python sitt socket bibliotek.

Den vanligste formen for kommunikasjon er TCP/IP som garanterer at alt du sender kommer frem, er rask og pålitelig. En annen form for kommunikasjon er over UDP som i motsetning til TCP/IP ikke krever ende- til ende kommunikasjon, og kan dermed sende informasjon til klienter samtidig. Selv om TCP/IP var et naturlig valg ble det også brukt UDP kommunikasjon. Vi valgte å bruke UDP i tillegg fordi vi trengte et stillbilde, og selv om TCP/IP ville vært marginalt dårligere er UDP et bedre valg for eventuell skalering av antall klienter. Bildene ble sendt over TCP/IP for å være sikker på å få høyest mulig oppløsning på bildene, og at all data ble sendt over til serveren.

Programvaren har også funksjon for oppdatering, omstart, avslutte, endring av kamerainnstillinger. Alle funksjonene er skalerbare. Det er også mulig å velge ut spesifikke klienter for feilsøking med de samme funksjonene.

4.2 GUI

For å gjøre programvaren mer brukervennlig var det nødvendig å lage en GUI. Brukergrensesnittet er særs enkelt og har kun funksjoner for å ta bilde, nedlasting og endring av parameterne skarphet og kontrast. Dette er kun de aller mest nødvendige funksjonene, og de eneste funksjonene en bruker trenger. Det er naturligvis en del funksjoner som kunne vært lagt til og GUI'en kunne vært jobbet mer med, men når oppgaven gikk mot slutten var det mer fornuftig å fokusere på 3D-modellen og hva som kunne gjøres for å få den best mulig. Figur 4.1 viser brukergrensesnittet.



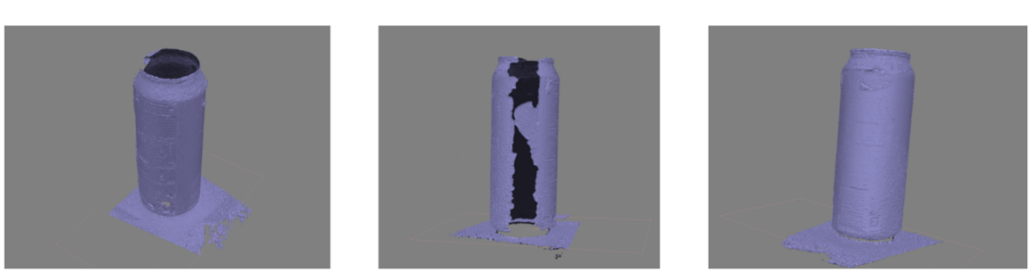
Figur 4.1: Brukergrensesnitt

4.3 Testing av programvare

Agisoft PhotoScan Pro

Tie point	22 min 3 sek
Dense cloud	3 t 23 min 55 sek
3D Model	2 t 19 min
Texture	11 min 37 sek
Totalt	**5 t 16 min 35 sek**

Tabell 4.1: Tidsbruk ved prosessering av modell av brusboks



Figur 4.2: Ferdig modell av brusboks i PhotoScan Pro

Autodesk ReCap Pro

ReCap Pro gir ingen rapport etter modellen er ferdig, og det finnes derfor ingen tall på tidsbruken i prosessen. Hele prosessen tok om lag 3 timer fra bildene ble lastet opp til den ferdige modellen var lastet ned. Figur 4.3 viser ferdig modell.

Figur 4.3: Ferdig modell av brusboks i ReCap Photo

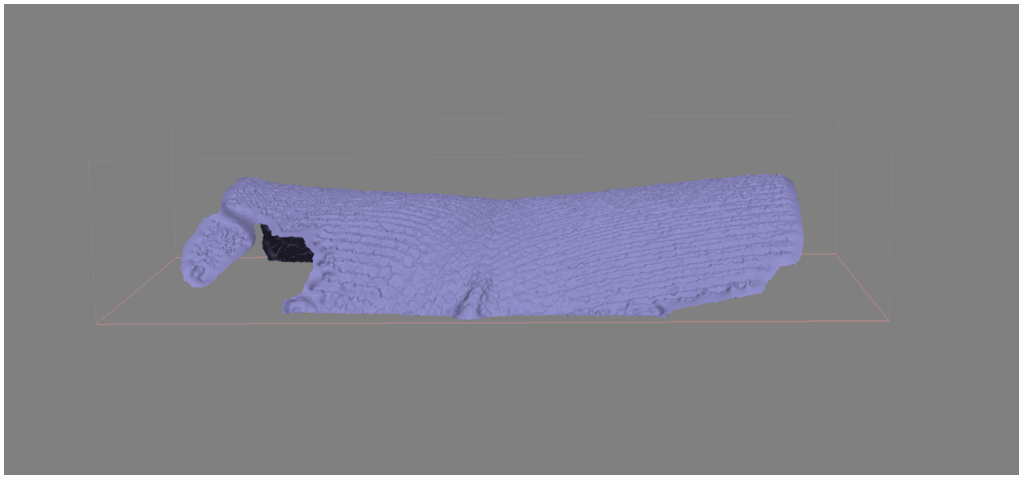
4.4 Testing av antall kameraer

Bildeseriene for testing av antall kameraer ble tatt av en hånd med en strømpe tredd over området mellom albue og fingre. Bildene ble tatt i tre serier hvor sylinderen i maskinen ble rotert om lag 20 grader mellom hvert sett. Til sammen ble det tatt 54 bilder. Alle modellene er prosessert med like innstillinger, høyest mulig

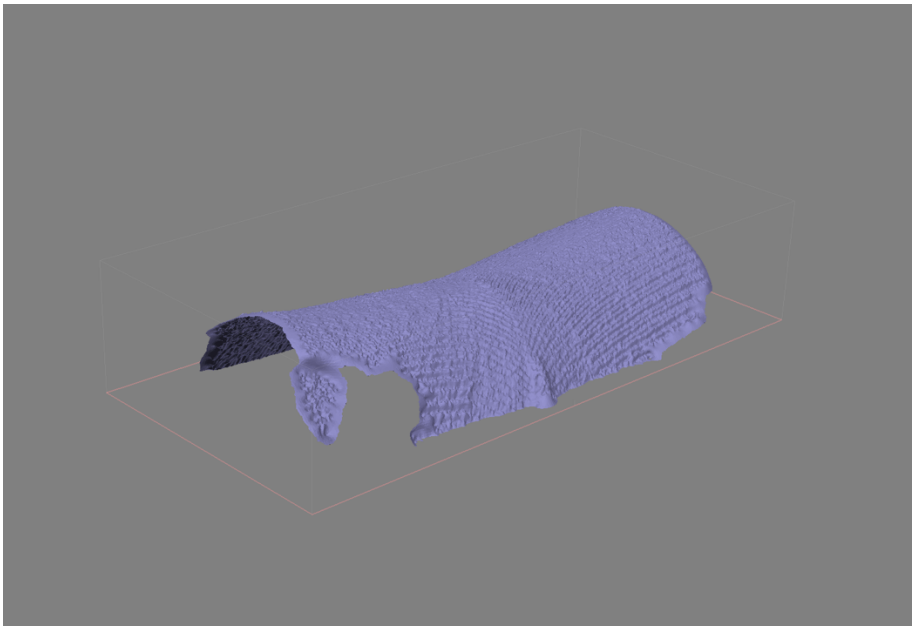
Første modell ble laget fra et sett på 18 bilder.
Antall kameraer plassert: 2 av 18
Antall nøkkelpunkter: 523
Tidsbruk:

Tie point	2 min 17 sek
Dense cloud	6 sek
3D Model	2 min 6 sek
Texture	35 sek
Totalt	**5 min 4 sek**

Tabell 4.2: Tidsbruk ved prosessering av modell første modell



Figur 4.4: Ferdig første modell -1



Figur 4.5: Ferdig første modell -2

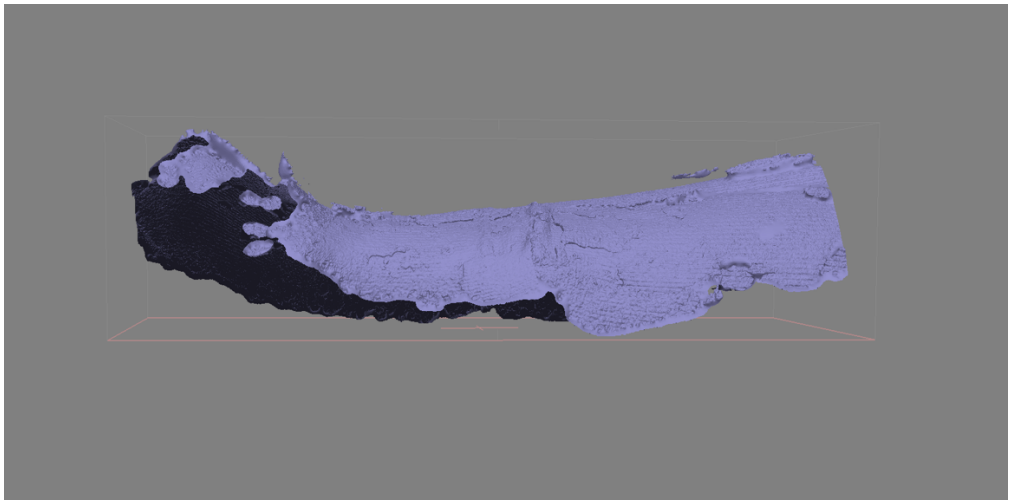
Andre modell ble laget fra to sett på 18 bilder, til sammen 36 bilder.

Antall kameraer plassert: 18 av 36

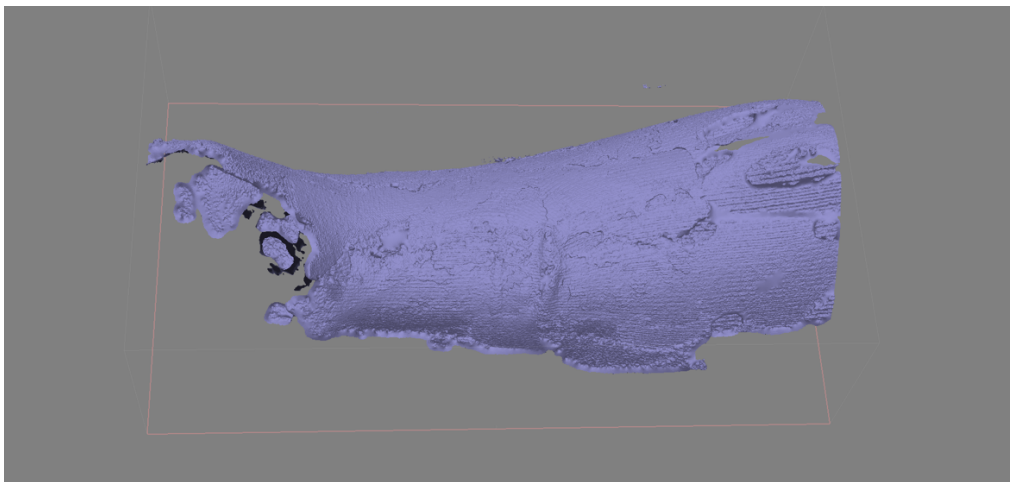
Antall nøkkelpunkter: 6902

Tie point	7 min 43 sek
Dense cloud	5 min 48 sek
3D Model	9 min 22 sek
Texture	2 min 7 sek
Totalt	**25 min**

Tabell 4.3 Tidsbruk ved prosessering av andre modell



Figur 4.6: Ferdig andre modell -1



Figur 4.7: Ferdig andre modell -2

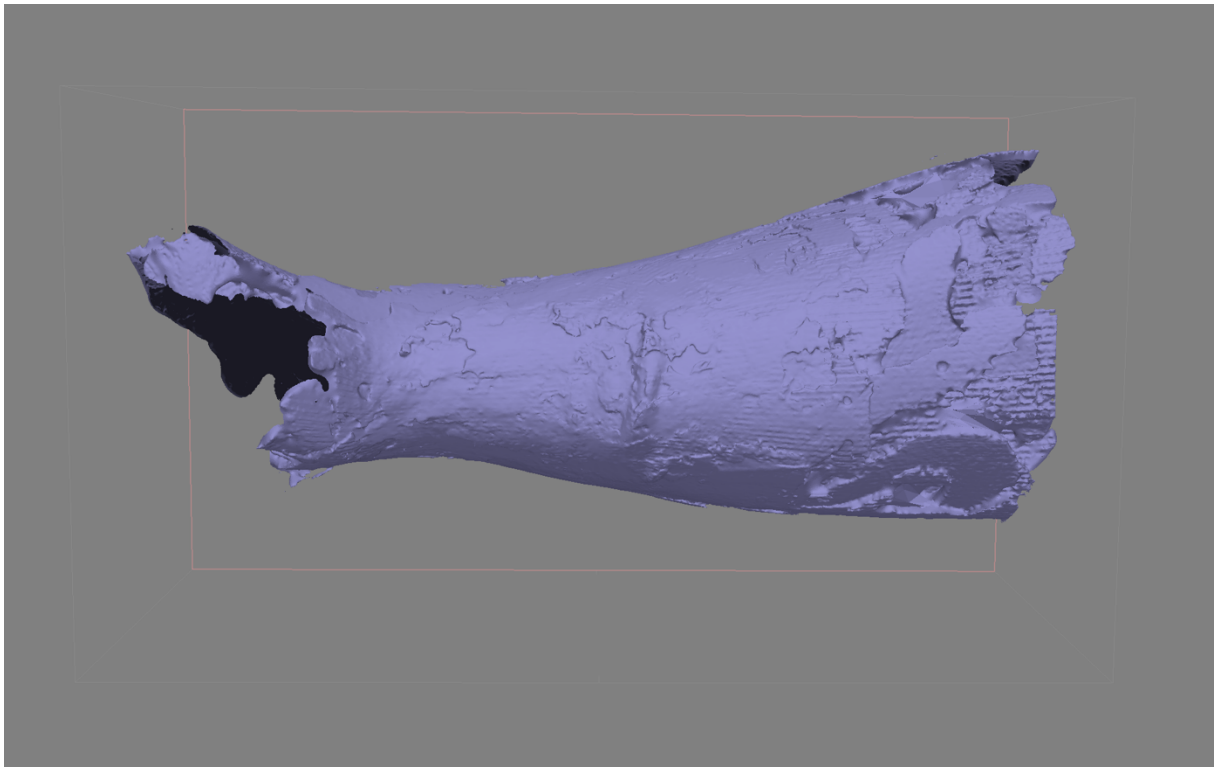
Tredje modell ble laget fra tre sett på 18 bilder, til sammen 54 bilder.

Antall kameraer plassert: 52 av 54

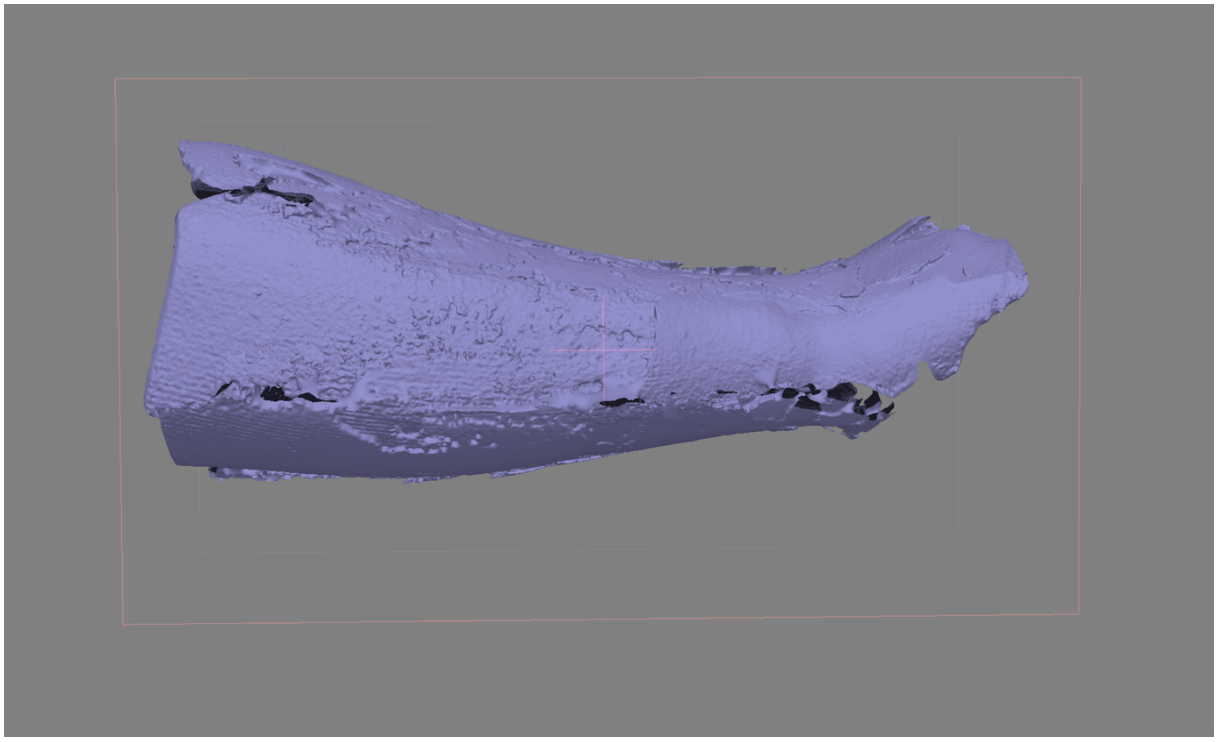
Antall nøkkelpunkter: 21396

Tie point	29 min 30 sek
Dense cloud	43 min 40 sek
3D Model	48 min 35 sek
Texture	10 min 18 sek
Totalt	**2 t 1 min 45 sek**

Tabell 4.4: Tidsbruk ved prosessering av tredje modell



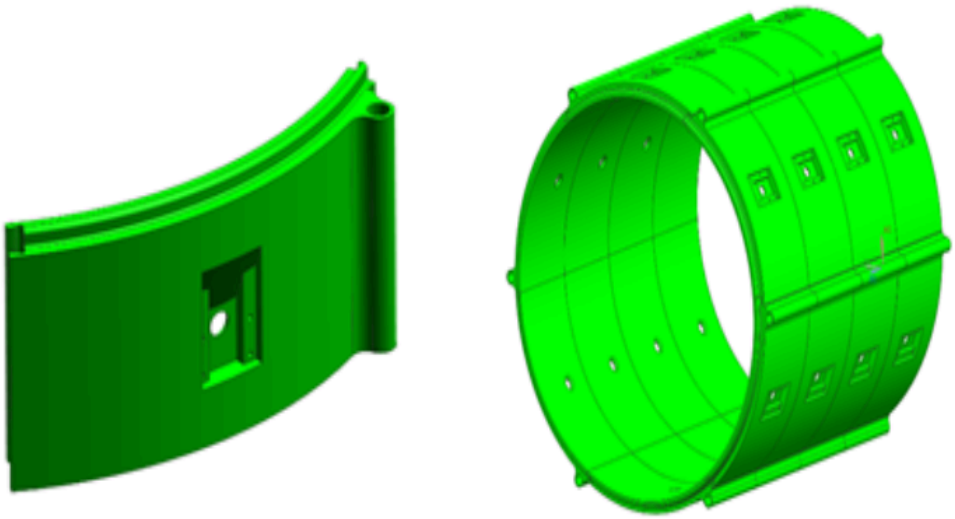
Figur 4.8: Ferdig tredje modell -1



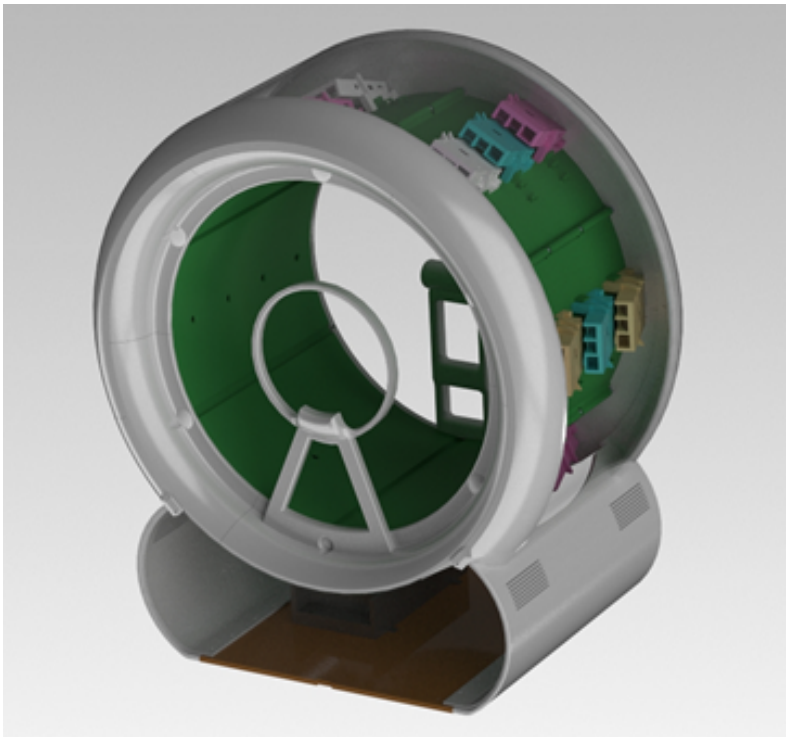
Figur 4.9: Ferdig tredje modell -2

4.5 Prototype

I figur 4.9 ser du stegvis hvordan prototypen ble til. Vi valgte å 3D-printe prototypen fordi dette ga muligheter til å spesialtilpasse designen. Dette var viktig fordi vi trengte spesifikke dimensjoner med tanke på overlapp mellom bildene, men også alle Raspberry Pi enhetene måtte festes til prototypen. Vi fikk også litt ekstra spillerom med tanke på endringer med denne måten. Selve designen og printingen var det POD gruppen som tok seg av, og vi kom med innspill om hvordan spesifikasjonene måtte være. Det viste seg at PLA materialet ga for mye gjenskinn, men det løste vi med å spraye innsiden med Plasti Dip. Prototypen tjente sitt formål bra, men skulle det vært aktuelt å bruke den i en profesjonell sammenheng måtte vi justert passformen på enkelte deler litt. Raspberry Pi har en tendens til å skli litt ut av sin beholder, og ikke alle plexiglassene sitter like godt. Dette skyldes nok også at ikke alle delene er printet av samme 3D-printer som gir litt slingring i målene.



Figur 4.10: Oppbygning av prototype [4]



Figur 4.11: Prototype skannemaskin [4]

5 DRØFTING

5.1 Oppgaven

Målsettingen for oppgaven har endret seg gradvis etterhvert som den ble jobbet med. I utgangspunktet var målsettingen å kunne produsere en ferdig gips i løpet av 10-15 minutter. Det viste seg fort at dette var urealistisk i forhold til tiden vi hadde til rådighet, men også i forhold til programvaren for fotogrammetri som eksisterte, antall kamera tilgjengelig, vinkel på kamera og kvaliteten på kameraene. Tiden det tok fra bildene ble lastet opp og vi fikk en ferdig 3D-modell kom fort opp i 2 timer. I tillegg til dette kan tiden det tar for å skrive ut en 3D-skinne være alt fra 8-12 timer lang.
Kvaliteten på kameraene viste seg også å være for lav. Dette var en risiko vi var klar over når vi begynte, men foruten Pi3dscan**Feil! Fant ikke referansekilden.** har det ikke vært gjort noe lignende prosjekt tidligere. Pi3dscan va r også et vellykket prosjekt, noe som ga motivasjon for en lignende tilnærming til vårt prosjekt, men det viste seg at kravene for suksess ikke var de samme.
Antall kamera var også en viktig faktor. På en bacheloroppgave er det begrenset hvor mye ressurser en kan bruke, noe som gjorde at vi endte opp med atten Raspberry Pi med kamera. Dette er i de fleste situasjoner mer enn tilstrekkelig, men for oppgaver som involverer fotogrammetri vil dette nesten alltid være for lite. Igjen var dette en mangel vi var klar over, men denne kunne lett omgås ved å bevege kameraene mellom hver gang vi tok bilder. At det var hensiktsmessig å ha vinkel på enkelte av kameraene var ikke noe vi var klar over før vi oppdaget det under testingen.
Med dette som utgangspunkt ble det umulig å nå den opprinnelige målsettingen. Vi satt derfor en ny målsetting om å lage den beste 3D-modellen vi kunne ut ifra de midlene vi hadde, og parameterne vi kunne endre på.

Etter vi fikk satt opp prototypen og begynte å teste gikk oppgaven over i å optimalisere de parameterne vi hadde kontroll over. Dette var:

- Lys
- Materiale
- Kamerainnstillinger
- Objektet

For å oppnå optimal lyssetting brukte vi frostet glass og seks led striper på medium styrke for å spre lyset best mulig. Dette bidro til et jevnt og klart lys i hele sylinderen, og ingen skygger på objektet vi tok bilde av.
En konsekvens av at vi valgte å 3D-printe prototypen var at vi måtte bruke materialet PLA. Problemet med dette var at det reflekterte for mye lys. Løsningen på dette ble å dekke delene med Plasti Dip.
Selv om kvaliteten på kameraet til Raspberry Pi var begrenset var det fortsatt noen parameter vi kunne med fordel endre på. Det ene var kontrast og det andre var skarphet, som forklart i kapittel 2.2.6.
Det som viste seg å ha størst betydning var hvordan vi påvirket selve håndleddet som skulle tas bilde av. I utgangspunktet bestemte vi oss for at det ikke var nødvendig å gjøre noe med håndleddet før det skulle tas bilde av. Det vi fant av kilder og tidligere tester tilsa at dette skulle være greit. Det viste seg likevel at selv om vi tok med veldig mange bilder, klarte ikke fotogrammetri-programvaren å gjenskape en 3D-modell. Løsningen på dette viste seg å tre en strømpe over armen. Denne ideen var en bisetning i en artikkel vi leste når vi hadde gått oss fast som viste seg å være svært verdifull for vårt prosjekt.

5.2 Programvare

Programvaren i dette prosjektet ble skrevet i Python. Ingen av oss hadde noen tidligere erfaring med Python, men språket er vidt brukt med mye ressurser på nett og virket som et naturlig valg for oppgaven.
Programmet inneholder all funksjonalitet som oppgaven krever, men samtidig er det rom for utvidelse. Spesielt fra et test-perspektiv kan det være hensiktsmessig å innføre funksjon for forhåndsvisning av kamera. Overføringshastigheten av bilder fra klient til server kan også forbedres. Foruten dette møter programvaren alle kravene til oppgaven, og fungerer som den skal.

5.3 GUI

Det grafiske brukergrensesnittet inneholder i likhet med programvaren funksjonaliteten som oppgaven krever, men også her er det rom for forbedringer. Generelt vil mange av de funksjonene man innfører i programvaren også måtte innføres i GUI.

5.4 Prototype

Prototypen egner seg greit til testing, men har en del detaljer som kan forbedres:

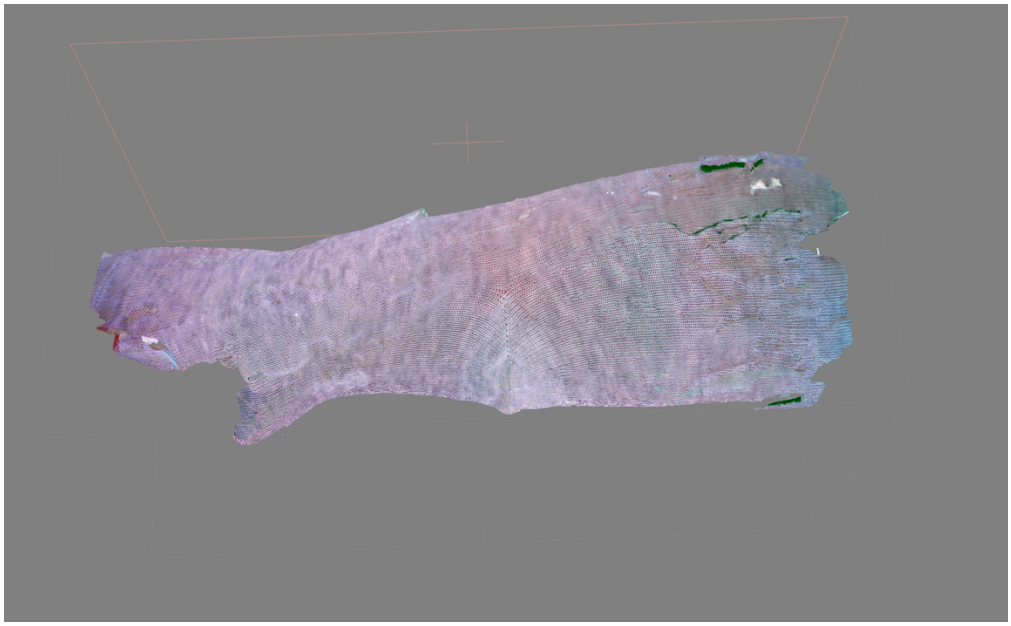
1. Passformen til plexiglassene er ikke perfekt, og faller til tider ut.
2. Antall kamera vil øke antallet ethernet kabler som kan bli uoversiktlig og rotete.
3. Enkelte RJ45 har dårligere kvalitet og kan ved bevegelse plugges ut av switch eller Raspberry Pi enhet.
4. Passformen til de 3D-printede delene har variasjon på grunn av avvik i forskjellige 3D-printere.
5. Tøystykkene brukt for å skjerme mot lys er ikke optimale og slipper inn litt lys.
6. Boksen under sylinderen er ikke stor nok til å romme alle komponentene.

Et alternativ som ble vurdert og diskutert var en mekanisk innretning som kunne rotert for hvert bilde. Dette hadde åpnet muligheter for færre kamera med bedre spesifikasjoner. Hovedargumentet mot denne løsningen var at en person, brukket håndledd eller ikke, ville hatt problemer med å holde hånden stødig nok over tiden det tok å ta bildene. En slik løsning er også uaktuelt ved bruk av Raspberry Pi siden den bruker rundt to sekunder på å stille fokus mellom hvert bilde.

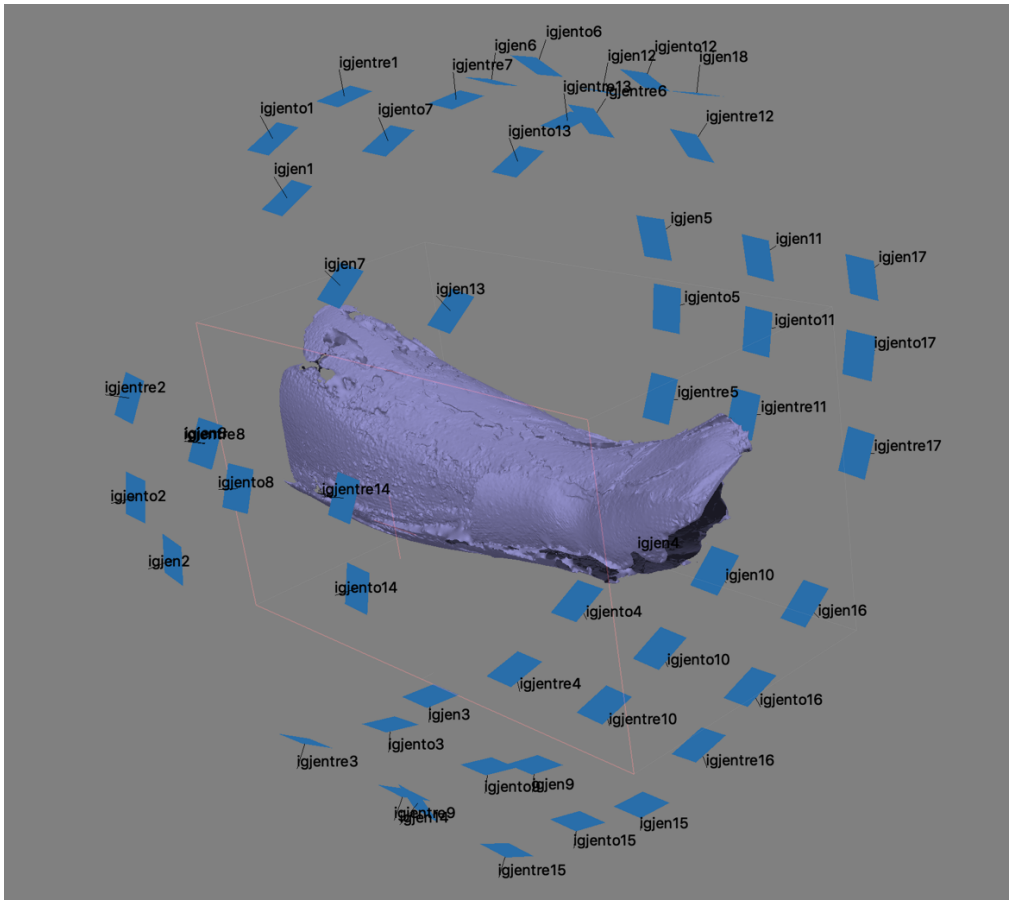
Nå etter vi har utforsket bruken av Raspberry Pi sin kameramodul er vi igjen på et stadium hvor vi mener at en mekanisk innretning kan være en løsning. Denne generasjons Raspberry Pi kameramodul har ikke kraftige nok spesifikasjoner, og vi vet heller ikke om noen andre kamera som tilfredsstiller kravene. Dette problemet kan unngås med en mekanisk innretning, men som nevnt vil dette by på andre utfordringer.

5.5 3D-modell

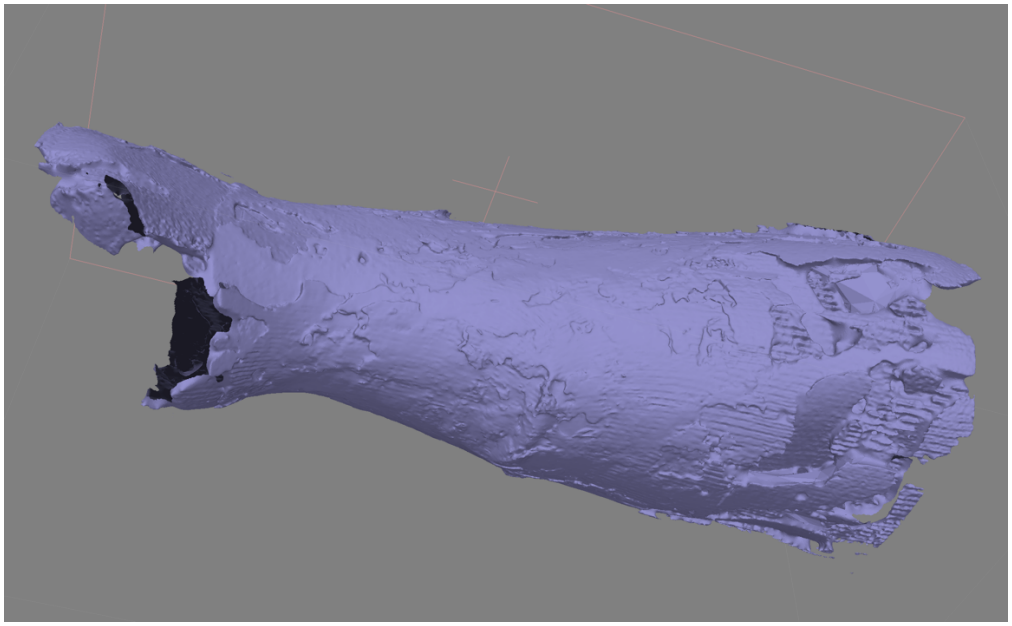
Den endelige 3D-modellen vi fikk ut etter å ha optimalisert alle parameterne vi hadde kontroll over er vist i figur 5.1. Man kan tydelig se at dette er en modell av et håndledd, og kvadratene rundt, vist i figur 5.2, er hvor programmet oppfatter at bildene er tatt fra. Geometrien stemmer ganske bra, vi ser at håndleddet er tykkere nede og blir smalere lenger opp. Går vi inn og ser på modellen fra et 3D-perspektiv ser vi at utformingen på hånden er helt lik helt opp til hvor strømpen stopper. Problemet oppstår når vi går nærmere inn og ser. I figur 5.3 ser man tydelig at geometrien er ujevn, og trenger videre behandling.



Figur 5.1: Ferdig modell med tekstur



Figur 5.2: Modell med kameraplassering



Figur 5.3: Ujevn geometri i ferdig modell

Figur 5.4: Hånden som ble tatt bilde av

5.6 3D-printet støtteskinne

Sammenligner vi en 3D-printet støtteskinne i PLA materiale med en tradisjonell gips kan vi se fordelene. Når kvaliteten på 3D-modellen når et akseptabelt nivå vil det ikke lenger være avhengig personell lenger hva det angår passform og komfort, men av bildene og 3D-printer. Det heldige med dette er at hvis bildene av håndleddet har like høy kvalitet for hver pasient, vil støtteskinne også ha den samme kvaliteten hver gang. Fra bacheloroppgaven skrevet tidligere [2] vet vi at kvaliteten på en 3D-printet støtteskinne er god, og at plager som kløe, lukt og tyngde ikke er en faktor. I tillegg er det mulig å ta den av, som gjør hygiene lettere. Med tanke på ressurser vil materialkostnader være som vist under Det store utslaget vil vise seg i personalkostnader. En tenkt løsning vil ta 10-15 minutter som sammenlignet med 20 minutter (rundt det en erfaren sykepleier bruker) er en stor forbedring. I tillegg vil prosessen være autonom, noe som frigjør personalet til andre oppgaver, være en betydelig positiv endring.

Utføres av:	Tid (timer)	Lønn (NOK)	Materialkost (NOK)	vedlikehold pr. gips (NOK)	Sum kost (NOK)	Differanse (NOK)	Årlig differanse (NOK)
Sykepleier	0,5	250	50	0	175	121,4	3642000
Turnuslege	0,75	350	150	0	412,5	358,9	10767000
Maskin	0,1	350	15	3,6	53,6	0	0

Tabell 5.1: Kostnader for gips

5.7 Forslag til endringer

Problematikken er nå drøftet, og forslag til endringer er også nevnt enkelte steder. Her vil vi liste opp de konkrete endringene vi mener er nødvendig.

- Antall kamera.

Selv om vi fikk tatt nok bilder ved å rotere prototypen er dette ikke en løsning i en profesjonell sammenheng.

- Kamera spesifikasjoner.

I likhet med antall kamera er dette en veldig tydelig mangel for øyeblikket. Med kraftigere spesifikasjoner vil bildene få flere punkt for programvaren å koble sammen slik at den kan lage en bedre modell.

- Kameravinkel.

Dette oppdaget vi når vi undersøkte området hvor håndleddet går over til fingrene. I dette området er det mer geometri, og det hadde vært lettere å fange opp med flere innfallsvinkler på kameraene.

- Spesialtilpasset strømpe.

For vårt prosjekt var dette den største åpenbaringen. Vi gikk fra en todimensjonal til en tredimensjonal modell når vi tok i bruk en strømpe. Det viste seg at geometrien er mye lettere å fange opp.

Vi brukte en sokk som ble kjøpt på den lokale Eurospar butikken. Dette var mot slutten av prosjektet, og vi rakk derfor ikke å tilpasse den vårt bruk i stor grad. Problemet med å bruke en sokk er at den ikke er tilpasset en hånd. Grunnen til at vi valgte en sokk istedenfor en hanske var at utvalget med tanke på mønster og stoff er mye større.

En ideell løsning hadde vært en tettsittende hanske, men ikke så stram at det skaper ubehag hos pasienten. Både mønster og stoff på sokken vi brukte virket bra, men her er det fortsatt rom for testing.

6 KONKLUSJON

Prosjektet ble gjennomført, etter revidering, som planlagt. I utgangspunktet var målet å utvikle en prototype som kunne ta bilder av et håndleddsbrudd for så å produsere en støtteskinne innen 15-20 minutter. Det ble fort klart at dette var, per i dag, uoppnåelig.

Det reviderte målet var å fremstille en 3D-modell med utgangspunkt i de midlene vi hadde tilgjengelig. Selv om vi nådde målet med å fremstille en 3D-modell var geometrien mangelfull, og vi kunne se at overflatestrukturen på figuren ikke var jevn nok. Dette gjorde at vi ikke kunne videreutvikle modellen til en fullverdig støtteskinne uten å behandle den nøye i CAD, som i seg selv er omfattende arbeid.

Etter endt prosjekt var det litt vemodig å sitte igjen uten noe særlig konkrete resultater å vise til, men vi har gjort omfattende testing med de midlene vi hadde og sitter igjen med mye kunnskap og tanker om videre utvikling vi ikke hadde når vi startet. Denne kunnskapen har vi prøvd å gjengi etter beste evne i denne rapporten.

Til slutt vil vi poengtere at vi tror dette prosjektet er mulig å realisere, spesielt etter vi har jobbet med det over en lenger periode. Problemet ligger i at alternativene to studenter har er svært begrenset. Hadde vi hatt tilgang til store mengder kamera som var tilpasset oppgaven, og i tillegg hatt en spesialtilpasset strømpe for hånden tror vi at 3D-modellen hadde blitt god nok til å 3D-printe. Tiden det tar fra bildene lastes opp og til en støtteskinne blir 3D-printet er rundt 8-12 timer, noe som heller ikke er en bærekraftig løsning i en profesjonell sammenheng. Denne prosessen er heller ikke noe vi har kontroll over, og på den måten kan endre.

7 REFERANSER

[1] Kaare Solheim (2018) *Benbrudd.* Tilgjengelig fra: https://sml.snl.no/benbrudd (hentet: 28. januar 2019)

[2] Gya, M. og Thorsen, A. D. (2017) *Spesialtilpasset gips for håndleddsbrudd ved bruk av dagens 3D-teknologi*. Bacheloroppgave. NTNU Ålesund.

[3] Pi3Dscan (2018) Tilgjengelig fra: http://www.pi3dscan.com/index.php (hentet: 25. januar 2019)

[4] Alvestad, V. A. J, Nedrelid, O. H og Sjåstad. D (2019) *Maskin for brukertilpasset gips.* Bacheloroppgave. NTNU Ålesund

[5] Dmitry Semyonov (2011) *Algorithms used in Photoscan.* Tilgjengelig fra: https://www.agisoft.com/forum/index.php?topic=89.0 (hentet: 10. april 2019)

[6] Dellaert, F. *CVPR 2014 Visual SLAM Tutorial* Tilgjengelig fra: http://www.cs.cmu.edu/~kaess/vslam_cvpr14/media/VSLAM-Tutorial-CVPR14-A13-BundleAdjustment.pdf (hentet 10. april 2019)

[7] Sinha, U. (2017) *SIFT: Theory and Practice.* Tilgjengelig fra: http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/ (hentet 10. april 2019)

VEDLEGG

Vedlegg 1	Forprosjektrapport
Vedlegg 2	Møtereferat
Vedlegg 3	Kode

VEDLEGG 1

Forprosjektrapport

1 INNLEDNING

Kort innledning om bakgrunn – om valg av oppgave, oppdragsgiver, den grunnleggende problemstillingen og formålet med oppgaven.

Håndleddsbrudd er den vanligste bruddskaden som finnes. I Norge er normalen over 15 000 forekomster per år. Etter bruddet, om nødvendig, blir beinet trykket tilbake og man gipser for å støtte håndleddet. Gipsing er en omfattende prosedyre som krever personell, ressurser og tar lang tid.

Valget falt på denne oppgaven fordi den er veldig reell, og en eventuell løsning på denne problemstillingen vil føre til en fundamental, positiv, endring i måten håndleddsbrudd blir behandlet på.

Oppdragsgiver er NTNU Ålesund, med Paul Steffen Kleppe og Webjørn Rekdalsbakken som veiledere.

Oppgaven går ut på å forbedre metoden for behandling av enkle håndleddsbrudd. Ved hjelp av teknologi som 3D-skanning og 3D-printing skal en lege eller sykepleier enkelt, og på kort tid, kunne behandle håndleddsbrudd. Dette skal løses ved å utvikle en løsning som skanner hånden med bruddet, lager en 3D-modell av hånden og printer en støtteskinne.

Vår del av oppgaven blir å lage en rigg med kameraer som pålitelig, og på kort tid kan lage en 3D modell av en hånd med håndleddsbrudd.

Formålet med oppgaven er å redusere behandlingstiden ved håndleddsbrudd slik at leger og sykepleiere kan bruke tiden på andre viktigere ting. En løsning vil over tid gi betraktelige reduserte kostnader både i form av lønn og materiale brukt til behandling.

2 BEGREPER

- Definisjon av sentrale begreper i prosjektet.
- 3D-skanning. Skanning av fysisk objekt med hensikt å lage en digital 3D-modell.
- 3D-printing. Fysisk utskrift av digital modell.
- CAD. Computer Aided Design. Programvare for redigering av 3D modeller.
- Fotogrammetri. Måleteknikk som brukes når en ved hjelp av bilder observerer og bestemmer egenskaper som beliggenhet, form, størrelse og identitet for avbildet

terreng eller andre gjenstander. Ut fra disse målingene kan en framstille 3D-modeller som kan brukes til diverse formål.

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
250956
748001

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget IR303612

3.1.1 Oppgaver for prosjektgruppen – organisering

Oppgaver:

1. Utforske og vurdere eksisterende teknologier innenfor 3D-modellering.
2. Finne ut hvilke kriterier som er viktige for å få en god 3D-modell ved bruk av fotogrammetri. (eliminere støy, lyssetting, kamerasetting osv)
3. Vurdere hvilke kriterier som er viktige i design av en rigg for 3D-modellering. (spesielle posisjoner av håndledd som er optimale for å lage en god 3D-modell, størrelse og form på rigg)
4. Produsere 3D-modeller ved bruk av fotogrammetri.
5. Produsere en prototype av ferdig produkt.

3.1.2 Oppgaver for medlemmer

- Ansvarsområder
- Arbeidsoppgaver

Torjus

Ansvarsområder:
-Elektronikk
-Raspberry Pi

Arbeidsoppgåver:
-Programmering av klient/server for Raspberry Pi.
-Oppsett av lokalt nettverk for flere klienter, med strømkilde og ethernet.

Astrid

Ansvarsområder:
Fotogrammetri
GUI

Arbeidsoppgåver:
-Finne og teste beste alternativ til fotogrammetri
-Utvikle en GUI

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

- Paul Steffen Kleppe, veileder
- Webjørn Rekdalsbakken, veileder
- Andreas Fagerhaug Dalen, kontaktperson

4 AVTALER

4.1 Avtale med oppdragsgiver

Det er kun blitt inngått muntlige avtaler med veiledere om at vi skal lage en rigg for framstilling av 3D-modeller som en del av et større prosjekt.

4.2 Arbeidssted og ressurser

- Tilgang til arbeidsplass
- Tilgang til ressurser
- Tilgang til personer
- Datasikkerhet/informasjon unndratt offentlighet
- Avtalt rapportering

Vår arbeidsplass vil være tunglabben hvor vi kan sette opp en testrigg for fotografering av objekter.

Til å begynne med bruker vi egne ressurser (dslr kamera og studiolys) til å foreta tester og avgjøre hvorvidt det er mulig å framstille en tilfredsstillende 3D-modell ut fra disse bildene. Hvis disse testene er positive vil det være nødvendig å teste kamera med dårligere kvalitet for å finne ut av minimum kvalitet på kamera nødvendig. I tillegg til dette vil det være nødvendig å designe et lukket miljø som gir ideelle forhold for fotografi. Prototypen vil i utgangspunktet ha form som en sylinder og flere sirkelforma åpninger for innsetting av lys og kamera.

Senere i prosjektet vil vi kjøpe inn flere Raspberry Pi med kamera, strømforsyning, LED-lys, router og ethernetkabel.

Prosjektet er hovedsakelig delt inn i to grupper. Den ene gruppen vil ha ansvar for å lage en fysisk modell av produktet, den andre gruppen har ansvar for elektronikken og software rundt produktet. Det vil underveis være nødvendig å kommunisere mellom gruppene om utforming, framgang og problemer som kan dukke opp. Vi vil også rådføre oss med oppdragsgiver for å vite at prototypen samsvarer med forventninger.

Webjørn Rekdalsbakken og Paul Steffen Kleppe er veiledere og vil bli kontaktet etter behov i tillegg til faste møter. Vi har også mulighet til å benytte resten av personalet ved NTNU Ålesund.

Rapportering vil foregå tirsdag annenhver uke klokken 14.00 fra og med 29.01.19 og til prosjektet er ferdig.

4.3 Gruppenormer – samarbeidsregler – holdninger

Avsnittet skal som hovedregel inneholde to hovedforhold.

En del som redegjør for hva slags normer en er enig om å legge til grunn for gruppen og samarbeidet i prosjektet.

En del som gir uttrykk for holdninger / perspektiver en ønsker å stå for som utøver av en profesjon. Hensikten med denne delen er å reflektere litt over rollen / profesjonen en går inn i som ferdig utdannet data/automasjonsingeniør, og begynne å formulere et grunnlag for holdninger / perspektiver en ønsker å stå for i den forbindelse.

- Medlemmer av gruppen skal møte presis til avtalt tid og sted. Ved uforutsette hendelser skal det gis beskjed så raskt som mulig.
- Viktige avgjørelser skal tas i plenum.
- Ved uenigheter skal veileder konsulteres.
- Arbeidsoppgaver og arbeidsmengde skal fordeles jevnt.

I et profesjonelt miljø vil enhver medarbeider bli påvirket, bevisst og ubevisst, av dine holdninger og verdier. Derfor er det viktig å opparbeide gode vaner og holdninger som påvirker miljøet rundt deg på en positiv måte. Det er viktig å tilegne seg gode vaner og holdninger som en kan holde på i en jobbsituasjon.

Noen av normene vi vil anvende under prosjekttiden er:

- Presis. Arbeidsdagen begynner hver dag kl 08:00. Er det avtalt et møte skal vi være der tidsnok, helst med god margin.
- Arbeidsvillig. Jobbe tilstrekkelig for at dagens gjøremål blir gjort, eventuelt jobbe lenger enn vanlig om nødvendig.
- Positiv. En god holdning vil påvirke medarbeidere positivt.
- Pålitelig. Over tid vise at du er en pålitelig arbeidstaker som får jobben gjort.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Formuleringer av den grunnleggende problemstillingen og hva en skal komme fram til i løpet av prosjektet – hovedmål og evt. delmål. Gjerne med en inndeling eller beskrivelse som skjelner mellom effektmål (verdimål), resultatmål og prosessmål.

"Ved armbrudd trengs det støttegips. Dette er en tidkrevende prosess. Prosessen kan automatiseres ved å «skanne» pasientens arm, lage en 3D-modell av armen med støtteskinne og deretter 3D-printe støtteskinna. Dette skal kunne gjøres i en sammenhengende operasjon."
-Forslag til bacheloroppgaver 2019.

Det overordnede målet med dette prosjektet er å komme fram til en komplett løsning som skanner og printer en skinne klar til bruk på 15-20 minutter. Et slikt prosjekt er omfattende og vår del vil være å lage en rigg med kameraer som kan produsere presise 3D-modeller.

Delmål vil være:

- Finne ut hvilke alternativer som finnes for 3D-skanning.
- Hvor mange kameraer vil være nødvendig.
- Hvor god må kvaliteten på kameraene være.
- Sette opp riggen.
- Hvilken form og styrke må lysene ha.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

- Formuleringer av spesifikasjoner, funksjonelle krav, standarder eller andre krav til en ferdig løsning eller resultat av prosjektet – inklusiv økonomiske rammer og krav til kvalitet.

- Leveranser fra prosjektet – hva skal anses som fullføring av prosjektet overfor oppdragsgiver i forhold til dokumenter, utviklet prototype/løsningsbeskrivelser og liknende.

Spesifikasjoner til riggen er foreløpig uvisst. Ideelt vil kostnadene være lavest mulig, og kvaliteten best mulig. Dette vil nødvendigvis måtte bli et kompromiss hvor kun flere tester vil kunne avsløre nødvendige spesifikasjoner.

Funksjonelt skal riggen kunne brukes av hvem som helst og likevel produsere gode resultater. Helst en løsning hvor man plasserer hånden inni en sylinder, trykker på en knapp og får ut en 3D-printet skinne.

Det er foreløpig ikke blitt diskutert økonomiske rammer rundt prosjektet. Kostnadene vil vise seg etter hvert som tester utføres.

5.3 Planlagt fremgangsmåte(r) for utviklingsarbeidet – metode(r)

- Begrunnelse for og kort beskrivelse av planlagt fremgangsmåte – slik som prosjektstyringsmetode og utviklingsmetode (eks. systemutviklingsmetode) og metodens(es) kjennetegn; dvs fokus, styrke(r) og mulig(e) svakhet(er). Som hovedregel skal en støtte seg på en kjent/anerkjent metode(r) for utvikling og styring og referere til hvilke(e) metode(r) som velges ved litteraturreferanse.

Hvis metoden(e) har kjente svakheter, skal det generelt sett også redegjøres for hvordan det planlegges å overkomme eller redusere disse.

Prosjektets fremgangsmåte blir "prøve og feile". Denne metoden er valgt fordi gruppen mangler erfaring innenfor dette feltet og ikke kjenner til mange fremgangsmåter.

Ved bruk av denne metoden har man lite oversikt fra starten, og det er viktig å dokumentere alle problemer og erfaringer man møter på underveis for å få mest mulig læringsutbytte. Blindveier kan bli dyre både i tid og penger, og det er derfor viktig å evaluere prosessen ofte for å unngå store feil.

5.4 Informasjonsinnsamling – utført og planlagt

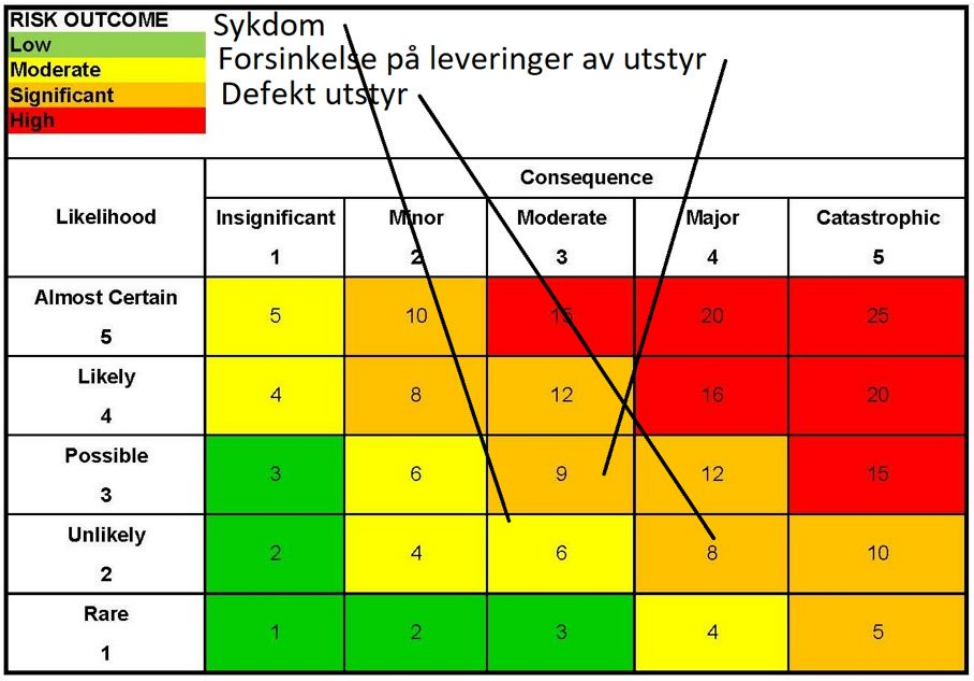
- Oversikt over informasjon om eksisterende anvendelser, systemløsninger eller kunnskap innenfor prosjektområdet, som en allerede har funnet fram til i arbeidet med forprosjektet.
- Oversikt over hvorfra og hvordan en videre vil sikre seg tilstrekkelig informasjon om eksisterende anvendelser, systemløsninger eller kunnskap innenfor prosjektområdet underveis i arbeidet med hovedprosjektet.

Eksisterende anvendelser av 3D-modellering er omfattende.
Innenfor industri er det vanlig å bruke 3D-modellering til prototyping, og ettersom kvaliteten på 3D-print blir bedre er det også blitt vanligere og bruke de i produksjon.
I helseindustrien er det gjort flere suksessfulle forsøk ved bruk av 3D-printede implanter. Bekken, kjeve og luftrør er noen av eksemplene. Det er også relativt nylig blitt laget beholdere til piller med fordeler ovenfor den konvensjonelle beholderen.
Andre industrier hvor 3D-printing er brukt: klesindustrien, bilindustri, paleontologi og bioteknologi er noen eksempler.

Spesifikt for vårt prosjekt vil vi ta i bruk fotogrammetri. Dette er teknologi som ved hjelp av bilder observerer og bestemmer egenskaper som for eksempel beliggenhet, form, størrelse og identitet for avbildet terreng eller andre gjenstander. Fotogrammetri blir allerede tatt i bruk

innenfor områder som topografiske kart, arkitektur, politi etterforskning, geologi og arkeologi. Siden bruken av fotogrammetri er så omfattende er det flere ressurser på internett som omtaler hvordan man skal gjennomføre fotogrammetri. Derfor vil vi hovedsakelig bruke internett som ressurs for informasjon, og rådføre oss med veiledere om vi skulle ha spørsmål.

5.5 Vurdering – analyse av risiko



Likelihood	Consequence				
	Insignificant 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
Almost Certain 5	5	10	15	20	25
Likely 4	4	8	12	16	20
Possible 3	3	6	9	12	15
Unlikely 2	2	4	6	8	10
Rare 1	1	2	3	4	5

5.6 Hovedaktiviteter i videre arbeid

Uke:	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Kartlegging av oppgave																		
Planlegging																		
Innhenting av informasjon																		
Testing av kamera																		
Fotogrammetri tester																		
Server/Klient programmering																		
Elektronikk																		
GUI																		
Rapport og logg																		

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

- Hovedtrekk i gjennomføringen
- Beskrivelse av planlagte hovedaktiviteter i forhold til listen under pkt 5.6, om nødvendig med kort beskrivelse av viktige underaktiviteter (hvilke, innhold, oppgavefordeling, ansvar, forventet starttidspunkt, sluttidspunkt osv.)

Milepæler (hva skal være oppnådd - når) – resultater / leveranser
Beslutningsprosess - viktige beslutningspunkter (når, om hva, av hvem)

1. Kartlegging av oppgave.

Finne ut hva som forventes av oppgaven utover beskrivelsen ved å rådføre oss med veiledere Paul Steffen og Webjørn.

2. Planlegging.

Legge en plan for hvordan vi skal tilnærme oss problemet. Underveis i prosjektet vil det mest sannsynlig bli nødvendig å gjenoppta denne aktiviteten siden det ofte kan oppstå uforutsette situasjoner og omstendigheter.

3. Innhenting av informasjon.

Som i de fleste prosjekter vil vi finne informasjon på internett. Fotogrammetri er allerede etablert innenfor flere yrker, og vi burde derfor ikke ha noen problem med å finne nok informasjon.

4. Fotogrammetri tester.

I begynnelsen av prosjektet vil vi gjøre oss kjent med fotogrammetri og foreta flere tester. De første testene vil være viktig med tanke på videre mulighet for å realisere prosjektet. Her vil vi bruke kamera av høy kvalitet for å fremstille en nøyaktig 3D-modell. Videre vil vi gradvis senke kvaliteten på kamera for å finne ut minimum kvalitet nødvendig.

5. Prototype av kamerarigg.

Sammenstilling av prototype. Når gruppen fra POD har laget en prototype av sylinderen skal vi prøve å sette sammen alle komponentene. For å designe en prototype er det en del parametre som skal optimaliseres. Antall kameraer, antall lys og materiale på sylinder er alle viktige parameter som kan øke kvalitet og redusere kostnader.

6. Rapport og logg.

For å unngå skippertak vil vi hele tiden jobbe med rapporten og føre logg over hva vi gjør.

5.7.2 Styringshjelpemidler

- Oversikt over hjelpemidler en ønsker å bruke i arbeidet med å styre prosjektet
- Eksempler, illustrasjoner (fra hjelpemidlene) som viser planleggingen av prosjektet der, med minimum: aktiviteter, tid, arbeidsfordeling og ansvar, milepæler / leveranser og økonomi

For å styre prosjektet og holde oversikt over fremgangen har vi valgt å lage et gantt-diagram som skal oppdateres fortløpende og vises frem ved de faste møtene med veiliederne.

5.7.3 Utviklingshjelpemidler

- Oversikt over hjelpemidler en vil ha behov for eller ønsker å bruke i arbeidet med å gjennomføre prosjektet. (Dette kan ses i sammenheng med punkt 9)

Viser til punkt 9 hvor vi går grundig gjennom dette.

5.7.4 Intern kontroll – evaluering

- om hvordan intern kontroll i prosjektet, oppfølging av fremdrift osv., vil bli gjennomført
- evaluering: hva skal være kriterier/kjennetegn på at mål/delmål er nådd?

Medlemmene i gruppen er fri til å jobbe med sine arbeidsoppgave når som helst og hvor som helst. For å sikre fremdrift og å unngå misforståelser skal medlemmene møtes minst to ganger i uken for en intern kontroll og oppdatering.

Annenhver uke møtes de to gruppene i prosjektet og de to veilederne til et rapporteringsmøte.

5.8 Beslutninger – beslutningsprosess

- Informasjon om hvordan beslutninger om avgrensning / presisering av oppgaven og andre sentrale beslutninger har blitt tatt under arbeidet med forprosjektet.
- Oversikt over hvordan viktige beslutninger planlegges tatt under arbeidet med hovedprosjektet. Dette gjelder hovedområder og viktige avgjørelser som skal/må tas underveis i arbeidet, slik det er forutsatt i hovedplanen (5.7.1).

Oppgaven vi ble gitt var allerede godt definert, og det ble derfor ikke nødvendig å ta noen videre beslutninger på hva vi skulle gjøre. Vi var også enig med Paul Steffen og Webjørn om hvordan vi skulle gå fram og hvordan et sluttprodukt ideelt ville være.
En del av problemet med dette prosjektet er at det er usikkert hvorvidt det er mulig å gjennomføre ved bruk av fotogrammetri. Og hvis det er mulig, hva vil være optimalt utstyr med tanke på kvalitet/kostnad. Usikkerhet rundt dette kan være svært kostbart, og det vil derfor være nødvendig å gjøre omfattende tester rundt dette. Alle beslutninger vil derfor tas etter grundig testing.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

- Hva slags dokumentasjon skal utarbeides – utforming, innhold
- Rutiner

- Godkjennelse
- Distribusjon / kopiering
- Oppbevaring
- Vedlikehold

En styringsrapport vil bli laget, og oppdatert før hvert styringsmøte. Denne vil inneholde alle arbeidsoppgaver med startdato og forventet sluttdato, status på oppgavene og en statusrapport som forklarer hvordan prosjektet utvikler seg.

Bachelorrapporten vil dokumentere alt vi har gjort og leveres inn ved slutten av prosjektperioden.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

- Planlagte møtedatoer/tidspunkt – innhold, rapportering etc.

Møter med styringsgruppen vil ta sted annenhver tirsdag klokken 14.00 fra og med 29.01.18 og ut prosjekttiden. Disse møtene vil ta for seg framdrift, problemer og framtidige planer.

7.1.2 Prosjektmøter

- Planlagte møtedatoer/tidspunkt – hensikt

Foruten møter med styringsgruppen er det foreløpig ikke andre planlagte møter. De to gruppene i prosjektet kan trenge å møtes oftere enn annenhver uke, men disse møtene blir avtalt etter behov. Om det blir nødvendig å få ekstern hjelp eller mulighet til møter med fagfolk vil naturligvis det bli planlagt møter etter behov.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

- Planlagt(e) rapportform(er)
- Planlagte rapportdatoer

Det vil bli laget et enkelt dokument(styringsrapport) som vil inneholde hovedmål og delmål. Foran hvert møte med styringsgruppen vil vi oppdatere dette dokumentet.

8 PLANLAGT AVVIKSBEHANDLING

- Hva skal gjøres dersom prosjektet (fremdrift/innhold) ikke går som planlagt.
- Planlagt prosedyre for endringer
- Ansvar

Ved avvik i planene vil vi konsultere veiledere for råd om hvordan vi skal løse problemet.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- Utstyr / programvare eller andre spesielle resurser som en vanligvis ikke har tilgang til og som er nødvendig for å gjennomføre prosjektet
- Eventuelt spesialutstyr / programvare som det søkes om innkjøp av- begrunnes

(Vanligvis vil det være oppdragsgivers ansvar å stille slikt utstyr og programvare til disposisjon for prosjektgruppen)

Hardware

- 4 x Raspberry Pi + kamera
- Strømforsyning
- Deksel til Raspberry Pi
- LED

Software

- Programvare til fotogrammetri

Under dette prosjektet vil det bli behov for å bruke en del utstyr. I første del vil vi ta i bruk et privat kamera i tillegg til at jeg låner studiolys fra Sunnmørsposten. Software vi bruker for å teste i begynnelsen vil være 3DF Zephyr Free som er gratis, men det kan bli aktuelt å bruke software fra xtura om Paul Steffen skaffer lisens.

VEDLEGG 2
Møtereferat

Møtereferat

Onsdag, 09.00, 30 januar, Rundskue NTNU Ålesund

Tilstede:

Automasjon Torjus Aa. Hoseth, Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Agenda:

- Definere Bacheloroppgave
- Fordeling av arbeidsoppgaver
- Tidligere bacheloroppgave

Vi gikk gjennom hva som var gjort tidligere, og resultatet av denne oppgaven. Arbeidsoppgavene ble i grove trekk fordelt, og vi fikk litt tidsrom til å undersøke forskjellige tilnærminger til oppgaven. Kravspesifikasjoner som behandlingstid og kvalitet ble også nevnt.

Møtereferat

Torsdag, 09.15, 14 februar, Åse sykehus

Tilstede:

Automasjon Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere

Sykehus Andreas Fagerhaug Dalen, Aleksander Skrede

Agenda:

- Omvisning
- Gjennomgang av prosedyre
- Møte med masterstudent

Vi møtte opp på sykehuset hvor vi fikk omvisning av lokalet hvor pasienter gipses. Andreas gikk gjennom prosessen med gipsing, og forklarte oss om prosedyren. Videre fikk vi omvisning på NTNU sin testlab hvor masterstudent Aleksander Skrede jobber.

Møtereferat

Onsdag, 10.00, 20 februar, Rundskue NTNU Ålesund

Automasjon Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Siemens Marius Slagsvoll

Agenda:

- Diskuter sykehusbesøket
- Diskuter valg av metode for gjennomføring av prosjekt
- Mini workshop med Siemens NX

Vi diskuterte besøket på sykehuset og hva vi hadde lært av det. Snakket også en del om NTNU sin testlab og hva som foregikk der. Vi gikk også i dybden på hvilken metode vi hadde valgt for å gjennomføre prosjektet og teknologien som lå bak. POD studentene fikk også gjennomgang av Siemens NX med Marius Slagsvoll.

Møtereferat

Onsdag, 09:30, 6 mars, Paul Steffen sitt kontor

Automasjon Torjus Aa. Hoseth, Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Agenda:

- Diskuter progresjon
- Kom med forslag til utstyr som må kjøpes inn

Vi gikk gjennom hvor langt vi var kommet og kom med forslag til hva utstyr vi trengte videre. Vi fikk tillatelse til å gjøre innkjøp av atten Raspberry Pi med tilhørende utstyr. Vi diskuterte også med POD gruppen videre utforming av prototypen.

Møtereferat

Torsdag, 09:00, 21 mars, Paul Steffen sitt kontor

Automasjon Torjus Aa. Hoseth, Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Agenda:

- Elektronikk
- Prototype
- Overlapp mellom kamera

Vi gikk gjennom hvordan vi hadde koblet systemet med POE konseptet og all "spaghettien" som fulgte med dette. Prototypen fant vi ut var for liten med tanke på graden av overlapp bildene måtte ha med hverandre.

Møtereferat

Onsdag, 09:00, 3 april, Rundskue NTNU Ålesund

Automasjon Torjus Aa. Hoseth, Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Agenda:

- Progresjon
- Programvare struktur

Siden sist møte var det ikke så mye progresjon å melde. Vi hadde et par bugs i koden som hindret oss i å gå videre, noe som gjorde at vi ikke hadde så mye å melde på dette møtet. Strukturen på programmet ble også gått gjennom i detalj for å få innspill, men vi var alle enige om at det som hadde blitt gjort var fornuftig.

Møtereferat

Onsdag, 09:00, 24 april, Rundskue NTNU Ålesund

Automasjon Torjus Aa. Hoseth, Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Agenda:

- Progresjon
- Resultater
- Forslag til endringer

Dette var det siste møtet hvor vi kunne gjøre endringer på prosjektet før det måtte leveres inn. Det ble gått gjennom progresjon siden sist møte, og vi presenterte resultatene vi hadde oppnådd så langt. Resultatet vi hadde oppnådd demonstrerte at det var mulig å realisere prosjektet, men ikke innen tidsrammen vi hadde. I tillegg ville prosjektet kreve endringer, og bedre kamera.

Møtereferat

Onsdag, 09:00, 24 april, Tunglabben NTNU Ålesund

Automasjon Torjus Aa. Hoseth, Astrid Nerhus Dale

POD Denis Sjåstad, Oscar Hatlo Nedrelid, Vegard André Alvestad

Veiledere Webjørn Rekdalsbakken, Paul Steffen Kleppe

Agenda:

- Rapport
- Endelig resultat
- Detaljer rundt innlevering

Det siste møtet handlet mer om formelle ting enn selve prosjektet. Her gikk vi gjennom hva som skulle leveres inn, til hva tid og lignende problemstillinger. Vi diskuterte for siste gang hva vi hadde oppnådd i løpet av prosjekttiden og læringsutbyttet.

VEDLEGG 3

Kode

```
import socket
import os
import struct
import time
import subprocess
import sys
import threading
import signal
import re
import os
import Queue
from os.path import isfile, join
from os import listdir


NUMBER_OF_THREADS = 2
JOB_NUMBER = [1,2]
queue = Queue.Queue()

class Client(object):
    #Client constructor
    def __init__(self):
        self.server_address = ""
        self.port = 9999
        self.tcp_pip = (self.server_address, self.tcp_port)
        self.TCP_sock = None
        self.MCAST_sock = None
        self.MCAST_GRP = '224.1.1.1'
        self.MCAST_PORT = 5007
        self.picturePath = "/home/pi/Pictures"
        self.clientPath = "/home/pi/Documents/PythonProjects/"
        self.oldClientPath = "/home/pi/Documents/PythonProjects/old/"
        self.currentClientPath = "/home/pi/Documents/PythonProjects/current/"
        self.newClientPath = "/home/pi/Documents/PythonProjects/new/"
        self.clientVersion = ""
        self.pictureName = None
        self.imageData = None
        self.clientID = ""
        self.cameraSettings = ""
        self.dead = False

    ##Signal handler, find out how to use it and what exactly it does***
    def register_signal_handler(self):
        signal.signal(signal.SIGINT, self.quit_program)
        signal.signal(signal.SIGTSTP, self.quit_program)
        return

    ##Quit down all connections and quit the program***
    def quit_program(self, signal=None, frame=None):
        if self.TCP_sock:
            try:
                self.TCP_sock.shutdown(2)
                self.TCP_sock.close()
            except Exception as e:
                print("Error closing server connection %s" %str(e))
                self.TCP_sock.close()
                #sys.exit(0)
            sys.exit(0)
            return

    ##Create a mcast socket for receiving commands simultaneously with other Raspberry Pi's***
    def create_mcast_socket(self):
        self.MCAST_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
        self.MCAST_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.MCAST_sock.bind(('', self.MCAST_PORT))
        self.MCAST_sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, socket.inet_aton(self.MCAST_GRP) + socket.inet_aton('0.0.0.0'))

    ##Create a tcp socket***
    def create_tcp_socket(self):
        try:
            self.TCP_sock = socket.socket()
        except socket.error as e:
            print ("Socket creation error: " + str(e))
            return
        return

    ##Connect the tcp socket***
    def connect_tcp_socket(self):
        try:
            self.TCP_sock.connect((self.server_address, self.tcp_port))
        except socket.error as e:
            print("Socket connection error: " + str(e))
            time.sleep(5)
        try :
            self.TCP_sock.send(str.encode(self.get_address()) + " " + self.clientID)
        except socket.error as e:
            print ("Cannot send hostname to server: " + str(e))
            time.sleep(5)
        return

    ##Reset the tcp socket in case another server wants to connect***
    def reset_tcp_socket(self):
        try:
            if self.TCP_sock:
                print("Sock shutdown")
                self.TCP_sock.shutdown(2)
                self.TCP_sock.close()
                time.sleep(1)
                self.TCP_sock = None
                print("Sock reset")
                self.dead = True
                print("Sock dead")
            else:
                print("reset_tcp_socket method else statement")
        except socket.error as e:
                print("Error, the server has disconnected, resetting client connection. Error: " + str(e))
                self.TCP_sock = None
                self.dead = True

    ##Helper method for resetting, creating and connecting a tcp socket***
    def reset_create_connect_tcp(self):
        self.reset_tcp_socket()
        self.create_tcp_socket()
        self.connect_tcp_socket()
        self.dead = False

    ##Client to the multicast network***
    def mcast_listen(self, event):
        eventHandler = event
        while True:
            try:
                print(threading.currentThread().getName())
                data, addr = self.MCAST_sock.recvfrom(1024)
                tot = str(data).split(" ")
                if(tot[0] == "setting"):
                    print("Setting camera " + tot[1]
                    for setting in tot[2:]:
                        self.cameraSettings = setting + " "
                    if self.TCP_sock:
                        if(self.clientID == 1):
                            subprocess.call("sudo python3 /home/pi/Documents/PythonProjects/lights.py 0.3", shell=True)
                            time.sleep(1)
                            eventHandler.clear()
                            self.take_picture()
                            subprocess.call("sudo python3 /home/pi/Documents/PythonProjects/lights.py 0", shell=True)
                            eventHandler.set()
                        else:
                            time.sleep(1)
                            eventHandler.clear()
                            self.take_picture()
                            eventHandler.set()
                    else:
                        pass
                elif(data == "shutdown"):
                    cmd = "sudo poweroff"
                    pid = subprocess.call(cmd, shell=True)
                elif(tot[0] == "reset"):
                    cmd = "sudo reboot"
                    pid = subprocess.call(cmd, shell=True)
                elif(tot[0] == "reset_tcp_socket"):
                    self.reset_tcp_socket()
                    self.dead = True
                elif(tot[0] == "new_client"):
                    if(self.server_address != tot[1]):
                        self.server_address = tot[2]
                        if not self.TCP_sock:
                            print("Setting TCP multithread")
                            eventHandler.set()
                        else:
                            print(threading.active_count())
                            self.renew_tcp_socket()
                    else:
                        if self.TCP_sock:
                            pass
                        else:
                            self.renew_tcp_socket()
                else:
                    pass
            except Exception as e:
                print("Error occured receiving mcast data " + str(e))

    ##Receive commands***
    def receive_commands(self, event):
        eventHandler = event
        while True:
            print("while true")
            event.wait()
            while (not self.dead):
                print("This is the end statement")
                try:
                    print("try statement")
                    print(threading.currentThread().getName())
                    data = self.TCP_sock.recv(1024)
                    tot = str(data).split(" ")
                    message = ""
                    print("Message")
                    if(tot[0] == "ack"):
                        eventHandler.wait()
                        print("eventHandler wait")
                        if self.imageData:
                            print("Sending send_picture method")
                            self.send_picture()
                        else:
                            print("Message = ack")
                            self.send_ack()
                    elif(data == "shutdown"):
                        message = self.clientID
                    elif(tot[0] == "shutdown"):
                        cmd = "sudo poweroff"
                        pid = subprocess.call(cmd, shell=True)
                    elif(tot[0] == "reset"):
                        cmd = "sudo reboot"
                        pid = subprocess.call(cmd, shell=True)
                    elif(tot[0] == "reset"):
                        self.reset_tcp_socket()
                        self.create_tcp_socket()
                        self.connect_tcp_socket()
                        self.receive_commands(event)
                    elif(tot[0] == "new_client"):
                        self.server_address = tot[2]
                        self.reset_create_connect()
                    else:
                        pass
                    if(message):
                        self.TCP_sock.sendall(message.encode())
                except Exception as e:
                    print("Error receiving data through TCP socket, resetting connection " + str(e))

    ##Method for sending the client program***
    def get_update(self):
        try:
            hs = self.TCP_sock.recv(128)
            newlength = struct.unpack('!I', hs)
            data = ''
            while len(data) < length:
                to_read = length - len(data)
                data += self.TCP_sock.recv(4096 if to_read > 4096 else to_read)
            #Write the new client program to a file
            with open(self.newClientPath + "new_client" + ".py", 'w') as fp:
                fp.write(data)
                fp.close()
        except Exception as e:
            print("Couldn't receive update " + str(e))

    ##Quit for picture update and restart client to try the new software***
    def quit_update(self):
        newClient = None
        currentClient = None
        newClientDescription = "new_client" + ".py"
        currentClientDescription = "current_client" + ".py"
        oldClientDescription = "old_client" + ".py"

        self.clear_folder(self.oldClientPath)

        newClient = self.read_file(self.newClientPath, newClientDescription)
        currentClient = self.read_file(self.currentClientPath, currentClientDescription)

        self.write_file(self.oldClientPath, currentClientDescription, currentClient)
        self.change_file_name(self.oldClientPath, currentClientDescription, oldClientDescription)
        self.clear_folder(self.currentClientPath)

        self.write_file(self.currentClientPath, currentClientDescription, newClient)
        self.change_file_name(self.currentClientPath, currentClientDescription, currentClientDescription)
        self.clear_folder(self.newClientPath)

    ##Start script that restarts client with new software***
    def restart_client(self):
        subprocess.call("python /home/pi/Documents/PythonProjects/restart.py", shell=True)
        print("Before quit_program")
        self.quit_program()

    ##Method for taking picture***
    def take_picture(self):
        print("Before self.picturePath")
        self.imageData = None
        tries = 0
        while tries < 5:
            cmd = "raspistill -n -t 3000 " + self.cameraSettings + " -o " + self.picturePath + self.pictureName + self.clientID + ".jpg"
            pid = subprocess.call(cmd, shell=True)
            self.read_picture()
            if self.imageData:
                tries = 5
            else:
                tries += 1
                print("Could not take picture, try %" + str(tries))
                time.sleep(1)

    ##Method for setting camera settings***
    def set_cameraSettings(self):
        pass

    ##Read the picture data and store it in the local variable***
    def read_picture(self):
        print("reading")
        try:
            with open(self.picturePath + self.pictureName + self.clientID + ".jpg", 'rb') as fp:
                self.imageData = fp.read()
                fp.close()
        except Exception as e:
            print("Could not find my picture, " + str(e))

    ##Send the most recently taken picture over the TCP connection***
    def send_picture(self):
        assert(len(self.imageData))
        print("Starting to send")
        length = struct.pack('!I', len(self.imageData))
        self.TCP_sock.sendall(length)
        self.TCP_sock.sendall(self.imageData)
        self.TCP_sock.sendall(self.imageData)
        print("Image data sent")

    ##Clear contents of a given folder***
    def clear_folder(self, folder):
        for file in os.listdir(folder):
            file_path = os.path.join(folder, file)
            try:
                if os.path.isfile(file_path):
                    os.unlink(file_path)
            except Exception as e:
                print("Could not clear picture folder, error: " + str(e))

    ##Write content to a given file***
    def write_file(self, folder, description, data):
        try:
            with open(folder + description, 'wb') as fp:
                fp.write(data)
                fp.close()
        except Exception as e:
            print("Could no write contents, to folder: " + folder + ", Error: " + str(e))

    ##Read content of a given file in a give folder***
    def read_file(self, folder, description):
        contents = None
        try:
            with open(folder + description, 'rb') as fp:
                contents = fp.read()
                fp.close()
        except Exception as e:
            print("Could not read file: " + folder + description + " contents, Error: " + str(e))
        return contents

    ##Change filename in a file to another given path to something new***
    def change_file_name(self, path, old, new):
        if os.path.isfile(path + old):
            os.rename(path + old, path + new)
        else:
            print("Cannot find file on path: " + path)

    ##Method for checking if it's necessary to update client***
    def compare_client_version(self, version):
        currentVersion = self.get_client_version()
        try:
            if(float(version)>float(currentVersion)):
                return True
            else:
                return False
        except Exception as e:
            print("Could not update client, error: " + str(e))

    ##Return the current version of client program being used***
    def get_client_version(self):
        self.clientVersion = self.get_newest_version(self.currentClientPath)
        print("Client version: " + self.clientVersion)
        return self.clientVersion

    ##Find the newest client version of a given folder***
    def get_newest_version(self, folder):
        newestVersion = ""
        try:
            c = re.compile('client')
            p = re.compile('\d\.\d')
            fileInDirectory = os.listdir(folder)
            for files in fileInDirectory:
                m = c.match(files)
                if m:
                    version = p.search(files)
                    if version:
                        versionString = version.group()
                        if not newestVersion:
                            newestVersion = versionString
                        elif(float(versionString)>float(newestVersion)):
                            newestVersion = versionString
                        else:
                            pass
                    else:
                        continue
        except Exception as e:
            print("Couldn't find client file, error: " + str(e))
        return newestVersion

    ##Helper method for asking if something has been completed***
    def get_ack(self):
        self.TCP_sock.settimeout(2)
        buf = ''
        try:
            while '\n' not in buf:
                buf += self.TCP_sock.recv(1).decode()
                num = int(buf)
                if num == 1:
                    return True
                if num == 0:
                    return False
                else:
                    return
        except socket.timeout as e:
            print("Server took too long responding")
        self.TCP_sock.settimeout(None)

    ##Helper method for confirming something has been completed***
    def send_ack(self, ack):
        ans = ''
        if ack == 1 or ack == 0:
            ans = str(ack) + ans
            self.TCP_sock.sendall(ack.encode())
        else:
            print("Not valid ack input")
        assert len(ans) == 2
        self.TCP_sock.sendall('\n')

    ##Get the local address of the Pi***
    def get_address(self):
        s = socket.gethostbyname(socket.gethostname())
        return s

    ##Get the ID of this raspberry pi and assign the clientID the value***
    def get_ID(self):
        with open("/home/pi/Documents/ID.txt", 'rb') as fp:
            self.clientID = fp.read()
            return

    ##Create all worker threads (2) each***
    def get_client(self):
        return self.clientID

    ##Stop thread***
    def stop_thread(self, event):
        print("Before wait before event.wait()")
        event.wait()
        print("Not blocked after event.wait()")
        self.create_tcp_socket()
        self.create_tcp_socket()
        self.connect_tcp_socket()
        self.receive_commands(event)


    ##Reset thread***
    def reset_thread(self, event):
        self.set_ID()
        time.sleep(5)#Delay so that the switch is ready receive multicast broadcast
        self.create_mcast_socket()
        self.mcast_listen(event)

def create_workers():
    client = Client()
    event = threading.Event()
    client.register_signal_handler()

    for _ in range(NUMBER_OF_THREADS):
        t = threading.Thread(target=work, args=(client, event))
        t.daemon = True
        t.start()
    return

def work(client, event):
    while True:
        x = queue.get()
        if x == 1:
            client.mcast_thread(event)
        if x == 2:
            client.tcp_thread(event)
        queue.task_done()
    return

def create_jobs():
    for x in JOB_NUMBER:
        queue.put(x)
    queue.join()
    return

def main():
    create_workers()
    create_jobs()

if __name__ == "__main__":
    main()
```

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'design.ui'
#
# Created by: PyQt5 UI code generator 5.10.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(792, 719)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setContentsMargins(-1, 0, -1, -1)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.btnConnect = QtWidgets.QPushButton(self.centralwidget)
        self.btnConnect.setObjectName("btnConnect")
        self.horizontalLayout.addWidget(self.btnConnect)
        self.btnPicture = QtWidgets.QPushButton(self.centralwidget)
        self.btnPicture.setObjectName("btnPicture")
        self.horizontalLayout.addWidget(self.btnPicture)
        self.btnDownload = QtWidgets.QPushButton(self.centralwidget)
        self.btnDownload.setObjectName("btnDownload")
        self.horizontalLayout.addWidget(self.btnDownload)
        self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_4.setObjectName("pushButton_4")
        self.horizontalLayout.addWidget(self.pushButton_4)
        self.pushButton_5 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_5.setObjectName("pushButton_5")
        self.horizontalLayout.addWidget(self.pushButton_5)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setObjectName("label")
        self.verticalLayout.addWidget(self.label)
        self.listView = QtWidgets.QListView(self.centralwidget)
        self.listView.setMaximumSize(QtCore.QSize(200, 16777215))
        self.listView.setObjectName("listView")
        self.verticalLayout.addWidget(self.listView)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menuBar = QtWidgets.QMenuBar(MainWindow)
        self.menuBar.setGeometry(QtCore.QRect(0, 0, 792, 21))
        self.menuBar.setObjectName("menuBar")
        self.menuControl = QtWidgets.QMenu(self.menuBar)
        self.menuControl.setObjectName("menuControl")
        self.menuConfig = QtWidgets.QMenu(self.menuBar)
        self.menuConfig.setObjectName("menuConfig")
        self.menuAbout = QtWidgets.QMenu(self.menuBar)
        self.menuAbout.setObjectName("menuAbout")
        MainWindow.setMenuBar(self.menuBar)
        self.menuBar.addAction(self.menuControl.menuAction())
        self.menuBar.addAction(self.menuConfig.menuAction())
        self.menuBar.addAction(self.menuAbout.menuAction())

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.btnConnect.setText(_translate("MainWindow", "Connect"))
        self.btnPicture.setText(_translate("MainWindow", "Take Picture"))
        self.btnDownload.setText(_translate("MainWindow", "Download"))
        self.pushButton_4.setText(_translate("MainWindow", "PushButton"))
        self.pushButton_5.setText(_translate("MainWindow", "PushButton"))
        self.label.setText(_translate("MainWindow", "PiCameras"))
        self.menuControl.setTitle(_translate("MainWindow", "Control"))
        self.menuConfig.setTitle(_translate("MainWindow", "Config"))
        self.menuAbout.setTitle(_translate("MainWindow", "About"))
```

```
from PyQt5.QtGui import*
from PyQt5.QtWidgets import*
from PyQt5.QtCore import*
import sys

from scanner import Ui_MainWindow

class MainWindow(QMainWindow, Ui_MainWindow):

    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)
        self.setupUi(self)

        self.show()
        self.pushButton.clicked.connect(self.hello)


    def hello(self):
        print("Hello World")

if __name__ == '__main__':

    app = QApplication([])
    window = MainWindow()
    app.exec_()
```

```python
#! python3
import socket
import struct
import sys
import time
import signal
import threading
import os
import re
from os import listdir
from os.path import isfile, join
from queue import Queue

NUMBER_OF_THREADS = 3
JOB_NUMBER = [1, 2, 3]
queue = Queue()

COMMANDS = {'help' :['Description of all commands'],
            'select' :['Select a client by its index, takes index as param'],
            }

PICTURE_SETTINGS = {'standard' : '-o',
            'contrast' : '-co 50',
            }


class Server(object):
    def __init__(self):
        self.mcast_port = 5000 #Port number that will be used for multicast communication. Maybe add a method for changing port?
        self.tcp_port = 6000 #Port number that will be used for tcp communication. Maybe add a method for changing port?
        self.mcast_grp = ('224.1.1.1', self.mcast_port) #First parameter is a ip address specifically reserved for multicast traffic. Second parameter is the port which will be used f
        self.tcp_grp = (self.get_address(), self.tcp_port)#First parameter is a method which finds the IP address of the server device running the program.
        self.MCAST_sock = None #Multicast socket needed for communicating with several (RaspPI) devices at the same time. Taking pictures, changing server device connected to all rasp
        self.TCP_sock = None #TCP socket needed for communication through TCP. Used for sending pictures from raspPIs to server, identifying each raspPI unit, sending messages to indi
        self.all_connections = [] #Store all socket objects in a list (Socket connection with each PI used for communication)
        self.all_addresses = [] #Store all the addresses of the PIs connected
        self.all_clients = [] #Store the personal ID of each pi and order them ascending
        self.pictureName = None #Name for the folder with most recent set of pictures
        self.pictureData = None #Variable to temporarily hold the picture data, set to None after finished using
        self.picturePath = 'C:/Python27/pythonpractice/pictures/' #TODO: maybe add a method for changing this location
        self.clientPath = 'C:/Python27/pythonpractice/client/'
        self.clientVersion = "" #TODO: find a way to update the PIs with new software and restart them
        self.numberOfClients = 18
        self.cameraSettings = "-co 50 -sh 50"

    """Close the connections and shut down the program"""
    def quit_program(self, signal=None, frame=None):
        print('\nQuitting the program')
        for i, conn in enumerate(self.all_connections):
            try:
                conn.shutdown(2)
                conn.close()
            except Exception as e:
                print('Could not close connection to PI #' + self.all_clients[i] + ': %S' % str(e)) #Figure out a way to get unique ID of each raspPI
                continue
        self_all_clients = []
        self.all_addresses = []
        self.TCP_sock.close()
        sys.exit(0)

    """Signal handler"""
    def register_signal_handler(self):
        signal.signal(signal.SIGINT, self.quit_program)
        signal.signal(signal.SIGTERM, self.quit_program)
        return

    """Print all the help alternatives"""
    def print_help(self):
        for cmd, v in COMMANDS.items():
            print ("{0}:\t{1}".format(cmd, v[0]))
        return

    """Create a socket and set it up for multicasting"""
    def create_mcast_socket(self):
        try:
            self.MCAST_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
            self.MCAST_sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 32)
        except socket.error as e:
            print ("Socket creation error: " + str(e))
            return
        return

    """Send a multicast message"""
    def mcast_message(self, message):
        if(message == 'address'):
            message += ' ' + str(self.get_address())
        elif(message == 'picture'):
            description = input('Description> ')
            message += ' ' + description + ' ' + self.cameraSettings
            self.pictureName = description
            self.create_folder()
        SCMD = message.encode()
        self.MCAST_sock.sendto(SCMD, self.mcast_grp)

    """Create a socket for TCP connection"""
    def create_tcp_socket(self):
        try:
            self.TCP_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        except socket.error as e:
            print ('Error trying to create socket: ' + str(e))
            return
        return

    """Bind tcp socket to a port and wait for connections"""
    def bind_tcp_socket(self):
        try:
            self.TCP_sock.bind(self.tcp_grp)
            self.TCP_sock.listen(5)
        except socket.error as e:
            print ('Socket binding error: ' + str(e))
            time.sleep(5)
            self.bind_tcp_socket()
        return

    """Close the TCP connection"""
    def close_tcp_socket(self):
        try:
            self.TCP_sock.shutdown(2)
            self.TCP_sock.close()
            self.TCP_sock = None
        except Exception as e:
            print("Could not close TCP socket: " + str(e))
        return
```

```
"""Reset all the clients, or a specific client"""
def reset_clients(self, client=""):
    if not client:
        self.mcast_message('reset')
        for c in self.all_connections:
            try:
                c.shutdown(2)
                c.close()
            except Exception as e:
                print('Connection has not yet been established' + str(e))
        self.all_clients = []
        self.all_connections = []
        self.all_addresses = []
    else:
        clientID = int(client)
        try:
            target = self.all_connections[clientID]
            target.send('reset')
            target.close()
        except Exception as e:
            print("Could not get client #" + client + ". Error: " + str(e))
        self.all_clients.remove(clientID)
        self.all_connections.remove(clientID)
        self.all_addresses.remove(clientID)

""" Accept connections from multiple clients and save to list """
def accept_connections(self):
    for c in self.all_connections:
        c.close()
    self.all_connections = []
    self.all_addresses = []
    self.all_clients = []

    while True:
        try:
            conn, address = self.TCP_sock.accept()
            conn.setblocking(1)
            client_hostname = conn.recv(1024).decode("utf-8") #Receive the client hostname + client ID e.g. 169.254.125.215 + 1
            txt = str(client_hostname).split()
            address = address + (txt[0],) #First address is the address bound to the socket on the other end of the connection. Second address is address of the client.
            clientID = int(txt[1])
        except Exception as e:
            print('Error accepting connections: %s' % str(e))
            continue
        self.append_clients(clientID, address, conn)
        print('\nConnection has been established: {0} ({1})'.format(address[-1], address[0]))
    return

"""Order all lists in ascending order according to their raspberry ID"""
def append_clients(self, clientID, address, conn):
    try:
        if not self.all_clients:
            self.all_clients.append(clientID)
            self.all_addresses.append(address)
            self.all_connections.append(conn)
        elif clientID < self.all_clients[0]:
            self.all_clients.insert(0, clientID)
            self.all_addresses.insert(0, address)
            self.all_connections.insert(0, conn)
        elif clientID > self.all_clients[-1]:
            self.all_clients.append(clientID)
            self.all_addresses.append(address)
            self.all_connections.append(conn)
        else:
            for i, value in enumerate(self.all_clients):
                if clientID > value and clientID < self.all_clients[i+1]:
                    self.all_clients.insert(i+1, clientID)
                    self.all_addresses.insert(i+1, address)
                    self.all_connections.insert(i+1, conn)
                elif clientID == value:
                    pass
    except Exception as e:
        print('Error adding client #' + str(clientID) + 'to list')

"""List all the clients currently connected"""
def list_connections(self):
    connections = []
    for i, conn in enumerate(self.all_connections):
        try:
            id = self.get_id(conn)
            if not(id):
                raise ValueError('Client not responding' )
            else:
                connections.append(id)
        except Exception as e:
            print("Client #" + str(i) + " is not responding and being deleted from lists, error: " + str(e))
            del self.all_clients[i]
            del self.all_addresses[i]
            del self.all_connections[i]
            continue
        results += str(id) + '   ' + str(self.all_addresses[i][0]) + '   ' + str(self.all_addresses[i][1]) + '   ' + str(self.all_addresses[i][2]) + '\n'
    if not results:
        print("No connections")
    else:
        print ('-----Clients-----' + '\n' + results)

"""Check regularly if all clients are connected. If not, tell which ones are not"""
def check_connections(self, numberOfClients):
    client = 1
    workingClients = []
    defectClients = []
    for i in self.all_connections:
        id = int(self.get_id(i))
        workingClients.append(id)
    while client <= numberOfClients:
        if client not in workingClients:
            defectClients.append(client)
        client +=1
    print(defectClients)

"""Get a target by its ID e.g. raspiPI #1, #2, #3 etc.."""
def get_target(self, cmd):
    target = cmd.split(' ')[-1]
    try:
        target = int(target)
    except exception as e:
        print('Client index should be an integer')
        return None, None
    try:
        index = self.all_clients.index(target)
        conn = self.all_connections[index]
    except IndexError:
```

```
            print('Not a valid selection')
            return None, None
        print("You are now connected to PI #" + str(self.all_clients[index]) + ', PI address: ' + str(self.all_addresses[index][0]))
        return index, conn

    """Connect with a remote target client(raspPI)"""
    def send_target_commands(self, target, conn):
        while True:
            try:
                cmd = input('Command> ')
                if cmd == 'id':
                    conn.send(cmd.encode())
                    client_response = conn.recv(80).decode()
                    print(client_response)
                elif cmd == 'preview':
                    conn.send(str.encode(cmd))
                elif cmd == 'get':
                    if(self.get_ack(conn)):
                        self.transfer_pictures(str(target))
                    else:
                        print("Client #" + target + "has not taken a picture yet")
                elif cmd == 'update':
                    self.send_update(str(target))
                    break
                elif cmd == 'reset':
                    self.reset_clients(target)
                    break
                elif cmd == 'shutdown':
                    conn.send(str.encode(cmd))
                    self.all_clients.remove(target)
                    self.all_addresses.remove(target)
                    self.all_connections.remove(target)
                    break
                elif cmd == 'reboot':
                    conn.send(str.encode(cmd))
                    self.all_clients.remove(target)
                    self.all_addresses.remove(target)
                    self.all_connections.remove(target)
                    break
                elif cmd == 'quit':
                    break
                else:
                    print("Command not recognized")
            except Exception as e:
                print("Connection was lost %s" %str(e))
                break
        return

    """Get the LAN address of the device running the program"""
    def get_address(self):
        IP = socket.gethostbyname(socket.gethostname()) #Get the IP address of the machine where the python interpreter is currently executing
        return IP

    """For receiving pictures the recvall method is not suited, large amounts of data is most effectively received in small packets"""
    """Send a request to each connection asking for the most recent picture taken. Then store the picture in a new folder at a given directory"""
    def get_pictures(self, conn, clientID):
        tries = 0
        conn.settimeout(4)
        while tries < 3:
            try:
                conn.sendall('get'.encode())
                bs = conn.recv(2048)
                (length,) = struct.unpack('>Q', bs)
                data = b''
                while len(data) < length:
                    to_read = length - len(data)
                    data += conn.recv(4096 if to_read > 4096 else to_read)
                print('Received picture from client #' + str(clientID))
                #Write data to the newly created folder
                with open((self.picturePath + self.pictureName + '/' + self.pictureName + str(clientID) + '.jpg'), 'wb') as fp:
                    fp.write(data)
                    fp.close()
                tries = 3
            except socket.timeout as e:
                tries += 1
                print("Client #" + str(clientID) + ' took too long responding, try #' + str(tries) + '. Error: ' + str(e))
            except Exception as e:
                tries += 1
                print('Error receiving data from client #' + str(clientID) + ', try #' + str(tries) + ". Error: " + str(e))
                time.sleep(1)
        conn.settimeout(None)

    """Method for getting pictures from clients, and getting a picture from a specific client if necessary"""
    def transfer_pictures(self, client=""):
        if not client:
            for i, conn in enumerate(self.all_connections):
                clientID = self.get_id(conn)
                self.get_pictures(conn, clientID)
        else:
            target = self.all_connections[int(client)]
            clientID = self.all_clients[int(client)]
            self.get_pictures(target, clientID)

    """Method for creating a new folder where the pictures reside"""
    def create_folder(self):
        pathExists = True
        try:
            newpath = self.picturePath + self.pictureName
            while pathExists:
                if os.path.exists(newpath):
                    self.pictureName = input('Name already exists, try again > ')
                    newpath = self.picturePath + self.pictureName
                else:
                    pathExists = False
            os.makedirs(newpath)
        except Exception as e:
            print("Could not create folder: " + str(e))

    """Method for pinging specific client to see if it's connected and working"""
    def ping_client(self, client=""):
        if not client:
            for i, target in enumerate(self.all_connections):
                self.ping_target(target, i)
        else:
            target = self.all_connections[int(client)]
            self.ping_target(target, int(client))

    """Helper method for ping_client method"""
    def ping_target(self, client, number):
        client.settimeout(1)
        try:
            id = self.get_id(client)
            if id:
```

```
            print("Client #" + id + " is working.")
        else:
            print("else")
    except socket.timeout as e:
        print("Client #" + str(number) + " did not respond in time. Timeout error: " + str(e))
        del self.all_clients[number]
        del self.all_addresses[number]
        del self.all_connections[number]
    except Exception as e:
        print("error try statement: " + str(e))
        del self.all_clients[number]
        del self.all_addresses[number]
        del self.all_connections[number]

"""Method for sending the update to every, or an individual client"""
def send_update(self, client=""):
    if not client:
        for i,target in enumerate(self.all_connections):
            id = self.get_id(target)
            self.update_client(target, id)
            print("Client #" + str(id) + " is now updated")
    elif(self.all_connections[int(client)]) is not None:
        target = self.all_connections[int(client)]
        id = self.get_id(target)
        self.update_client(target, id)
        print("Client #" + str(id) + " is now updated")
    else:
            print("Not a valid selection")

"""Helper method for fetching the client file from server"""
def get_update(self):
    update = None
    try:
        with open (self.clientPath + 'client' + '.py', 'rb') as fp:
            update = fp.read()
            fp.close()
    except Exception as e:
        print("Could not fetch update: " + str(e))
    finally:
        return update

"""Transfer the file to client"""
def update_client(self, client, id):
    try:
        update = self.get_update()
        client.sendall('update'.encode())
        time.sleep(0.5)
        assert(len(update))
        length = struct.pack('>Q', len(update))
        client.sendall(length)
        client.sendall(update)
    except AssertionError as e:
        print("Could not get update: " + str(e))
    except Exception as e:
        print("Could not send update: " + str(e))

"""Compare the updated client version with the version on each client to know if an update is needed"""
def compare_version(self, client):
    try:
        version = 'update ' + self.get_clientVersion()
        client.sendall(version.encode())
        return self.get_ack(client)
    except Exception as e:
        print("Something went wrong comparing client versions: " + str(e))

"""Helper method for asking if something has been completed"""
def get_ack(self, conn):
    conn.settimeout(5)
    buf = ''
    try:
        while ' ' not in buf:
            buf += conn.recv(8).decode()
        num = int(buf)
        if num == 1:
            return True
        elif num == 0:
            return False
        else:
            print("lol")
            return
    except socket.timeout as e:
        print("Client  #"    + "took too long responding")
    conn.settimeout(None)

"""Helper method for confirming something has been completed"""
def send_ack(self, conn, ack):
    sep = ' '
    if ack == 1 or ack == 0:
        ack = str(ack) + sep
        conn.sendall(ack.encode())
    else:
        print("Not valid ack input")

"""Get the current version of client software"""
def get_clientVersion(self):
    self.set_clientVersion()
    print(self.clientVersion)
    return self.clientVersion

"""Get the current version of the program and assign it to the client version variable"""
def set_clientVersion(self):
    newestVersion = ""
    try:
        c = re.compile('client')
        p = re.compile(r'\d+.?\d*')
        filesInDirectory = os.listdir(self.clientPath)
        for files in filesInDirectory:
            m = c.match(files)
            if m:
                version = p.search(files)
                if version:
                    versionString = version.group()
                    if not newestVersion:
                        newestVersion = versionString
                    elif(float(versionString)>float(newestVersion)):
                        newestVersion = versionString
                    else:
                        pass
            else:
                continue
        self.clientVersion = newestVersion
    except Exception as e:
        print("Couldn't find client file, error: " + str(e))
```

```
    """Method for setting where on the server pictures are stored"""
    def set_picturePath(self):
        pass

    """Set the number of clients to use"""
    def set_number_of_clients(self, number):
        self.numberOfClients = number

    """"Method to get id of client"""
    def get_id(self, client):
        client.settimeout(2)
        try:
            client.sendall(b'id')
            id = client.recv(8).decode()
        except Exception as e:
            print("Something went wrong, " + str(e))
        client.settimeout(None)
        return id

    def print_length(self):
        print(len(self.all_connections))
        print(len(self.all_addresses))
        print(len(self.all_clients))

    """Continous broadcasting of connection info to the network for clients to connect"""
    def broadcast_address(self):
        freqLim = 15
        currentFreq = 1
        while True:
            if(currentFreq < freqLim):
                self.mcast_message("address")
                # self.check_connections(self.numberOfClients)
                time.sleep(currentFreq)
                currentFreq += 2
            else:
                self.mcast_message("address")
                # self.check_connections(self.numberOfClients)
                time.sleep(currentFreq)

    """Interactive prompt for sending commands"""
    def start_turtle(self):
        while True:
            cmd = input('Command> ')
            if cmd == 'length':
                self.print_length()
            elif cmd == 'shutdown':
                self.reset_clients()
                self.mcast_message(cmd)
            elif cmd == 'reboot':
                self.reset_clients()
                self.mcast_message(cmd)
            elif cmd == 'help':
                self.print_help()
            elif cmd == 'llst':
                self.list_connections()
            elif cmd == 'address':
                self.mcast_message(cmd)
            elif cmd == 'reset':
                self.reset_clients()
            elif cmd == 'picture':
                self.mcast_message(cmd)
                time.sleep(0.5)
                self.transfer_pictures()
            elif cmd == 'get':
                self.transfer_pictures()
            elif cmd == 'ping':
                try:
                    client = input("Input client, or empty to ping every client: ")
                    if not client:
                        self.ping_client()
                    elif(self.all_connections[int(client)-1] is not None):
                        self.ping_client(client)
                    else:
                        print("Not valid input")
                except Exception as e:
                    print("Error: " + str(e))
            elif cmd == 'settings':
                settings = input('camera settings: ')
                self.cameraSettings = settings
            elif cmd == 'update':
                client = input("Input client, or empty to update every client: ")
                self.send_update(client)
            elif 'select' in cmd:
                target, conn = self.get_target(cmd)
                if conn is not None:
                    self.send_target_commands(target, conn)
            elif cmd == 'quit':
                self.reset_clients()
                time.sleep(0.3)
                i = 0
                while i < NUMBER_OF_THREADS:
                    queue.task_done()
                    i += 1
                print('Server shutdown')
                break
            elif cmd == '':
                pass
            else:
                print("Command not recognized")

def create_workers():
    """ Create worker threads (will die when main exits) """
    server = Server()
    server.register_signal_handler()
    for _ in range(NUMBER_OF_THREADS):
        t = threading.Thread(target=work, args=(server,))
        t.daemon = True
        t.start()
    return


def work(server):
    """ Do the next job in the queue (thread for handling connections, another for sending commands)
    :param server:
    """
    while True:
        x = queue.get()
        if x == 1:
            server.create_mcast_socket()
            server.create_tcp_socket()
            server.bind_tcp_socket()
            server.accept_connections()
```

```
        if x == 2:
            server.start_turtle()
        if x == 3:
            server.broadcast_address()
        queue.task_done()
    return

def create_jobs():
    """ Each list item is a new job """
    for x in JOB_NUMBER:
        queue.put(x)
    queue.join()
    return

def main():
    create_workers()
    create_jobs()


if __name__ == '__main__':
    main()
```