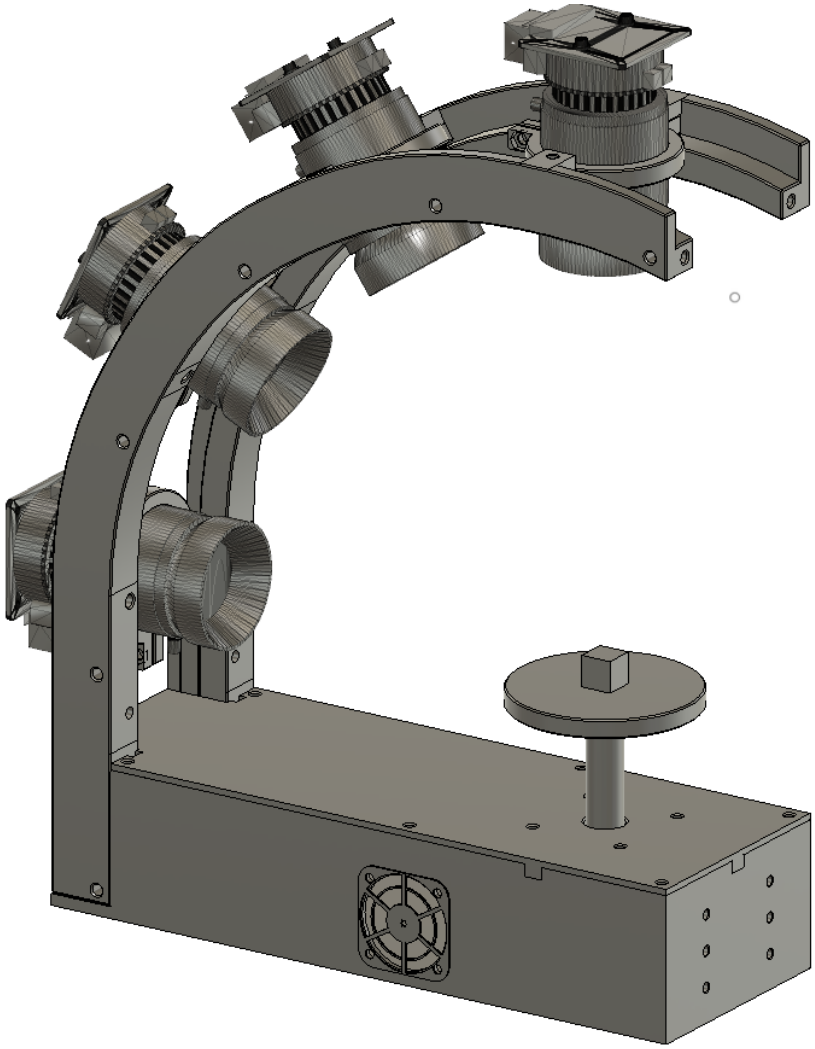
CHALMERS



Design och konstruktion av en 3D-scanner för små objekt

Examensarbete inom högskoleprogrammet Mekatronikingenjör

Remus Damberg

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2026

www.chalmers.se

Examensarbete 2026

Design och konstruktion av en 3D-scanner för små objekt

Remus Damberg

CHALMERS

Institutionen för Elektroteknik

Chalmers Tekniska Högskola

Göteborg 2026

Design och konstruktion av en 3D-scanner för små objekt

Remus Damberg

Handledare: Andreas Ljung, Wiretronic AB
Examinator: Tomas McKelvey, Elektroteknik

Examensarbete 2026
Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: CAD-modell föreställande det färdiga systemet.

Skriven i LaTeX
Göteborg 2026

Sammanfattning

Detta arbete beskriver konstruktionen av ett skanningssystem för små objekt, utvecklat på uppdrag av Wiretronic AB. Systemet är ämnat att komplettera Wiretronic's existerande lösningar genom att erbjuda högdetaljerad fotografering av små objekt i storleksordningen 0,5-20mm i syfte att användas för 3D-modellering och träning av AI-modeller. Systemet består av en kvartcirkelformad båge med fyra kameramoduler, ett stegmotordrivet rotationsbord, LED-belysning med manuellt justerbar ljusstyrka och färgtemperatur, samt en Raspberry Pi 4B som beräkningsenhet. Tre avbildningslösningar utvärderades. Raspberry Pi HQ-kameran med teleobjektiv valdes då den optiska förstorningen möjliggjorde effektivt utnyttjande av sensorns upplösning och gav ett tillräckligt skärpedjup för de aktuella objekten. Valet av motorsystem gjordes på teoretisk grund, där stegmotorns positioneringsnoggrannhet utan behov av extern återkoppling bedömdes vara det mest lämpade alternativet. Detta bekräftades vid praktisk testning, där stegmotorn uppvisade mycket god noggrannhet och repeterbarhet. LED-strips med hög färgåtergivning och justerbar färgtemperatur valdes som belysningslösning då dessa gav ett jämnare ljus med färre skuggproblem jämfört med punktformade LED-moduler. Det färdiga systemet fungerar väl och uppfyller de ställda kraven. Identifierade förbättringsmöjligheter inkluderar en mer precis tillverkningsmetod för de mekaniska delarna, då 3D-printing med kommersiell utrustning introducerar geometriska toleransavvikelser som ger en viss gungning i rotationsbordet som blir märkbar först vid inzoomning på objektet. Fast montering av kameramodulerna via deras kretskortsfästen skulle ge en mer robust långsiktig lösning jämfört med det klämbaserade monteringssystemet. Belysningsstyrningen som i nuvarande utförande sker manuellt via potentiometrar skulle med fördel kunna regleras via Raspberry Pi för smidigare kontroll och reproducerbarhet.

Nyckelord: CAD, 3D-printing, FDM, Raspberry Pi, datainsamling, träningsdata.

Abstract

This thesis describes the design and construction of a scanning system for small objects, developed on behalf of Wiretronic AB. The system is intended to complement Wiretronic's existing scanning equipment by enabling high-detail photography of components in the range of 0.5-20mm, such as terminal pins and connector housings, for use as training data for AI models and as input for 3D-modelling. The system consists of a quarter-circle arc with four camera modules, a stepper motor based rotating platform, LED lighting with adjustable brightness and colour temperature, and a Raspberry Pi 4 as the computing unit. The mechanical structure is manufactured using FDM 3D-printing, and all electronics are housed either in the base of the system or mounted to the inside of the base lid. Three photography solutions were evaluated. The Raspberry Pi HQ camera with a telephoto lens was selected because the optical magnification allowed efficient use of the sensor's resolution and provided sufficient depth of field for the test objects. The choice of motor system was made on theoretical grounds, with the stepper motors positioning accuracy deemed the most suitable option. This was confirmed during practical testing where the stepper motor demonstrated excellent accuracy and repeatability. LED strips with high colour rendering index and adjustable colour temperature were selected as the lighting solution, as they produced more even illumination with fewer shadows compared to LED modules. The completed system performs well and meets the specified requirements. Identified areas for improvement include a more precise manufacturing method for the parts, as 3D-printing introduces geometric tolerances that result in minor wobble in the rotating platform. Direct mounting of the camera modules via their PCB mounting holes would provide a more robust solution compared to the current clamp-based system. The lighting control, currently implemented via manual potentiometers, could be integrated through the Raspberry Pi for increased control and repeatability between sessions.

Keywords: CAD, 3D-printing, FDM, Raspberry Pi, data collection, training data.

Förord

Detta examensarbete, motsvarande 15HP, har utförts vid institutionen för Elektroteknik som en del av en kandidatexamen i Mekatronik på 180HP vid Chalmers tekniska högskola, på uppdrag av Wiretronic AB.

Jag vill rikta ett stort tack till Wiretronic för möjligheten att genomföra detta arbete, för den värdefulla erfarenhet det har gett mig, och för en trevlig rundtur på deras fina huvudkontor. Ett särskilt tack går till min handledare på Wiretronic, Andreas Ljung, för stöd och vägledning under arbetets gång. Jag vill också tacka min examinator vid Chalmers, Tomas McKelvey, för värdefulla synpunkter och återkoppling.

Remus Damberg, Göteborg, Juni 2026

Innehåll

Terminologi/Förkortningar **xv**

Figurer **xvii**

Tabeller **xix**

1 Inledning **1**

1.1 Bakgrund 1

1.2 Syfte 2

1.3 Avgränsningar 2

1.4 Precisering av frågeställningen 2

2 Teoretisk och teknisk bakgrund **5**

2.1 Likströmsmotor 5

2.2 Stegmotor och stegmotordrivare 6

2.3 LED-belysning 7

2.4 Kameramodulerna 7

2.4.1 LI-IMX219-MIPI-FF-NANO-H136 7

2.4.2 Raspberry Pi HQ med objektiv 8

2.4.3 Digitalt mikroskop 8

2.4.4 Arducam B0483 8

2.5 4-MUX adapterkort 8

2.6 Arduino 8

2.7 Raspberry Pi 9

2.8 3D-printing 9

2.9 Python 9

2.10 PWM 9
2.11 Transistor, potentiometer och kretskort 9
2.12 Autodesk Fusion 10
2.13 DC-DC buck converter 10

3 Metod **11**
3.1 Arbetsprocess 11
3.2 Utvärdering av kameramoduler 11
3.3 Utvärdering av belysning 12
3.4 Verifiering av rotationssystem 12
3.5 Mjukvara 12

4 Hårdvaruval och Konstruktion **13**
4.1 Första prototypen 13
4.1.1 Mekanisk konstruktion 14
4.1.2 Montering av kameror och LED-moduler 15
4.1.3 Test av hårdvara med prototypen 17
4.2 Slutlig design 20
4.2.1 Mekanisk konstruktion 20
4.2.2 Val av motorsystem 21
4.2.3 Elektrisk konstruktion 23
4.2.3.1 Stegmotor 23
4.2.3.2 Belysningssystem 24
4.3 Programmering 25
4.3.1 Arduino 25
4.3.2 Raspberry Pi 25
4.4 Problem och lösningar 26
4.4.1 Störningar och kortslutning 26

5 Resultat **29**
5.1 Avbildning 29
5.2 Rotationssystem 30
5.3 Belysning 30
5.4 Det färdiga systemet 31

6 Diskussion **35**

6.1 Val av kamerasystem och avbildning 35

6.2 MUX-adapterns begränsning 35

6.3 Val av motorsystem 36

6.4 Belysningsstyrning 36

6.5 Framtida arbete 36

6.6 Arbetsprocess 37

6.7 Etik och hållbarhet 37

7 Slutsats **39**

Referenser **41**

A Källkod **I**

A.1 Arduinokod I

A.2 Pythonkod II

Terminologi/Förkortningar

AI	Artificial Intelligence
CAD	Computer-Aided Design
CRI	Color Rendering Index
CSI	Camera Serial Interface
DC	Direct Current - likström
FDM	Fused Deposition Modeling - en metod för 3D-printing
GPIO	General Purpose Input/Output
HMI	Human Machine Interface
IDE	Integrated Development Environment
LED	Light Emitting Diode
MP	Megapixel
MUX	Multiplexer (Multi Camera Adapter Board)
NPN	Negative-Positive-Negative - en typ av bipolär transistor
PCB	Printed Circuit Board - kretskort
PWM	Pulse Width Modulation
RPI	Raspberry Pi
SSH	Secure Shell - Nätverksprotokoll för krypterad kommunikation

Buck converter	En switchad spänningsregulator som omvandlar en högre inspänning till en lägre utspänning med hög verkningsgrad.
Encoder	En sensor som mäter en axels rotationsposition och skickar denna information till ett styrsystem.
Heat-set insert	En gängad metallinsättning som värmepressas in i plast för att ge ett hållbart gängfäste.
Low-side switching	En kopplingsteknik där en transistor bryter kretsen på jordsidan av lasten.
Multiplexer	En komponent som kopplar samman flera ingångssignaler till en gemensam utgång, en i taget.
Skärpedjup	Det avståndsspann inom vilket ett objekt avbildas med acceptabel skärpa av en kamera.
Stegmotor	En elektrisk motor som rör sig i diskreta steg och möjliggör exakt positionering utan extern återkoppling.
Teleobjektiv	Ett objektiv med lång brännvidd som ger optisk förstoring och snäv bildvinkel.

Figurer

2.1 Intern vy av elektromagneter och magnetiserad kärna hos en stegmotor. Genom aktivering av elektromagneterna i en bestämd sekvens roterar rotorn i önskad riktning och hastighet. 6

4.1 CAD-modell av första prototypen med de modulära testplattformarna. . . 14
4.2 Sidovy av prototypen med kamerornas siktlinjer och hålens mittlinjer inritade. Skärningspunkterna användes för att välja ut optimala testpositioner. 15
4.3 Detaljvy av monteringssystemet för kameror och LED-moduler. Modulernas hållare skruvas fast i gängade metallinsatser monterade i bågens infästningspunkter. 16
4.4 Jämförelse av bilder från mikroskopkameran. Till höger syns en bild tagen i 720p och till vänster en tagen i "4k". Tydliga tecken på digital bildbehandling syns på vänster sida i form av bland annat vertikala linjer och brist på detaljer som finns i högra bilden. 17
4.5 Exempelbild på en fotad terminalkontakt tagen med Arducam 64MP. Notera hur den bakre och främre delen av kontakten ej är i fokus. 18
4.6 Exempelbild på en fotad terminalkontakt tagen med Raspberry Pi HQ-kamera. 19
4.7 Den slutliga designen av skanningssystemet med kameramoduler monterade. 20
4.8 Vy som visar montering av stegmotor, RPI och platta för stegmotordrivaren på insidan av locket. Syns även hål för muttrar för fäste av armarna, samt hål för potentiometrar, knappar och DC-plugg. 21
4.9 Kopplingsschema för stegmotorsystemet. Raspberry Pi 4B styr TMC2209 stegmotordrivaren via GPIO-pinnarna 21 (STEP), 20 (DIR) och 16 (ENABLE). 23

4.10 Kopplingsschema för belysningssystemet. Tre potentiometrar kopplade till Arduino Nano styr ljusstyrkan hos de två LED-kanalerna via NPN-transistorer. Nano försörjs med 5V från en DC-DC buck converter. 24
4.11 Närbild på den felkonstruerade kontaktytan på CSI-kabeln, där ledare sträckte sig över flera pins och orsakade kortslutning. Notera de svarta märkena mellan pinsen där plasten smält. 26

5.1 Exempelbild på en skannad terminalkontakt tagen med systemets Raspberry Pi HQ-kamera. Terminalkontakten är ca 13*3*3 mm stor 29
5.2 Det färdiga skanningssystemet. CSI-kablage till kameramodulerna är ej installerat på bilden. Systemet är ca 270*80*220 mm stort. 31
5.3 Vy av elektroniklådans insida, med potentiometrar monterade till höger samt fläkt och Arduino-kretskort monterat till vänster. Det lediga utrymmet ockuperas av Raspberry Pi, stegmotordrivare och stegmotor då locket är monterat. 33
5.4 Skärmdump som visar systemets användargränssnitt med en live preview till vänster, och menyalternativ till höger. 34

Tabeller

2.1 Jämförelse av kameramodulernas relevanta specifikationer. Med bildvinkel avses horisontell bildvinkel. 7

1

Inledning

1.1 Bakgrund

Wiretronic utvecklar och tillverkar primärt kabelmontage inom bil och försvarsindustrin, samt tillhandahåller teknisk support, reparationsinstruktioner och reservdelar.

I takt med att produkternas komplexitet ökar blir servicearbetet svårare, delvis till följd av svårigheter med att identifiera vilket artikelnummer en specifik komponent har när problem uppstår i fält. För att lösa detta har Wiretronic tränat en AI-modell på bilder av komponenter, vilket gör det möjligt att identifiera en komponent direkt från ett foto taget med en vanlig smartphone.

Wiretronic har sedan tidigare två skanningssystem bestående av Nikon DSLR-kameror arrangerade i en kvartsbåge över ett roterande objekt. Kamerorna tar bilder medan objektet roterar, vilket genererar träningsdata för AI-modellen. De befintliga systemen är utformade för större objekt i storleksordningen 20cm och större.

Wiretronic har nu ett behov av liknande system för betydligt mindre komponenter för att kunna skapa 3D-modeller och träningsdata för artiklar så små som enskilda stiftkontakter. De befintliga systemen är inte lämpade för avbildningar i denna skala, och ett nytt system behöver därför utvecklas.

1.2 Syfte

Syftet med detta arbete är att konstruera ett skanningssystem kapabelt att ta högdetaljerade bilder av små objekt i storleksordningen 0,5-20mm från flera vinklar, för att producera ett bildunderlag som täcker objektet från 360 grader. Systemet ska kunna användas som datainsamlingsplattform för AI-träning och 3D-modellering av Wiretronics minsta komponenter såsom kontakter och kontakthus, alternativt som bas för fortsatt utveckling.

1.3 Avgränsningar

Bildbehandling och analys av de insamlade bilderna för användning i modellering och AI-träning ligger utanför arbetets omfattning och kommer inte att hanteras. Utvärderingen av bildernas kvalitet görs enbart utifrån egen subjektiv bedömning i samråd med handledare på Wiretronic.

Systemet är ämnat att skanna små objekt i storleksordningen 0,5-20mm. Hänsyn har inte tagits till större objekt eller möjlighet av anpassning av systemet för att hantera större objekt.

Endast de kameramoduler, motorsystem, belysningskällor och beräkningsenheter som varit tillgängliga har utvärderats. Hänsyn har inte tagits till övriga lösningar på marknaden.

Systemet utgör en funktionell prototyp och är ej optimerat för produktionsmiljö eller skalbarhet.

1.4 Precisering av frågeställningen

Hur ska ett skanningssystem för små objekt utformas för att uppnå hög bildkvalitet, repeterbar objektpositionering och god belysning i ett kompakt format?

Avbildning

Vilken av de tillgängliga kameramodulerna ger högst bildskärpa och detaljnivå vid fotografering av små objekt?

Rotationssystem

Vilket av de tillgängliga motorsystemen lämpar sig bäst för precis och repeterbar rotation av plattan?

Belysning

Vilken av de tillgängliga belysningslösningarna ger bäst förutsättningar för det valda kamerasystemet? Centralt monterade LED-moduler eller sidmonterade LED-strips? Hur kan ljusstyrka och färgtemperatur enkelt justeras för att optimera bildkvaliteten?

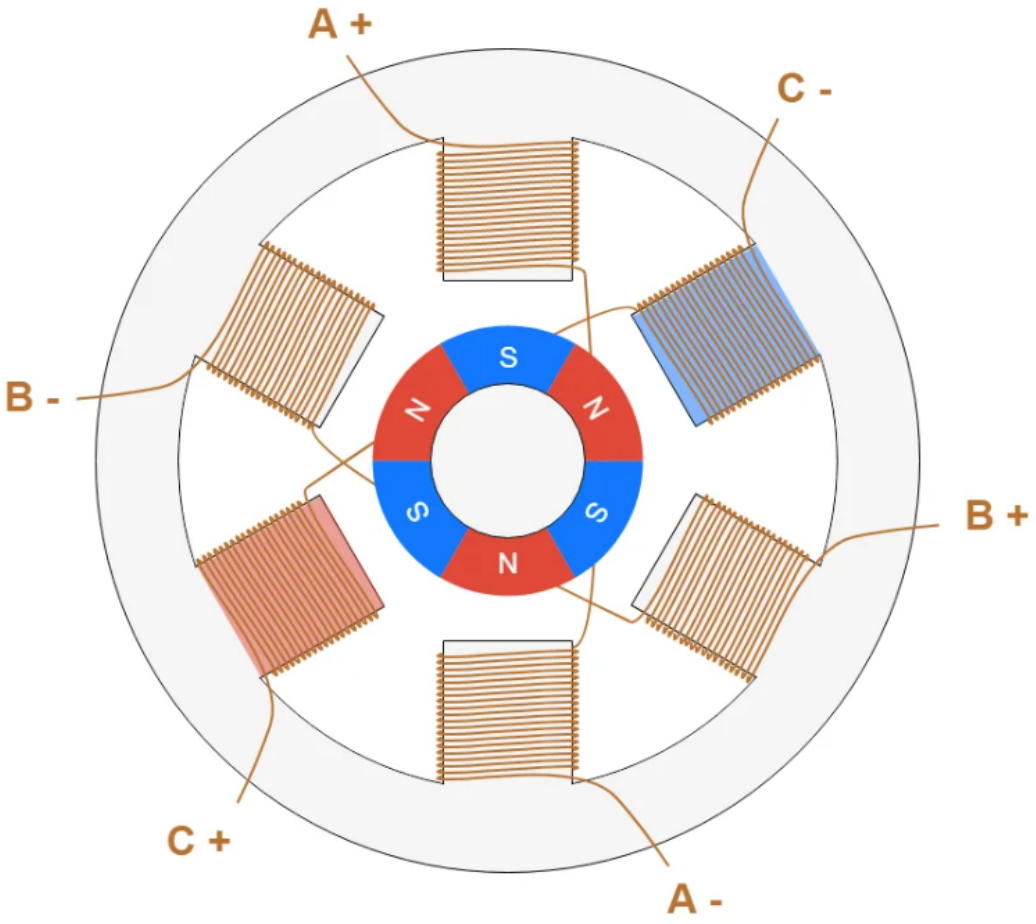
2

Teoretisk och teknisk bakgrund

2.1 Likströmsmotor

En likströmsmotor (DC-motor) omvandlar elektrisk energi till rotationsrörelse genom samspelet mellan ett magnetfält och en strömförande ledare. [1]. Motorns varvtal regleras genom att variera spänningen eller strömmen. DC-motorer saknar inbyggd positionering och kräver därför ett externt system för att bestämma axelns position, till exempel en encoder.

2.2 Stegmotor och stegmotordrivare



Figur 2.1: Intern vy av elektromagneter och magnetiserad kärna hos en stegmotor. Genom aktivering av elektromagneterna i en bestämd sekvens roterar rotorn i önskad riktning och hastighet.

En stegmotor är en typ av elektrisk motor som rör sig i diskreta steg istället för kontinuerligt likt en DC-motor. se figur 2.1 Varje steg motsvarar en fast vinkel, i normalfallet 1,8 grader per steg vilket motsvarar 200 steg per full rotation. Det är även möjligt att låsa rotorns position för att förhindra oönskade rörelser. [2] Detta gör stegmotorn särskilt lämpad för applikationer där exakt och repeterbar positionering behövs, utan behov av extern encoder.

Stegmotorn drivs av en stegmotordrivare, som omvandlar styrsignaler från en mikrokontroller till de strömpulser som behövs för att rotera motorn. Det finns flera olika stegmotordrivare, men i detta arbete kommer en av modellen TMC2209 att användas på grund av dess låga ljudnivå.

2.3 LED-belysning

LED (Light Emitting Diode) är en halvledarkomponent som avger ljus när ström leds genom den. LED-belysning är energieffektiv, har lång livslängd och finns i ett brett spektrum av färgtemperaturer från varmt vitt till kallt vitt. Färgtemperaturen påverkar hur ett objekt upplevs i bild och kan ha stor inverkan på bildkvaliteten vid fotografering.

Den LED-strip som använts utgörs av typen 2835, med 224 LEDs per meter där varannan LED har en färgtemperatur på 2800k, och varannan en färgtemperatur på 6500k. Detta ger möjligheten att mycket kontrollerat justera belysningens färgtemperatur. [3] Maximala ljusnivån anges av tillverkaren till 2000 Lumen per meter, och effekten till 18W per meter. Färgåtergivningen uppges till >90 CRI (Color Rendering Index).

Den LED-modul som testats är av märket "Hiltand" och utgörs av en enstaka LED på 1W monterad på en aluminiumplatta. Modulen är specificerad till 200 Lumen med en färgtemperatur på 3000-3200k. [4] Ingen information om färgåtergivning ges.

2.4 Kameramodulerna

Fyra olika kameramodeller utvärderades för att identifiera den som bäst lämpar sig för avbildning av testobjekten. Tabell 2.1 sammanfattar de viktigaste specifikationerna för respektive modul.

Tabell 2.1: Jämförelse av kameramodulernas relevanta specifikationer. Med bildvinkel avses horisontell bildvinkel.

Modell	Sensor	Upplösning	Fokus	Bildvinkel
LI-IMX219	Sony IMX219	8 MP	Fast, 30 cm-inf	136°
RPi HQ	Sony IMX477R	12,3 MP	Manuellt	25°
Digitalt mikroskop	Okänd	8 MP (påstådd)	Manuellt	Mycket Smal
Arducam B0483	OV64A40	64 MP	Auto/manuellt	68°

2.4.1 LI-IMX219-MIPI-FF-NANO-H136

Baserad på Sony IMX219 med 8MP upplösning, 136 graders vidvinkel och ett fixerat fokus på 30 cm med ett skärpedjup som beskrivs som oändligt. [5]. Benämns fortsättningsvis som IMX219.

2.4.2 Raspberry Pi HQ med objektiv

Baserad på Sony IMX477R med 12,3MP upplösning, bakgrundsbelyst sensor och fäste för C-/CS-objektiv där ett teleobjektiv av modell PT3611614M10MP monteras. Modulen får då en snäv bildvinkel på cirka 25 grader, manuellt fokus och variabel bländare. [6] [7]

2.4.3 Digitalt mikroskop

Digitalt mikroskop från Cainda, baserad på en okänd sensor, med en påstådd upplösning på 8MP. [8]

Ingen bildvinkel anges, men upplevs som mycket smal. Mikroskopet har manuellt reglerad fokus och digital zoom samt inbyggd justerbar ringbelysning runt objektivet.

2.4.4 Arducam B0483

Modulen är baserad på OmniVision OV64A40-sensorn med en upplösning på 64 MP och ett motordrivet fokus, vilket stödjer både automatisk och manuell kontroll digitalt. [9]

2.5 4-MUX adapterkort

Adapterkort från Arducam av modell B012001 som möjliggör anslutning av fyra kameramoduler till en Raspberry Pi, som i normalfallet enbart kan hantera en kamera. De anslutna kamerorna kan endast användas en i taget, vilket innebär att simultan bildtagning med alla fyra anslutna kamerasensorer inte är möjligt. [10]

2.6 Arduino

Arduino är en plattform för inbyggda system som kombinerar mikrokontrollerkort med den tillhörande utvecklingsmiljö Arduino IDE. Programmeringsspråket är baserat på C/C++ och miljön tillhandahåller ett enkelt gränssnitt för att skriva, kompilera och ladda upp kod till kortet.

Arduino Nano är den minsta modellen och är endast 45x18mm stor, vilket gör den lämplig för integration i andra projekt. [11] Systemet kan läsa av både analoga och digitala

signaler, samt styra utgångar och utföra enklare kod och har även tillgång till hårdvarubaserad PWM-reglering. Detta gör den lämplig för att hantera enklare styruppgifter som inte kräver ett fullständigt operativsystem.

2.7 Raspberry Pi

Raspberry Pi är en enkortsbaserad dator utvecklad av Raspberry Pi Foundation. Den kör ett fullständigt Linux-baserat operativsystem och är utrustad med bland annat USB-portar, GPIO-pinnar och ett CSI-gränssnitt för en kamera. I detta projekt används Raspberry Pi 4 Model B, som har en quad-core ARM Cortex-A72-processor och 8GB RAM. [12]

2.8 3D-printing

3D-printing, eller additiv tillverkning, är en process där objekt byggs lager för lager utifrån en digital 3D-modell. Den vanligaste metoden för detta görs med plast genom FDM (Fused Deposition Modeling), där ett plastfilament smälts och extruderas på ett förutbestämt sätt för att forma objektet. [13] Det är också denna metod som används i projektet.

2.9 Python

Python är ett högnivåprogrammeringsspråk känt för sin läsbarhet och sitt stora ekosystem av färdiga bibliotek. Det är det primära programmeringsspråket för Raspberry Pi och används i stor utsträckning inom inbyggda system.

2.10 PWM

PWM (Pulse Width Modulation), eller pulsbreddsmodulering, är en teknik för att styra den effektiva spänningen eller strömmen till en last genom att snabbt slå av och på en digital signal med varierande pulsbredd. [14]

2.11 Transistor, potentiometer och kretskort

En transistor är en halvledarkomponent som kan användas antingen som en elektronisk strömbrytare eller som ett styrbart motstånd. Genom att applicera en liten styrsignal på

transistorns bas- eller gate-ingång kan ett betydligt större strömflöde regleras.

En potentiometer är ett variabelt motstånd vars värde justeras manuellt via ett vred.

Ett PCB (Printed Circuit Board), eller kretskort, är en platta av isolerande material med etsade ledningsbanor av koppar som förbinder elektroniska komponenter.

2.12 Autodesk Fusion

Autodesk Fusion är ett CAD-program (Computer Aided Design) som används i stor utsträckning inom produktdesign och konstruktion. Programmet används i detta projekt för att modellera de fysiska delarna i skanningssystemet, inför tillverkning med 3D-printer.

2.13 DC-DC buck converter

En DC-DC buck converter är en switchad spänningsregulator som med hög verkningsgrad omvandlar en högre inspänning till en lägre utspänning. Till skillnad från en linjär spänningsregulator, som omvandlar överskottsenergin till värme, så reglerar en buck converter utspänningen genom att snabbt slå av och på en transistor. [15]

3

Metod

3.1 Arbetsprocess

Arbetet genomfördes i tre faser. I den första fasen konstruerades en förenklad prototyp för att möjliggöra strukturerad utvärdering av kameramoduler och belysningslösningar. I den andra fasen fattades hårdvarubeslut utifrån resultaten från prototypfasen, varefter det slutliga systemet konstruerades, programmerades och monterades. I den tredje fasen verifierades systemet genom praktisk testning och justeringar.

3.2 Utvärdering av kameramoduler

Kameramodulerna utvärderades med en "U shape splice crimp"(Hädanefter kallad Terminalkontakt") och en komplett molex-kontakt som testobjekt. Terminalkontakten valdes då den representerar den minsta typ av komponenter systemet är avsett att avbilda samt ställer höga krav på upplösning, skärpedjup och hantering av reflektioner. Molex-kontakten valdes som ett kompletterande testobjekt som en större jämförelsepunkt, då systemet även är avsett att avbilda hela kontakthus.

Prototypens modulära plattformssystem användes för att placera testobjektet på de avstånd och höjder som identifierats via siktlinjeanalysen i CAD-modellen. Varje kameramodul fotograferade testobjekten från samma positioner under identiska ljusförhållanden.

De producerade bilderna bedömdes subjektivt utifrån följande kriterier:

- Detaljnivå och skärpa på testobjektets detaljer.
- Skärpedjup, huruvida hela objektet kunde hållas i fokus samtidigt.
- Effektivt utnyttjande av bildsensorn, hur stor andel av bilden som upptogs av objektet jämfört med omgivningen.

Bedömningen av kameramodulernas bildkvalité gjordes subjektivt i samråd med handledare på Wiretronic.

Efter val av kameramodul genomfördes ytterligare tester för att fastställa det optimala avståndet mellan objekt och kameramodul. Testobjekten placerades på varierande avstånd från kameran och fotograferades, varefter bilderna bedömdes utifrån balansen mellan skärpedjup, ljusinsläpp och detaljnivå. Det avstånd som gav den bästa sammantagna bildkvaliteten valdes som utgångspunkt för dimensionering av den slutliga bågen.

3.3 Utvärdering av belysning

Belysningen utvärderades med samma testobjekt och den utvalda kameramodulen för att isolera belysningens påverkan på bildkvaliteten. De två belysningslösningarna testades var för sig under annars identiska förhållanden.

Bedömningen gjordes subjektivt med fokus på:

- Jämnhet i belysningen över objektets yta
- Förekomst av skuggor och reflektioner
- Upplevd ljusnivå

Eftersom ingen kalibrerad luxmätare fanns tillgänglig baserades bedömningen av ljusnivå på visuell jämförelse mellan de två lösningarna.

3.4 Verifiering av rotationssystem

Stegmotorns förmåga att uppfylla kraven på positioneringsnoggrannhet och repeterbarhet verifierades genom praktisk testning. Motorn programmerades att rotera ett bestämt antal steg och återgå till ursprungsläget upprepade gånger, varefter eventuell positionsavvikelse observerades visuellt.

3.5 Mjukvara

CAD-konstruktioner genomfördes i Autodesk Fusion. Kopplingsscheman ritades i Fritzing. Programmering av Raspberry Pi gjordes med Python i Visual Studio Code, och programmering av Arduino Nano gjordes i Arduino IDE. AI-agenten "Claude Code"har använts för framställning, felsökning och modifiering av programkoden. [16]

4

Hårdvaruval och Konstruktion

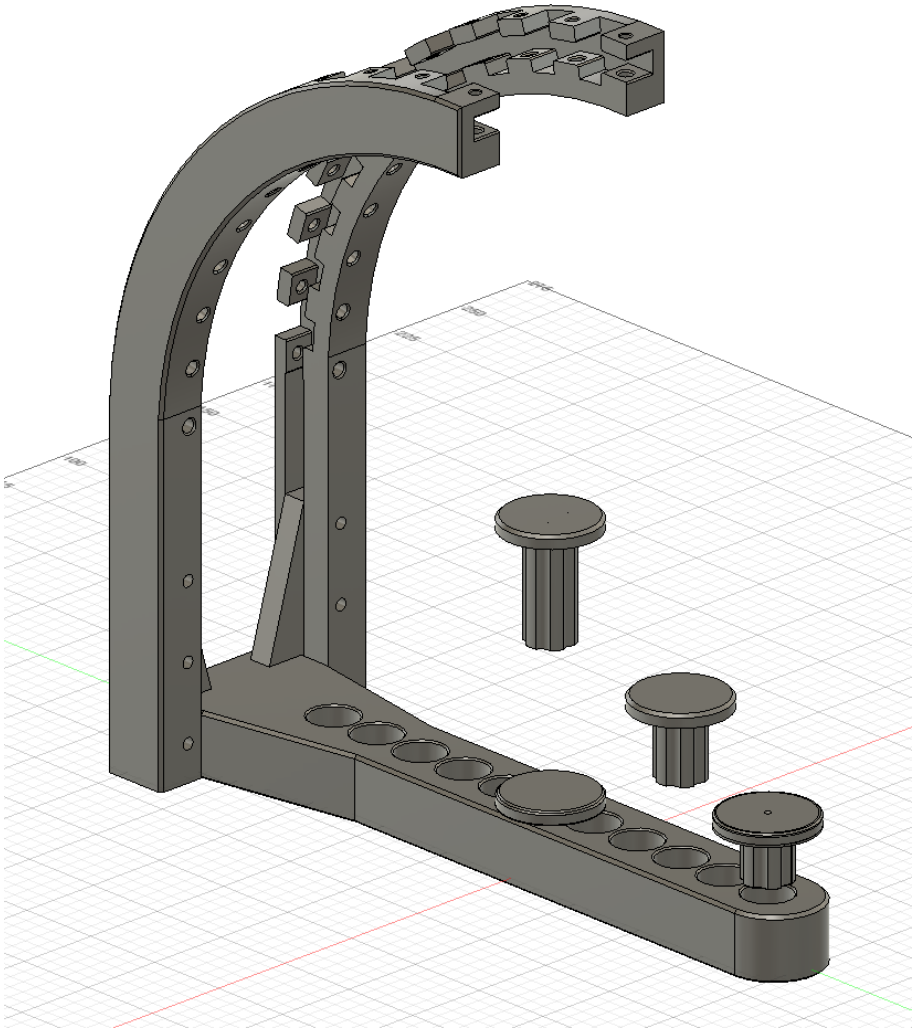
Vid utformning av ett skanningssystem av denna typ finns det flera möjliga konstruktionsprinciper att utgå ifrån. En potentiell lösning vore placering av ett antal kameror på ett stativ eller en arm, som i sin tur roterar runt ett stationärt objekt. Denna princip används för exempelvis bilar eller andra stora objekt som vore opraktiska att rotera, men ger ökad komplexitet och bedöms onödigt för detta arbete.

Den valda designen baseras på en kvartcirkelformad båge placerad över ett roterande bord. Kamerorna är monterade längs bågen och kan i kombination med det roterande bordet ge full visuell täckning av objektet, förutom botten.

4.1 Första prototypen

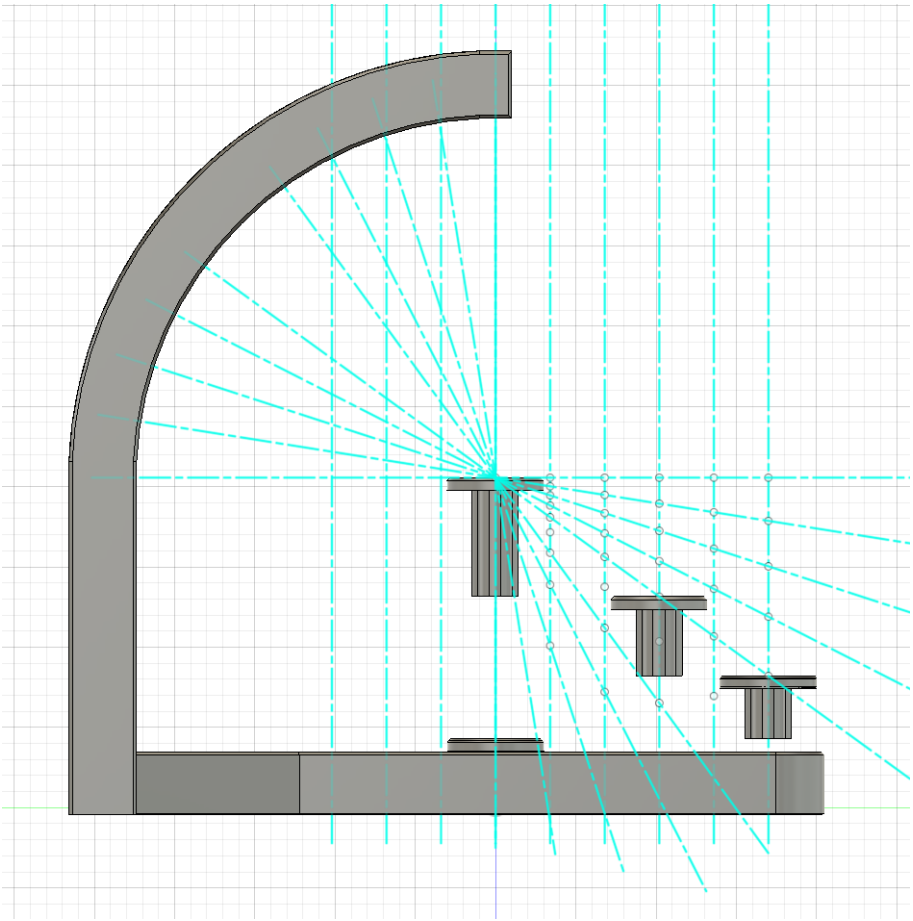
För att tidigt i processen kunna utvärdera kameramodulernas kapacitet, lämpligt objektavstånd och belysning konstruerades en förenklad första prototyp. Denna saknade elektroniklåda och roterande bord, och syftade uteslutande till att utgöra en justerbar testplattform för de optiska och belysningstekniska aspekterna av systemet.

4.1.1 Mekanisk konstruktion



Figur 4.1: CAD-modell av första prototypen med de modulära testplattformarna.

Prototypen bestod av en dubbelarmad båge med en plan bas, i vilken ett antal cirkulära hål placerats längs en linje från bågfoten och utåt. Se figur 4.1 I hålen monterades modulära plattformar med inbyggda magneter i topp och botten, vilket möjliggjorde att plattformarna kunde staplas till valfri höjd. På så sätt kunde testobjekt placeras på olika avstånd och vinklar relativt kamerorna utan att någon ny del behövde konstrueras eller skrivas ut.



Figur 4.2: Sidovy av prototypen med kamerornas siktlinjer och hålens mittlinjer inritade. Skärningspunkterna användes för att välja ut optimala testpositioner.

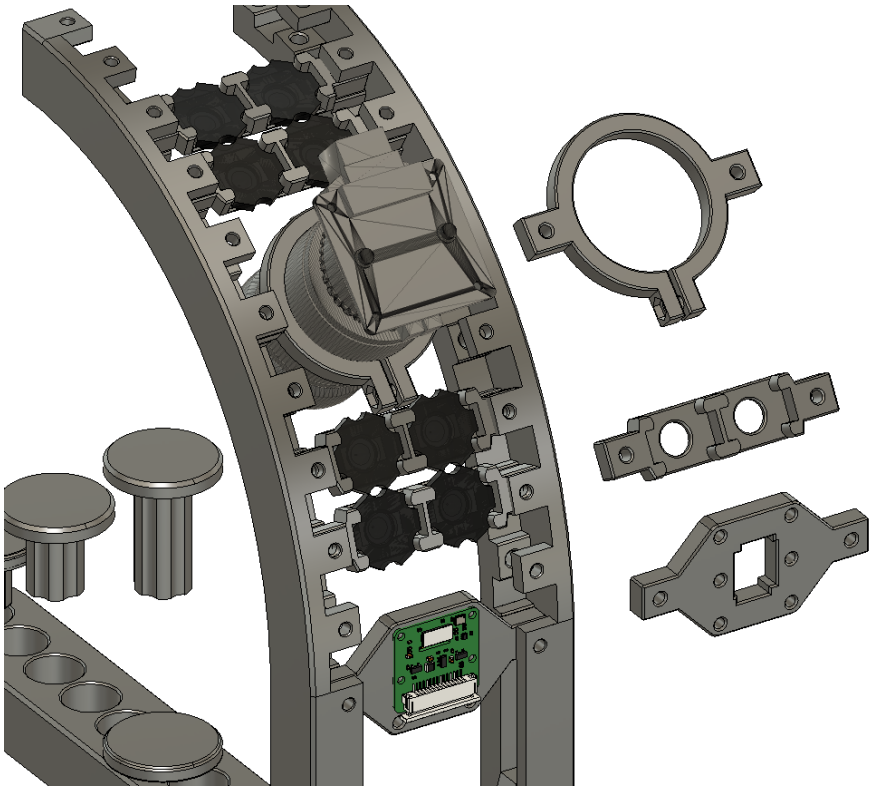
För att systematisera valet av plattformhöjder användes CAD-modellen för att rita ut kamerornas siktlinjer samt hålens mittlinjer, vilket illustreras i figur 4.2. Genom att identifiera skärningspunkterna mellan siktlinjerna och mittlinjerna kunde ett antal kombinationer av hålposition och stapelhöjd identifieras, vilka tillsammans täckte ett representativt urval av vinklar och avstånd som bedömdes vara tillräckligt för utvärdering av kameramodulerna.

4.1.2 Montering av kameror och LED-moduler

Både kameror och LED-moduler monteras i bågen via ett gemensamt system av fästpunkter längs bågens insida, vilket illustreras i figur 4.3. För att förhindra den förslitning som uppstår vid upprepad skruvning direkt i plast är gängade metallinsättningar (så kallade heat-set inserts) inpressade i botten av varje fästpunkt. Maskinskruvarna dras därmed mot metall snarare än mot plast, vilket ger en betydligt mer hållbar infästning

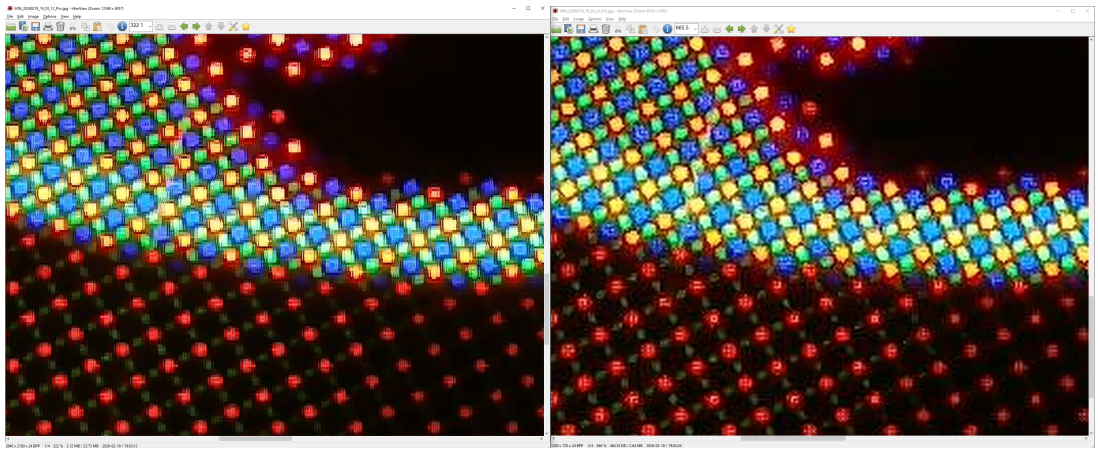
och möjliggör att kameror och LED-moduler kan monteras om upprepade gånger utan att fästpunkterna slits ut.

En klämhållare monteras kring RPI HQ-kameras objektiv. Klämhållaren fästs sedan i bågens fästpunkter med maskinskruvar på samma sätt som övriga komponenter. Genom att klämman omsluter objektivet snarare än att kamerans kretskort används som infästning kan bågens dimensioner hållas inom de som krävs för tillverkning med en kommersiell 3D-skrivare. 64MP-modulen monteras med maskinskruv i en hållare, som sedan monteras i bågen likt övriga moduler.



Figur 4.3: Detaljvy av monteringssystemet för kameror och LED-moduler. Modulernas hållare skruvas fast i gängade metallinsatser monterade i bågens infästningspunkter.

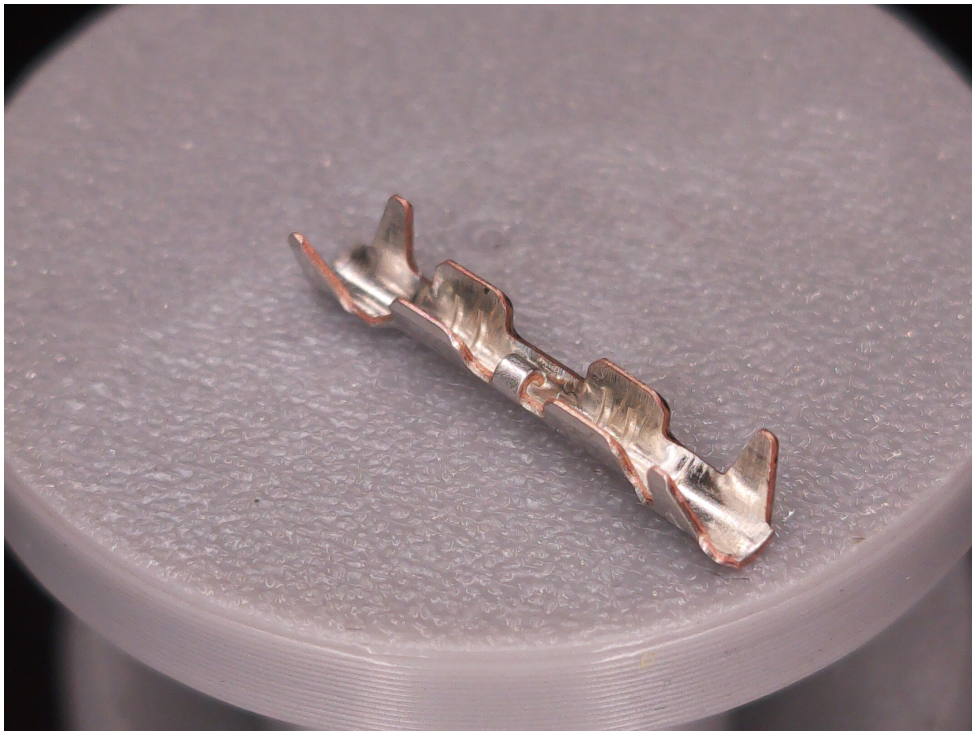
4.1.3 Test av hårdvara med prototypen



Figur 4.4: Jämförelse av bilder från mikroskopkameran. Till höger syns en bild tagen i 720p och till vänster en tagen i "4k". Tydliga tecken på digital bildbehandling syns på vänster sida i form av bland annat vertikala linjer och brist på detaljer som finns i högra bilden.

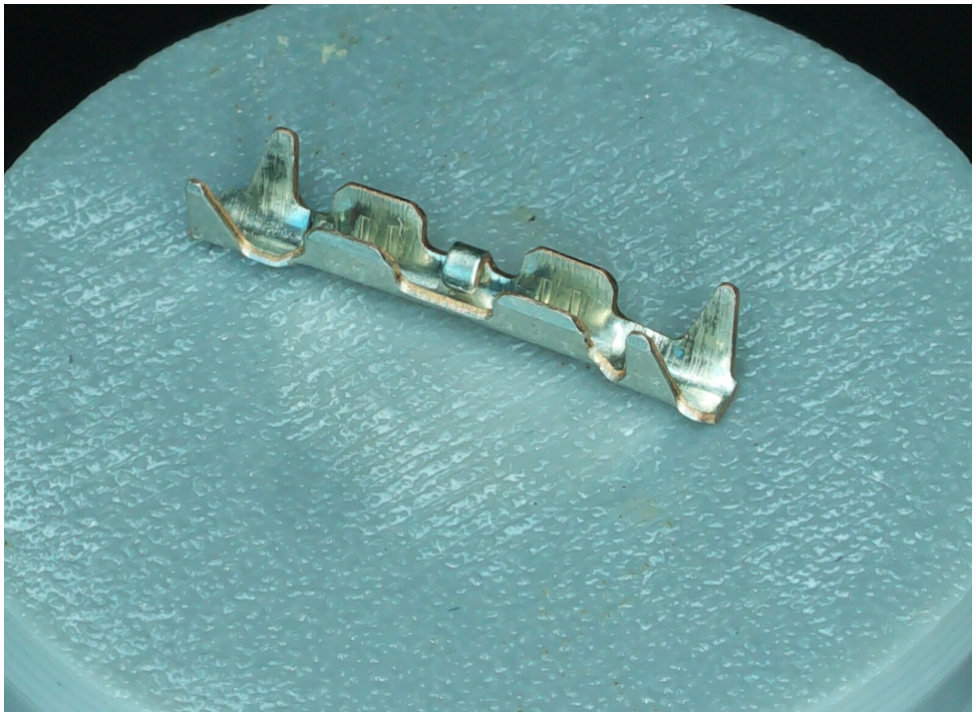
Test av det digitala mikroskopet visade att det inte hade kapacitet att ta riktiga bilder i 8MP. Istället används troligen en mindre sensor vars bilder sedan genom digital bildbehandling skalas upp för att ge en bild i den marknadsförda upplösningen. Se figur 4.4

IMX219 valdes bort på teoretisk grund utan vidare testning. På grund av den mycket breda vinkeln och det fixerade fokuset bedömdes värdet i att testa modulen vara så lågt att det inte var värt tiden.



Figur 4.5: Exempelbild på en fotad terminalkontakt tagen med Arducam 64MP. Notera hur den bakre och främre delen av kontakten ej är i fokus.

Arducam B0483 med 64MP-sensor producerade visserligen högupplösta och skarpa bilder, men till följd av avsaknaden av optisk förstoring upptogs en för stor del av bilden av omgivningen runt testobjektet för att den höga upplösningen skulle kunna utnyttjas effektivt. Arducam 64MP-sensorn kunde inte heller hålla hela testobjektet i fokus från ett hanterbart avstånd. Se figur 4.5. Raspberry Pi HQ-kameran valdes därför för den slutliga designen.



Figur 4.6: Exempelbild på en fotad terminalkontakt tagen med Raspberry Pi HQ-kamera.

Testerna med prototypen visade att Raspberry Pi HQ-kameran med teleobjektiv gav bäst upplösning av testobjektet och kunde upprätthålla ett tillfredsställande skärpedjup. Se figur 4.6.

Efter vidare testning uppskattades ett avstånd på 13cm från objektiv till objekt ge den optimala balansen mellan skärpedjup, ljusinsläpp och närhet till objektet. Detta avstånd användes som utgångspunkt vid dimensionering av den slutliga bågen.

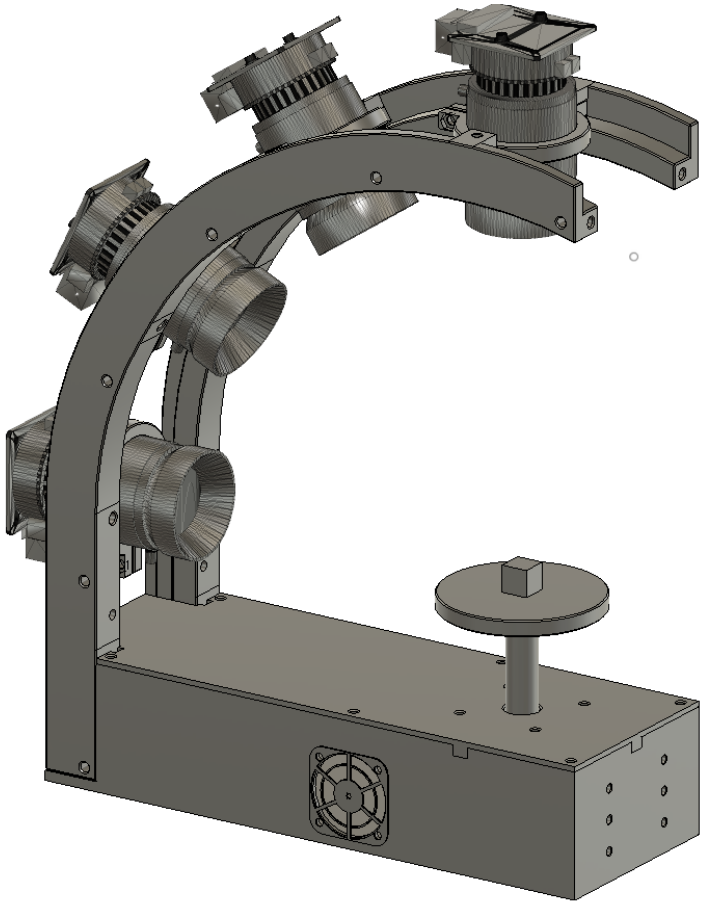
Belysningen utvärderades genom att montera 30cm LED-strip längs insidan av vardera arm, totalt 60cm, vilket vid maximal ljusstyrka motsvarar 10,8W och 1200 lumen. Som jämförelse monterades upp till sex LED-modulhållare med två moduler vardera, totalt 12 moduler à 1W, vilket gav ett nominellt värde på 12W och 2400 lumen enligt tillverkarens specifikation.

LED-modulerna producerade ett mer riktat ljus, men gav upphov till betydligt tydligare skuggor och reflektioner från testobjektet jämfört med LED-stripsen. Ingen kalibrerad luxmätare fanns tillgänglig för objektiv mätning av ljusnivåerna, men en subjektiv bedömning indikerade att tillverkarens angivna 200 lumen per modul var överdrivna, då

LED-stripsen gav till synes mer ljus än modulerna. LED-strips valdes på grund av dessa faktorer som belysningslösning till den slutliga designen.

4.2 Slutlig design

4.2.1 Mekanisk konstruktion

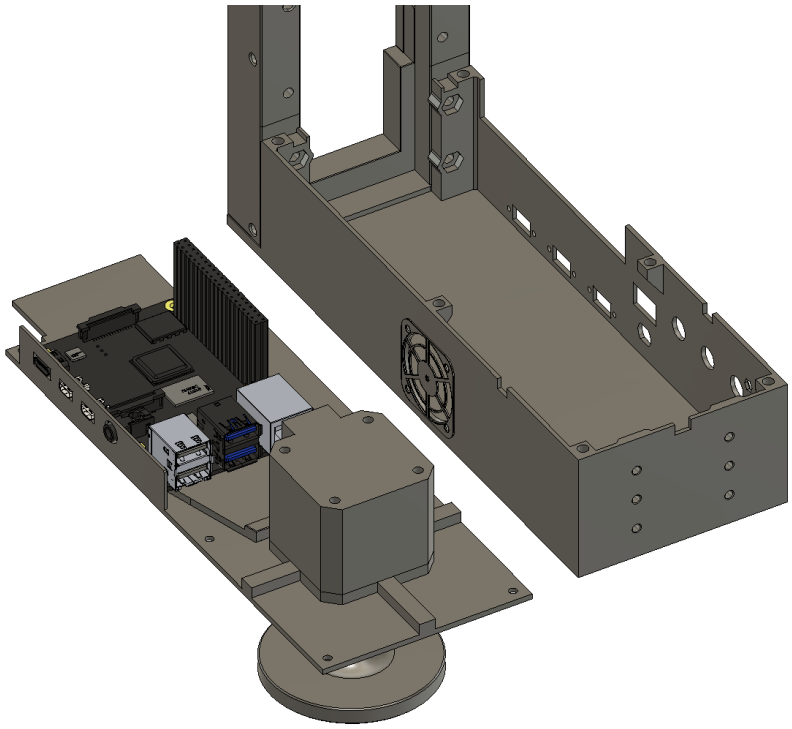


Figur 4.7: Den slutliga designen av skanningssystemet med kameramoduler monterade.

Den slutliga designen bygger vidare på prototypens grundkoncept med två separata bågarmar monterade i en gemensam bas, men med ett antal förändringar baserade på erfarenheterna från prototypfasen. Se figur 4.7. Bågarmarna förlängdes något jämfört med prototypen så att LED-stripsen sträcker sig en bit ovanför objektet, vilket ger bättre och jämnare belysning från ovansidan. Armarna försågs även med fler gänginsättningar än vad som krävs för att möjliggöra montering av eventuella framtida tillägg utan att delarna behöver göras om.

Bågarmarna fästs i elektroniklådans bas med två maskinskruvar vardera i muttrar mon-

terade på insidan av lådan. Det roterande bordet är monterat direkt på stegmotorns D-formade axel vilket ger en spelfri infästning som överför rotationen utan glapp.



Figur 4.8: Vy som visar montering av stegmotor, RPI och platta för stegmotordrivaren på insidan av locket. Syns även hål för muttrar för fäste av armarna, samt hål för potentiometrar, knappar och DC-plugg.

På insidan av locket är stegmotorn, Raspberry Pi och stegmotordrivaren monterade. I lådans botten sitter Arduino Nano, ett DC-jack för strömförsörjning, en 24V-fläkt för kylning, tre potentiometrar för justering av belysningen samt en central strömbrytare och en för fläkten. Denna uppdelning, där belysningskrets och strömförsörjning är samlad i botten medan Raspberry Pi och stegmotor sitter i locket, underlättar montering och demontering genom att hela locket kan lyftas av med tillhörande elektronik, samtidigt som hela lådans utrymme används effektivt. Se figur 4.8.

4.2.2 Val av motorsystem

Valet av motorsystem för det roterande bordet gjordes primärt på teoretisk grund utifrån de krav som ställs på systemet. Det centrala kravet är att bordet ska rotera till exakta och repeterbara vinklar, så att bilder från varje rotation kan kopplas till en känd orientering av objektet.

Tre motorsystem övervägdes: en stegmotor, en DC-motor med encoder samt en vanlig DC-motor utan positionsåterkoppling.

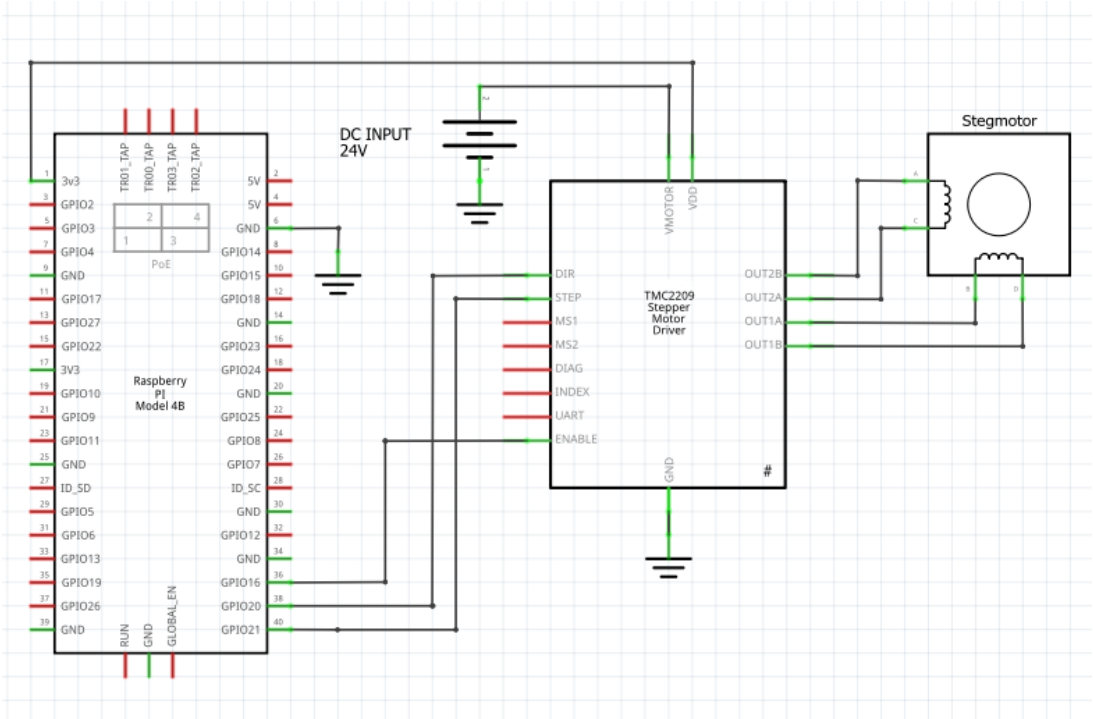
Wiretronics befintliga skanningssystem använder en vanlig DC-motor utan encoder, med kända markörer på rotationsplattan för att i mjukvaran kunna para ihop de olika bilderna med respektive position. Wiretronics system använder emellertid fyra separata systemkameror som är kapabla att exakt samtidigt ta alla fyra bilder. Därför behöver rotationsbordet aldrig sluta snurra, och exaktheten i positioneringen blir mindre viktig. Till följd av att MUX-adaptern i detta projekt bara kan använda en kamera i taget behöver objektet vara stationärt i ett antal sekunder för att ge systemet möjlighet att ta de fyra bilderna. Därför behövs antingen en stegmotor eller en DC-motor med encoder och extern återkoppling, och en vanlig DC-motor uteslöts därför tidigt.

Stegmotorns förmåga att replicerbart rotera ett valt antal grader utan behov av extern återkoppling ger möjlighet att med mycket god precision ta bilder från exakt de vinklar som behövs. Stegmotorns möjlighet att låsa positionen minskar även risken för oönskade rörelser.

Vid praktisk testning uppfyllde stegmotorn omedelbart samtliga krav. Inga fler motorer än stegmotorn testades därför praktiskt. Valet av motorsystem gjordes i och med det på teoretisk grund.

4.2.3 Elektrisk konstruktion

4.2.3.1 Stegmotor

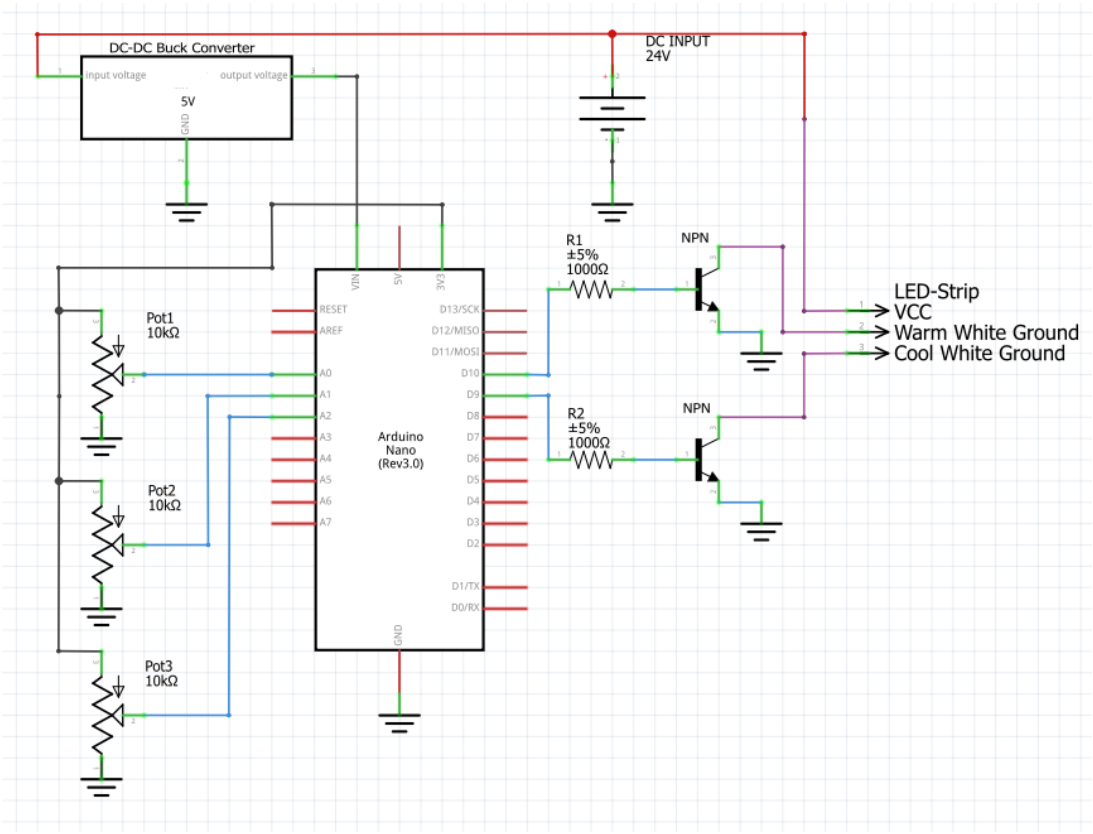


Figur 4.9: Kopplingsschema för stegmotorsystemet. Raspberry Pi 4B styr TMC2209 stegmotordrivaren via GPIO-pinnarna 21 (STEP), 20 (DIR) och 16 (ENABLE).

Stegmotorn styrs av en TMC2209 stegmotordrivare som tar emot styrsignaler från Raspberry Pi 4B via tre GPIO-pinnar. Kopplingsschemat visas i figur 4.9.

GPIO-pinne 21 (STEP) skickar pulser till drivaren där varje puls motsvarar ett steg, GPIO-pinne 20 (DIR) bestämmer rotationsriktningen, och GPIO-pinne 16 (ENABLE) aktiverar drivaren. Stegmotorn försörjs med 24V via stegdrivaren, medan själva stegdrivaren försörjs med 3.3V från Raspberry Pi.

4.2.3.2 Belysningssystem



Figur 4.10: Kopplingsschema för belysningssystemet. Tre potentiometrar kopplade till Arduino Nano styr ljusstyrkan hos de två LED-kanalerna via NPN-transistorer. Nano försörjs med 5V från en DC-DC buck converter.

Belysningssystemet styrs av en Arduino Nano som läser av tre potentiometrar och med PWM-signaler via två NPN-transistorer reglerar ljusstyrkan hos de två LED-kanalerna, kallt vitt och varmt vitt. Kopplingsschemat visas i figur 4.10.

LED-stripsen försörjs med 24V direkt från den primära strömkällan, medan transistorerna bryter på jord-sidan (low-side switching). Detta innebär att transistorerna reglerar hur mycket ström som får flöda från LED-stripens respektive kanaler till jord, vilket är en enkel och tillförlitlig metod för analog dimming av LEDs.

Programkodens funktion innebär att varje kanals slutliga ljusstyrka är produkten av kanalens egna potvärde och masterpotens värde, skalat till ett 8-bitars PWM-värde mellan 0 och 255. Detta ger operatören möjlighet att oberoende justera balansen mellan kallt och varmt ljus, samt skala upp eller ner den totala ljusnivån utan att balansen förändras.

4.3 Programmering

4.3.1 Arduino

Arduinons kod läser kontinuerligt av tre potentiometrar via de analoga ingångarna A0, A1 och A2. De analoga värdena, som har ett intervall på 0–1023, konverteras till 8-bitars värden mellan 0 och 255 genom ett bitskift med två steg.

Pot1 (A0) styr andelen kallt vitt ljus, Pot2 (A1) styr andelen varmt vitt ljus, och Pot3 (A2) fungerar som en masterkontroll för den totala ljusstyrkan. Varje kanals slutliga ljusstyrka beräknas som produkten av kanalens eget potvärde och masterpotens värde, skalat tillbaka till ett 8-bitarsvärde genom ytterligare ett bitskift. Det resulterande värdet skrivs ut som en PWM-signal på utgångarna D9 och D10, som styr transistorerna för respektive LED-kanal.

Den fullständiga koden återfinns i Appendix A.1.

4.3.2 Raspberry Pi

Programmet på Raspberry Pi koordinerar hela skanningsprocessen och tillhandahåller ett grafiskt gränssnitt för operatören. Gränssnittet är byggt med biblioteket pygame och visar en förhandsvisning från den valda kameran tillsammans med en sidopanel för inställningar.

Kamerorna styrs via Raspberry Pi:s rpicam-/libcamera-library, som anropas från Python för att hämta förhandsvisningsbilder och ta högupplösta stillbilder. Vid uppstart identifierar programmet automatiskt vilka IMX477-kameror som är anslutna via adapterkortet.

Via gränssnittet kan operatören:

- Växla mellan de fyra kamerorna och se en förhandsvisning av vald kamera
- Aktivera hjälplinjer i form av kors och rutnät för att underlätta positionering av objektet på plattan.
- Ställa in antal stopp per varv, vilket avgör rotationsvinkeln mellan varje bildtagning och då även det totala antalet bilder som tas.
- Ange ett namn för mappen där bilderna sparas.
- Starta skanningen och se hur många rotationer som återstår.

När skanningen startas körs en sekvens där systemet vid varje stopp fotograferar med samtliga fyra kameror i tur och ordning. Eftersom adapterkortet endast tillåter en aktiv kamera åt gången införs en kort fördröjning om 0.2 sekunder mellan varje kamera för att säkerställa att rätt kamera hunnit aktiveras. För varje bild görs vid behov totalt tre försök, vilket ökar tillförlitligheten om en enskild kamera misslyckas enstaka gånger.

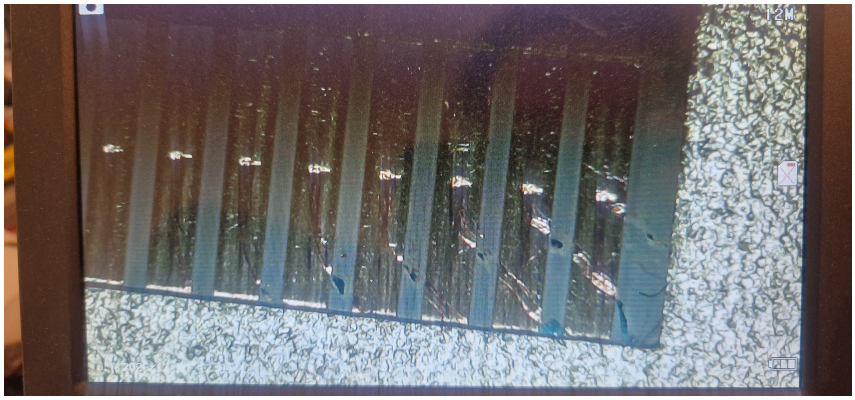
Efter varje stopp roterar stegmotorn bordet ett bestämt antal grader, beräknat som 360 grader dividerat med antalet stopp, varefter systemet väntar en kort stund för att låta objektet stabiliseras innan nästa bildtagning. När alla stopp är genomförda återgår bordet automatiskt till startpositionen.

Bilderna sparas med filnamn som anger vilken kamera och vilken sektion de tillhör, vilket gör det enkelt att i efterhand koppla varje bild till en känd kameraposition och rotationsvinkel.

Den fullständiga koden återfinns i Appendix A.2.

4.4 Problem och lösningar

4.4.1 Störningar och kortslutning



Figur 4.11: Närbild på den felkonstruerade kontaktytan på CSI-kabeln, där ledare sträckte sig över flera pins och orsakade kortslutning. Notera de svarta märkena mellan pinsen där plasten smält.

I ett tidigt skede av testningen producerades mycket störningar av bilden vid användning av 4-MUX-adaptern, samt sporadisk frånkoppling av kameror och allmän instabilitet.

Den medföljande CSI-kabeln till 4-Mux-adapterkortet visade sig under felsökningen vara felkonstruerad. Till en början hade en annan CSI-kabel använts, men under felsökningen

testades även den medföljande. Kontaktytan på kabeln hade tillverkats felaktigt, med ledare som sträckte sig över flera pins samtidigt, se figur 4.11 Detta orsakade kortslutning i adapterkortet, vilket ledde till att kortet gick sönder och ett nytt fick införskaffas. Det observerades dock att varken Raspberry Pi eller kameramodulerna tog skada.

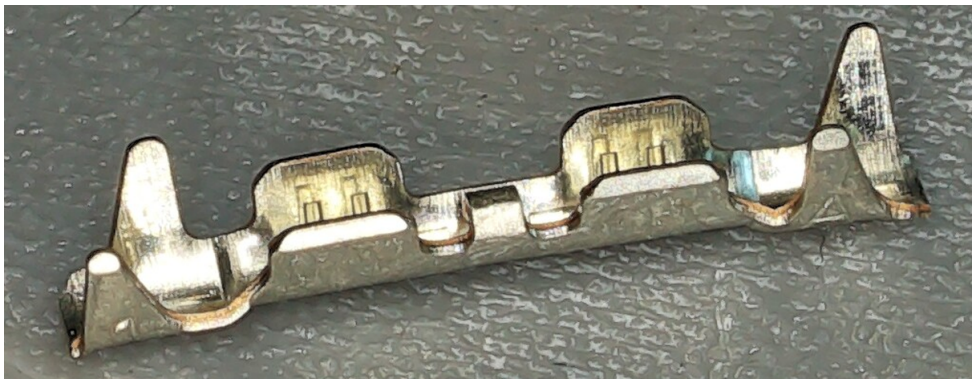
Trots ny adapter och CSI-kabel fortsatte störningarna. Efter fortsatt felsökning i både mjukvara och hårdvara visade sig grundproblemet vara att CSI-kabeln mellan adapterkortet och Raspberry Pi låg vikt i kontakt med sig själv. Med en kabelstrumpa skapades ett skyddande hölje runt kabeln för att förhindra direktkontakt. Detta löste problemet omedelbart och systemet har därefter varit stabilt.

5

Resultat

Det färdiga skanningssystemet uppfyller de krav som ställdes i frågeställningen. Systemet producerar detaljerade bildserier på testobjekten från flera vinklar och fungerar robust och tillförlitligt i ett kompakt format. Nedan presenteras resultaten kopplat till de tre delfrågorna.

5.1 Avbildning



Figur 5.1: Exempelbild på en skannad terminalkontakt tagen med systemets Raspberry Pi HQ-kamera. Terminalkontakten är ca 13*3*3 mm stor

Med Raspberry Pi HQ-kameran och teleobjektivet uppnås en god bildkvalitet med tillräckligt skärpedjup, hög detaljnivå och en optisk förstoring som gör att objektet fyller en stor del av bilden. Hela testobjektet kan hållas i fokus samtidigt, och detaljnivån är tillräcklig för att urskilja små detaljer på de avbildade komponenterna. Se figur 5.1.

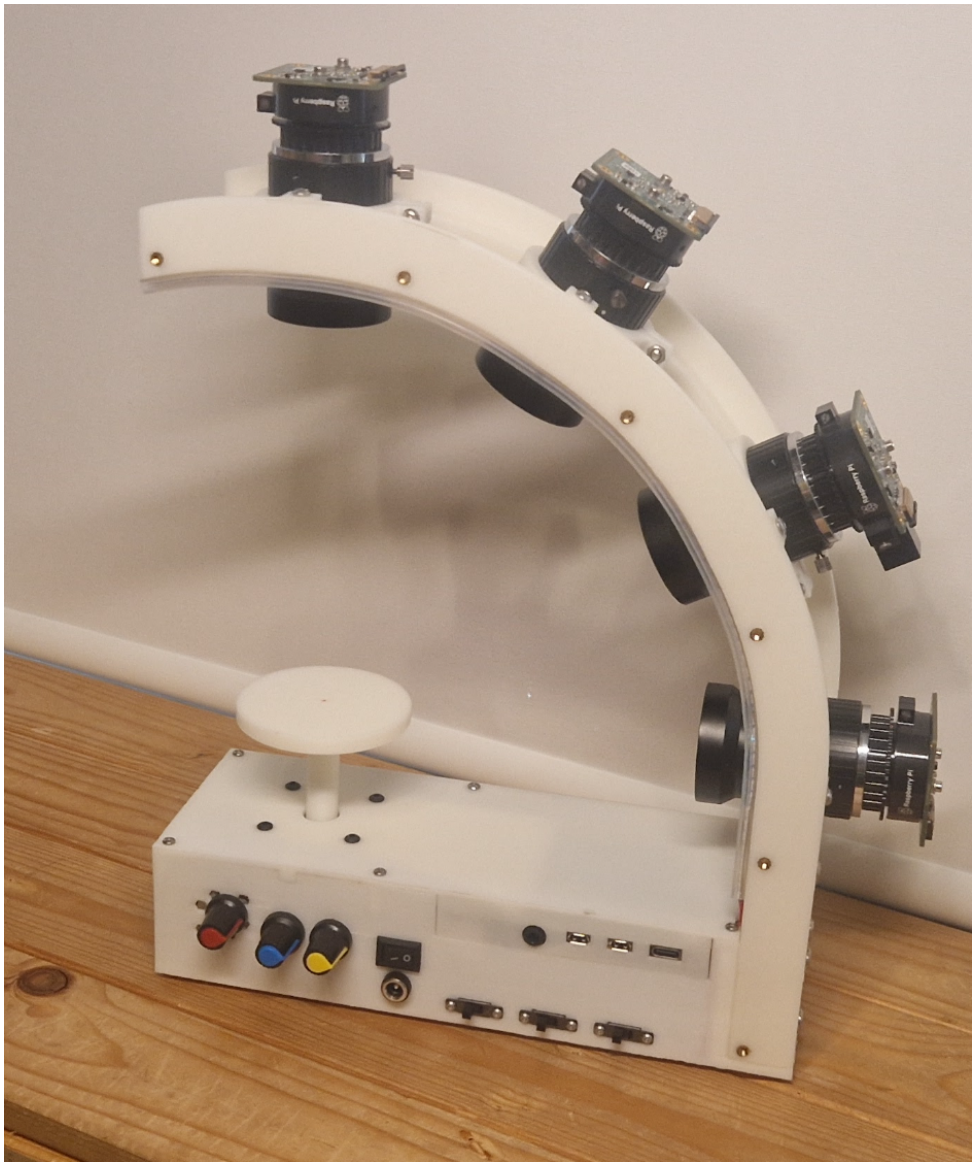
5.2 Rotationssystem

Rotationssystemet fungerar väl och stegmotorn roterar till rätt position med hög precision och god repeterbarhet. När stegmotorn hålls i låst läge uppstår små vibrationer i bordet, men dessa har inte gett några observerbara negativa effekter på bildkvaliteten.

5.3 Belysning

Belysningen med LED-strips fungerar väl och ger ett jämnt och tillräckligt ljus för möjliggörandet av högdetaljerade bilder. Den justerbara ljusstyrkan och färgtemperaturen gör det möjligt att anpassa belysningen efter objekt och förhållanden.

5.4 Det färdiga systemet

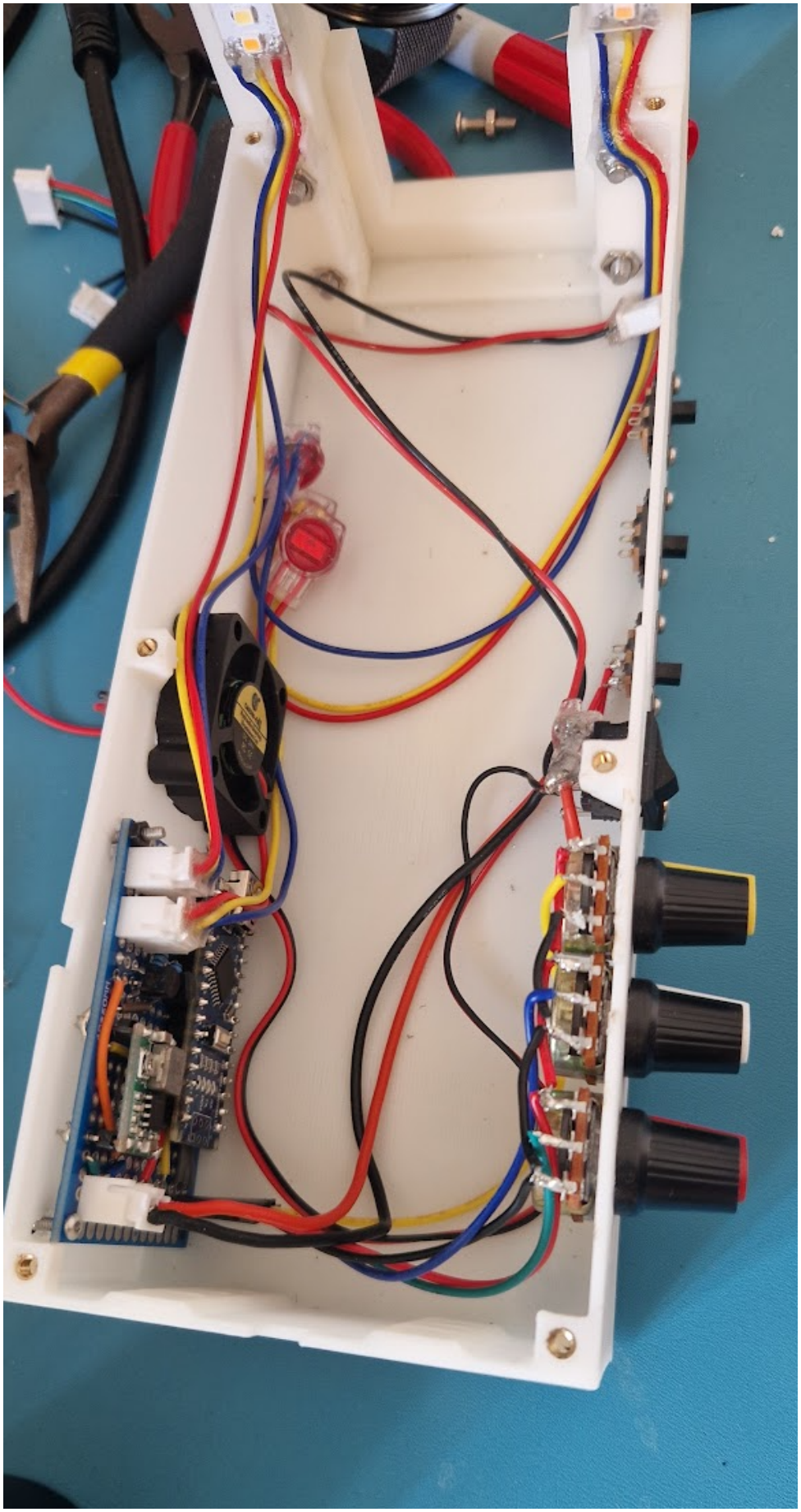


Figur 5.2: Det färdiga skanningssystemet. CSI-kablage till kameramodulerna är ej installerat på bilden. Systemet är ca 270*80*220 mm stort.

Systemet som helhet fungerar väl och upplevs som robust. Se figur 5.2. Fläkten i basen håller elektronikens temperatur på en stabil nivå under drift. De geometriska toleransavvikelser som uppstått vid 3D-printing med kommersiell utrustning hade mindre påverkan än förväntat, även om inriktningen av kamerorna hade kunnat fungera bättre. Systemet är mycket kompakt, och skulle troligen inte kunna göras så mycket mer kompakt utan stora designändringar.

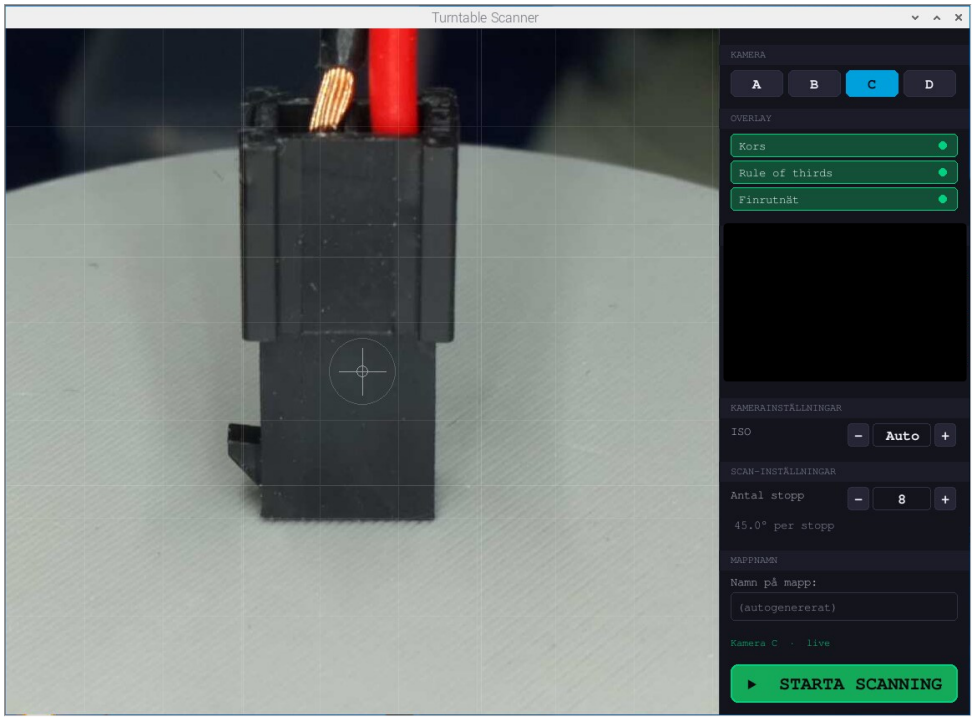
Antalet rotationsvinklar kan ställas in fritt av operatören. En uppsättning om fyra bilder,

en från varje kamera, tar cirka fyra sekunder att fånga, vilket motsvarar ungefär en sekund per bild. En fullständig skanning från tio olika rotationsvinklar tar därmed cirka 40 sekunder och resulterar i 40 bilder.



Figur 5.3: Vy av elektroniklådans insida, med potentiometrar monterade till höger samt fläkt och Arduino-kretskort monterat till vänster. Det lediga utrymmet ockuperas av Raspberry Pi, stegmotordrivare och stegmotor då locket är monterat.

Insidan av elektroniklådan använder utrymmet väl, med plats för stegmotor och Raspberry Pi med kontakter. Se figur 5.3. Anslutning mellan elektroniklåda och lock sker enbart med en enda kontakt, för 24v ström till stegdrivaren.



Figur 5.4: Skärmdump som visar systemets användargränssnitt med en live preview till vänster, och menyalternativ till höger.

Användargränssnittet visar en live preview från en kamera som kan ändras i "KAMERA-sektionen av menyn till höger. Se figur 5.4. Det går att aktivera eller avaktivera ett antal overlays som är ämnade att hjälpa användaren centrera objektet väl. I menyn kan även ISO, antalet stopp och namnet på mappen där bilderna sparas bestämmas. Efter start av scanning visas antal stopp som återstår, samt en knapp för att avbryta och återgå till startläget.

Under testningen av den färdiga prototypen förekom inga misslyckade fotograferingar eller omförsök, vilket tyder på att åtgärderna som tagits för att isolera CSI-kabeln mellan MUX-adapter och Raspberry Pi fungerar och att systemet är stabilt i drift.

6

Diskussion

6.1 Val av kamerasystem och avbildning

De avgränsningar och beslut som togs gällande fotografilösning upplevs ha varit välgrundade och ger ett gott resultat. Raspberry Pi HQ-kameran med teleobjektiv levererar bildkvalitet som väl uppfyller kraven. Möjligen hade ett ännu bättre anpassat objektiv kunnat förbättra resultatet ännu mer.

Ett alternativ till de utvärderade kameramodulerna hade varit att använda systemkameror med makroobjektiv, liknande de Wiretronic använder i sina befintliga system. Detta hade dock medfört en betydligt högre kostnad och ett mycket större system, vilket går emot målet att skapa en kompakt lösning.

6.2 MUX-adapterns begränsning

En central begränsning i systemet är att 4-Mux-adaptern bara tillåter en aktiv kamera åt gången, vilket innebär att objektet måste vara stationärt under fotograferingen. Lösningen känns dock tillräcklig, då resultatet blev tillfredsställande och den totala skanningstiden är fullt acceptabel.

Ett alternativ hade varit att använda fyra separata Raspberry Pi-enheter för att möjliggöra simultan fotografering. Med fyra Raspberry Pi Zero hade systemet kunnat hållas relativt kompakt, men det hade samtidigt introducerat betydligt mer komplexitet i form av synkronisering och datahantering. Den valda lösningen med en enda Raspberry Pi och ett adapterkort ger en god balans mellan simplicitet, kostnad, storlek och prestanda.

6.3 Val av motorsystem

Stegmotorn sågs som det självklara valet för rotationssystemet, framför allt på grund av MUX-adapterns begränsning som kräver exakt och repeterbar positionering med möjlighet att hålla objekten stationärt. Beslutet att välja stegmotorn på teoretisk grund utan jämförande praktisk testning av övriga alternativ kan ifrågasättas ur ett metodbaserat perspektiv, men eftersom stegmotorn omedelbart uppfyllde kraven vid praktisk testning hade vidare jämförelser varit överflödiga och ett slöseri med tid.

6.4 Belysningsstyrning

Den lösning som valdes för belysningsstyrning, med fristående potentiometrar och en Arduino Nano, fungerar väl men hade kunnat förbättras. Valet motiveras av ett beslut gällande elektroniklådans utformning. För att förenkla montering och demontering bestämdes att montera Raspberry Pi, stegmotor och stegmotordrivare i lockets undersida, medan belysningskretsen placerades i lådans botten.

6.5 Framtida arbete

Digital styrning av belysningen via Raspberry Pi skulle ge ett mer komplett intryck och framför allt öka möjligheten att reproducera identiska ljusförhållanden mellan olika användningar. Detta hade inte varit särskilt svårt att implementera, troligen till och med enklare än den valda lösningen, och är något som jag med facit i hand känner att jag borde gjort annorlunda.

Utöver digital belysningsstyrning finns flera möjliga förbättringar av systemet. En vit bakgrund bakom objektet skulle kunna reflektera ljus mot objektet och minska påverkan från omgivningen, vilket antagligen skulle förbättra bildkvaliteten ytterligare.

Montering av kameramodulerna direkt via deras kretskortsfästen, istället för det klämbaserade systemet runt objektivet, skulle ge en fastare infästning. Detta skulle dock kräva en större bågradie eller en uppdelning av bågen i fler delar, vilket var skälet till att klämlösningen ändå valdes.

En mer precis tillverkningsmetod än 3D-printing skulle sannolikt minska de geometriska toleransavvikelser som ger upphov till den lilla rörelse som kan observeras vid rotation av bordet.

Att få kamerorna korrekt inriktade mot objektet visade sig vara svårt. Mest troligt på grund av icke perfekt geometri vid 3D-printing. Ett sätt runt detta hade varit att designa kameramodulernas hållare på ett sätt som möjliggjorde små justeringar av positionering. Enbart montering i modulernas PCB hade troligen inte löst detta problem.

Test av fler kameramoduler och linser skulle eventuellt kunna ta fram en kamera som ger bättre resultat och möjlighet att hålla större objekt i fokus.

6.6 Arbetsprocess

Upplägget med arbetsprocessen fungerade väl. Den första prototypen gav mycket värdefull information och fungerade som en utmärkt testbädd för utvärdering av kameramoduler, objektavstånd och belysning. Genom att separera utvärderingsfasen från den slutliga konstruktionen kunde jag fatta välgrundade beslut innan det slutliga systemet konstruerades.

6.7 Etik och hållbarhet

Ur ett hållbarhetsperspektiv har systemet flera positiva egenskaper. Materialåtgången för de 3D-printade delarna är liten, och mycket lite material gick till spillo under tillverkningen. Systemets totala energiförbrukning är låg, omkring 25W under drift. Energiförbrukningen skulle kunna minskas ytterligare genom digital styrning av belysningen, där LED-belysningen endast aktiveras i samband med att bilderna tas istället för att vara tänd under hela processen.

Konstruktionens uppdelning i separata delar underlättar reparation och utbyte av enskilda komponenter och delar, vilket bidrar till en längre livslängd för systemet. Skulle någon del av konstruktionen gå sönder kan man 3D-printa en ny.

Genom att möjliggöra fotografering av små komponenter kan systemet bidra till bättre serviceunderlag för Wiretronics produkter. Detta kan i förlängningen förlänga produkternas livslängd genom att underlätta reparation och reservdelshantering, istället för att

hela produkter behöver bytas ut när enskilda komponenter går sönder eller blir för svåra att identifiera.

Ur ett etiskt perspektiv bedöms projektet ha mycket liten påverkan eftersom ingen personlig eller känslig data hanteras.

7

Slutsats

Målet med arbetet var att bygga ett kompakt system som kan ta detaljerade bilder av små objekt från flera vinklar. Det målet är uppnått. Det färdiga systemet fungerar bra och tar detaljerade bildserier på ett pålitligt sätt.

När det gäller avbildning var Raspberry Pi HQ-kameran med teleobjektiv det bästa av de alternativ jag testade. Tack vare den optiska förstoringen fyller objektet en stor del av bilden, vilket ger hög detaljnivå och ett skärpedjup som räcker för de objekt systemet är tänkt för.

För rotationen var stegmotorn det självklara valet. Den roterar till exakta och repeterbara vinklar utan att behöva någon extern positionsåterkoppling, vilket är helt avgörande eftersom adapterkortet bara kan använda en kamera i taget och objektet därför måste stå still under själva fotograferingen.

För belysningen gav LED-stripsen klart bäst resultat. De gav ett jämnare ljus med mindre skuggor och reflektioner än de centralt monterade LED-modulerna, och med justerbar ljusstyrka och färgtemperatur går det att anpassa ljuset efter objektet.

Sammantaget gör systemet det jag och Wiretronic ville att det skulle göra, till en låg kostnad och med ett litet fotavtryck. Det kan användas för att samla in bilder för AI-träning och 3D-modellering av Wiretronics minsta komponenter, och fungerar dessutom som en bra grund att bygga vidare på.

Referenser

[1] Jernkontoret: *Elmotorer, teknik och funktion*, Jernkontoret, https://www.energihandbok.se/elmotorer-teknik-och-funktion (Acc 2026-06-03)

[2] Phidgets: *Stepper Motor and Controller Guide*, Phidgets Inc, https://www.phidgets.com/docs/Stepper_Motor_and_Controller_Guide (Acc 2026-06-03)

[3] Joylit: *LED Strip A2835-224-CCT-V24-TWP, produktförpackning*, Joylit

[4] Hilitand: *Hilitand 25Pcs LED Chips LED Lamp Beads DIY Lighting Fixtures High Power 5V 200LM 1W (Warm White 3000-3200K)*, Hilitand, https://www.amazon.se/-/en/Hilitand-25Pcs-Lighting-Fixtures-3000-3200K/dp/B09VM7KPFY (Acc 2026-06-03)

[5] Leopard Imaging Inc.: *LI-IMX219-MIPI-FF-NANO Specification*, Leopard Imaging Inc, https://cdn.sparkfun.com/assets/6/9/7/7/3/DS-16260-Leopard_Imaging_Camera_-_136_Degree_FOV.pdf (Acc 2026-06-03)

[6] Raspberry Pi Ltd: *High Quality Camera Product Brief*, Raspberry Pi Ltd, 2023, https://datasheets.raspberrypi.com/hq-camera/hq-camera-product-brief.pdf (Acc 2026-06-03)

[7] Raspberry Pi Ltd: *PT3611614M10MP C-Mount 16mm Telephoto Lens Datasheet*, https://www.alldatasheet.com/datasheet-pdf/pdf/1425040/ETC/PT3611614M10MP.html (Acc 2026-06-03)

[8] Cainda: *4K WiFi Digital Microscope X201*, Cainda, https://caindahelp.com/cainda-4k-wifi-digital-microscope-x201-p00095p1.html (Acc 2026-06-03)

[9] Arducam: *64MP OwlSight OV64A40 Camera Module Product Brief*, Arducam, 2023, https://blog.arducam.com/downloads/datasheet/Arducam_64mp_ov64a40_product_brief.pdf (Acc 2026-06-03)

[10] Arducam: *Multi-Camera Adapter Board B012001 Quick Start Guide*, Arducam, https://docs.arducam.com/Raspberry-Pi-Camera/Multi-Camera-CamArray/Quick-Start-Guide-for-Multi-Adapter-Board/ (Acc 2026-06-03)

[11] Arduino: *Nano Documentation*, Arduino, 2025, https://docs.arduino.cc/hardware/nano (Acc 2026-06-03)

[12] Raspberry Pi Ltd: *Raspberry Pi 4 Model B*, Raspberry Pi Ltd, 2026, https://pip-assets.raspberrypi.com/categories/545-raspberry-pi-4-model-b/documents/RP-008344-DS-5-raspberry-pi-4-product-brief.pdf (Acc 2026-06-03)

[13] Fused filament fabrication, Wikipedia, https://en.wikipedia.org/wiki/Fused_filament_fabrication (Acc 2026-06-03)

[14] Pulse-width modulation, Wikipedia, https://en.wikipedia.org/wiki/Pulse-width_modulation (Acc 2026-06-03)

[15] Analog Devices: *Advantages of Switching Regulators Over Linear Regulation*, Analog Devices, https://www.analog.com/en/resources/technical-articles/advantages-of-switching-regulators-over-linear-regulation.html (Acc 2026-06-03)

[16] Anthropic PBC: *Generated using Claude Code*, Claude Code Opus 4.6, 2026, [Large Language Model]

A

Källkod

Programkoden i denna appendix har genererats, felsökts och modifierats med Claude Code. Jag har sedan granskat och anpassat koden. [16]

A.1 Arduinokod

```
//Definierar pins för input och output
const int potCool = A0, potWarm = A1, potMaster = A2;
const int outCool = 9, outWarm = 10;

void setup() {
  //Initierar pins 9 och 10 som outputs
  pinMode(outCool, OUTPUT);
  pinMode(outWarm, OUTPUT);
}

void loop() {
  //Läser av de tre potentiometrarna och skalar ner värdet
  int cool = analogRead(potCool) >> 2;      // -0255
  int warm = analogRead(potWarm) >> 2;      // -0255
  int master = analogRead(potMaster) >> 2;  // -0255

  // Multiplicerar varje kanal med master brightness och skalar sedan
     ner värdet
  cool = (cool * master) >> 8;
  warm = (warm * master) >> 8;

  //Skriver ut värdet till transistorerna på pin 9 och 10
  analogWrite(outCool, cool);
  analogWrite(outWarm, warm);
```

```
24 }
```

[16]

A.2 Pythonkod

```
[language=Python]
#!/usr/bin/env python3
"""
turntable_scan.py - Turntable scanner med live-preview, sidopanel för
overlay-toggles och kamerainställningar, samt automatisk återgång till start.

Flöde

  1. Preview-fönster öppnas. Sidopanel till höger visar:
       · Kameraväljare (A/B/C/D)
       · Overlay-toggles (kors, rutnät, rule-of-thirds)
       · Kamerainställningar (EV, shutter, ISO, AWB)
       · Scan-inställningar (stopp, step-delay, settle)
     [Enter] eller knappen "STARTA SCANNING" startar.
     [Esc/Q] avslutar.

  2. Scanner fotograferar alla kameror vid varje stopp.

  3. Motorn återgår till startposition, preview öppnas igen.

Beroenden

  sudo apt install -y rpicam-apps python3-pygame
  pip3 install RPi.GPIO
"""

import sys
```

```
import time
import argparse
import subprocess
import tempfile
import threading
from pathlib import Path
from datetime import datetime

try:
    import RPi.GPIO as GPIO
except ImportError:
    print("[!] RPi.GPIO inte hittat. Kör: pip3 install RPi.GPIO")
    sys.exit(1)

try:
    import pygame
except ImportError:
    print("[!] pygame inte hittat. Kör: sudo apt install -y python3-pygame")
    sys.exit(1)



#  GPIO
STEP_PIN = 21
DIR_PIN  = 20
EN_PIN   = 16

#  Kameror
CAMERA_LABELS  = {0: "A", 1: "B", 2: "C", 3: "D"}
LABEL_TO_INDEX = {"A": 0, "B": 1, "C": 2, "D": 3}

#  Layout
PREVIEW_W = 854          # bredden på kamerabilden
```

```
PANEL_W   = 300          # sidopanelens bredd
WIN_W     = PREVIEW_W + PANEL_W
WIN_H     = 680

#  Färgpalett
C_BG        = ( 12,  12,  16)
C_PANEL     = ( 20,  20,  28)
C_PANEL2    = ( 28,  28,  40)   # sektionshuvud
C_BORDER    = ( 50,  50,  70)
C_ACCENT    = (  0, 210, 130)   # grön
C_ACCENT2   = (  0, 160, 220)   # blå (aktiv kamera)
C_DIM       = (140, 145, 165)
C_TEXT      = (240, 242, 250)
C_TEXTDIM   = (185, 190, 205)
C_WARN      = (230, 185,   0)
C_RED       = (210,  55,  55)
C_BTN       = ( 38,  38,  55)
C_BTN_HOV   = ( 55,  55,  78)
C_BTN_ACT   = ( 20,  80,  55)   # aktiv toggle
C_START     = ( 20, 170,  90)
C_START_HOV = ( 30, 210, 110)



#
# rpicam / libcamera helpers
#

def find_base():
    for name in ("rpicam", "libcamera"):
        try:
            subprocess.run([f"{name}-hello", "--help"],
                           capture_output=True, timeout=3)
```

```
            return name
        except Exception:
            pass
    return None


def get_camera_indices(base):
    try:
        r = subprocess.run([f"{base}-hello", "--list-cameras"],
                           capture_output=True, text=True, timeout=5)
        output = r.stdout + r.stderr
        indices = []
        for line in output.splitlines():
            s = line.strip()
            if s and s[0].isdigit() and "imx477" in s.lower():
                try:
                    idx = int(s.split(":")[0].strip())
                    if idx not in indices:
                        indices.append(idx)
                except ValueError:
                    pass
        if indices:
            return sorted(indices)
        if "imx477" in output.lower():
            return [0, 1, 2, 3]
    except Exception as e:
        print(f"[!] Kunde inte lista kameror: {e}")
    return []


def capture_once(base, cam_index, output_path, warmup_ms, cam_settings=None):
    if output_path.exists():
```

```
        output_path.unlink()
    cmd = [
        f"{base}-still",
        "--camera",   str(cam_index),
        "--output",   str(output_path),
        "--timeout",  str(warmup_ms),
        "--nopreview",
        "--encoding", "jpg",
        "--quality",  "95",
    ]
    if cam_settings:
        ev  = cam_settings.get("ev", 0.0)
        iso = cam_settings.get("iso", 0)      # 0 = auto
        shu = cam_settings.get("shutter", 0)  # 0 = auto (µs)
        awb = cam_settings.get("awb", "auto")
        if ev != 0.0:
            cmd += ["--ev", str(ev)]
        if iso > 0:
            cmd += ["--analoggain", str(iso / 100.0)]
        if shu > 0:
            cmd += ["--shutter", str(shu)]
        if awb != "auto":
            cmd += ["--awb", awb]
    try:
        result = subprocess.run(cmd, capture_output=True, text=True,
                                timeout=warmup_ms / 1000 + 15)
        if result.returncode == 0 and output_path.exists():
            return True, output_path.stat().st_size // 1024, None
        err_lines = result.stderr.strip().splitlines()
        err = next((l for l in reversed(err_lines)
                    if "ERROR" in l or "error" in l.lower()), "unknown error")
        return False, 0, err
```

```
    except subprocess.TimeoutExpired:
        return False, 0, "timed out"
    except Exception as e:
        return False, 0, str(e)


def capture_with_retry(base, cam_index, output_path, warmup_ms,
                       retries, retry_wait_ms, cam_settings=None):
    for attempt in range(1, retries + 2):
        ok, size_kb, err = capture_once(base, cam_index, output_path,
                                        warmup_ms, cam_settings)
        if ok:
            return True, size_kb, attempt
        if attempt <= retries:
            print(f"retry {attempt}/{retries}... ", end="", flush=True)
            time.sleep(retry_wait_ms / 1000)
        else:
            print(f"({err}) ", end="", flush=True)
    return False, 0, retries + 1


def capture_preview(base, cam_index, output_path, timeout_ms=800,
                    cam_settings=None):
    if output_path.exists():
        output_path.unlink()
    cmd = [
        f"{base}-still",
        "--camera",   str(cam_index),
        "--output",   str(output_path),
        "--timeout",  str(timeout_ms),
        "--nopreview",
        "--encoding", "jpg",
```

```
        "--quality",  "55",
        "--width",    str(PREVIEW_W),
        "--height",   str(WIN_H),
    ]
    if cam_settings:
        ev  = cam_settings.get("ev", 0.0)
        iso = cam_settings.get("iso", 0)
        shu = cam_settings.get("shutter", 0)
        awb = cam_settings.get("awb", "auto")
        if ev != 0.0:
            cmd += ["--ev", str(ev)]
        if iso > 0:
            cmd += ["--analoggain", str(iso / 100.0)]
        if shu > 0:
            cmd += ["--shutter", str(shu)]
        if awb != "auto":
            cmd += ["--awb", awb]
    try:
        result = subprocess.run(cmd, capture_output=True, text=True,
                                timeout=timeout_ms / 1000 + 10)
        return result.returncode == 0 and output_path.exists()
    except Exception:
        return False


#
# Motor helpers
#

def motor_setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(STEP_PIN, GPIO.OUT)
```

```
    GPIO.setup(DIR_PIN,  GPIO.OUT)
    GPIO.setup(EN_PIN,   GPIO.OUT)
    GPIO.output(EN_PIN, GPIO.LOW)


def motor_release():
    GPIO.output(EN_PIN, GPIO.HIGH)
    GPIO.cleanup()


def move_degrees(degrees, microsteps=8, step_delay=0.001, direction=True):
    steps_per_rev = 200 * microsteps
    steps = int((degrees / 360.0) * steps_per_rev)
    GPIO.output(DIR_PIN, GPIO.HIGH if direction else GPIO.LOW)
    for _ in range(steps):
        GPIO.output(STEP_PIN, GPIO.HIGH)
        time.sleep(step_delay)
        GPIO.output(STEP_PIN, GPIO.LOW)
        time.sleep(step_delay)


#
# UI helpers
#

def draw_rect_bordered(surf, rect, fill, border, radius=6, border_w=1):
    pygame.draw.rect(surf, fill,   rect, border_radius=radius)
    pygame.draw.rect(surf, border, rect, border_w, border_radius=radius)


def draw_toggle(surf, rect, active, font, label, hover=False):
    fill   = C_BTN_ACT  if active else (C_BTN_HOV if hover else C_BTN)
```

```
    border = C_ACCENT   if active else C_BORDER
    draw_rect_bordered(surf, rect, fill, border, radius=5)
    dot_col = C_ACCENT if active else C_DIM
    dot_x = rect.right - 18
    dot_y = rect.centery
    pygame.draw.circle(surf, dot_col, (dot_x, dot_y), 5)
    if active:
        pygame.draw.circle(surf, C_ACCENT, (dot_x, dot_y), 3)
    t = font.render(label, True, C_TEXT if active else C_TEXTDIM)
    surf.blit(t, (rect.x + 10, rect.centery - t.get_height() // 2))



def draw_stepper(surf, rect, value, label, font, font_s,
                 btn_minus, btn_plus, hover_minus=False, hover_plus=False):
    """Ritar ett etikett + [] värde [+] på en rad."""
    lbl = font_s.render(label, True, C_TEXTDIM)
    surf.blit(lbl, (rect.x, rect.y + 2))

    bw = 26
    val_x = rect.x + 140

    #  knapp
    bm = pygame.Rect(val_x, rect.y, bw, rect.h)
    draw_rect_bordered(surf, bm,
                       C_BTN_HOV if hover_minus else C_BTN, C_BORDER, radius=4)
    m = font.render("", True, C_TEXT)
    surf.blit(m, m.get_rect(center=bm.center))
    btn_minus.update(bm)

    # värde
    val_surf = font.render(str(value), True, C_TEXT)
    vr = pygame.Rect(val_x + bw + 4, rect.y, 70, rect.h)
```

```
    draw_rect_bordered(surf, vr, C_PANEL, C_BORDER, radius=4)
    surf.blit(val_surf, val_surf.get_rect(center=vr.center))

    # + knapp
    bp = pygame.Rect(val_x + bw + 4 + 70 + 4, rect.y, bw, rect.h)
    draw_rect_bordered(surf, bp,
                       C_BTN_HOV if hover_plus else C_BTN, C_BORDER, radius=4)
    p = font.render("+", True, C_TEXT)
    surf.blit(p, p.get_rect(center=bp.center))
    btn_plus.update(bp)


class Rect:
    """Mutable rekt-wrapper för knappregioner."""
    def __init__(self):
        self._r = pygame.Rect(0, 0, 0, 0)

    def update(self, r):
        self._r = r

    def collidepoint(self, pos):
        return self._r.collidepoint(pos)


#
# PreviewWindow
#


class PreviewWindow:

    def __init__(self, base, indices, args):
```

```
        self.base      = base
        self.indices   = indices
        self.args      = args          # scan-inställningar (mutable via UI)
        self.cam_pos   = 0
        self.frame     = None
        self.status    = "Hämtar bild..."
        self._lock     = threading.Lock()
        self._stop_evt = threading.Event()

        #  Overlay-flaggor
        self.show_crosshair = True
        self.show_grid      = True
        self.show_thirds    = True

        #  Kamerainställningar (per kamera)
        n = len(indices)
        self.cam_settings = [
            {"iso": 0}
            for _ in range(n)
        ]

        #  pygame
        pygame.init()
        pygame.display.set_caption("Turntable Scanner")
        self.screen  = pygame.display.set_mode((WIN_W, WIN_H))
        self.font_l  = pygame.font.SysFont("monospace", 22, bold=True)
        self.font_m  = pygame.font.SysFont("monospace", 17, bold=True)
        self.font_s  = pygame.font.SysFont("monospace", 14)
        self.font_xs = pygame.font.SysFont("monospace", 12)

        #  Knapp-regioner (uppdateras varje frame)
        self._cam_btns      = [Rect() for _ in indices]
```

```
        self._tog_cross     = Rect()
        self._tog_grid      = Rect()
        self._tog_thirds    = Rect()
        # ISO
        self._iso_m = Rect(); self._iso_p = Rect()
        # Stopp
        self._stops_m = Rect(); self._stops_p = Rect()
        # Start-knapp
        self._start_btn = Rect()
        # Mappnamn (textinmatning)
        self.folder_name      = ""
        self._folder_active   = False
        self._folder_rect     = Rect()
        # Hover-state
        self._hover = None

    #  Kamera

    def _ci(self):
        return self.indices[self.cam_pos]

    def _cs(self):
        return self.cam_settings[self.cam_pos]

    def _label(self):
        return CAMERA_LABELS.get(self._ci(), str(self._ci()))

    #  Bakgrundstråd

    def _fetch_loop(self):
        tmp = Path(tempfile.mktemp(suffix=".jpg"))
        while not self._stop_evt.is_set():
```

```
            cam = self._ci()
            cs  = dict(self._cs())
            ok  = capture_preview(self.base, cam, tmp, timeout_ms=700,
                                  cam_settings=cs)
            if ok:
                try:
                    surf = pygame.image.load(str(tmp))
                    surf = pygame.transform.scale(surf, (PREVIEW_W, WIN_H))
                    with self._lock:
                        self.frame  = surf
                        self.status = f"Kamera {self._label()}  ·  live"
                except Exception as e:
                    with self._lock:
                        self.status = f"Bildfel: {e}"
            else:
                with self._lock:
                    self.status = f"Kamera {self._label()}  ·  ingen signal"
            self._stop_evt.wait(self.args.preview_ms / 1000)

    #  Overlay

    def _draw_overlay(self):
        ov = pygame.Surface((PREVIEW_W, WIN_H), pygame.SRCALPHA)
        cx, cy = PREVIEW_W // 2, WIN_H // 2

        if self.show_thirds:
            col = (255, 255, 255, 40)
            for x in [PREVIEW_W // 3, 2 * PREVIEW_W // 3]:
                pygame.draw.line(ov, col, (x, 0), (x, WIN_H), 1)
            for y in [WIN_H // 3, 2 * WIN_H // 3]:
                pygame.draw.line(ov, col, (0, y), (PREVIEW_W, y), 1)
```

```
    if self.show_grid:
        col = (255, 255, 255, 22)
        for x in range(0, PREVIEW_W, PREVIEW_W // 9):
            pygame.draw.line(ov, col, (x, 0), (x, WIN_H), 1)
        for y in range(0, WIN_H, WIN_H // 7):
            pygame.draw.line(ov, col, (0, y), (PREVIEW_W, y), 1)

    if self.show_crosshair:
        col = (255, 255, 255, 130)
        sz  = 28
        pygame.draw.line(ov, col, (cx - sz, cy), (cx + sz, cy), 1)
        pygame.draw.line(ov, col, (cx, cy - sz), (cx, cy + sz), 1)
        pygame.draw.circle(ov, col, (cx, cy), 7, 1)
        pygame.draw.circle(ov, (255, 255, 255, 50), (cx, cy), 40, 1)

    self.screen.blit(ov, (0, 0))

#  Sidopanel

def _draw_panel(self, mouse_pos):
    px = PREVIEW_W                    # panel start x
    pw = PANEL_W
    pygame.draw.rect(self.screen, C_PANEL,
                     pygame.Rect(px, 0, pw, WIN_H))
    pygame.draw.line(self.screen, C_BORDER, (px, 0), (px, WIN_H), 1)

    x  = px + 14
    rw = pw - 28                      # row width
    y  = 14

    def section(title):
        nonlocal y
```

```
        y += 6
        pygame.draw.rect(self.screen, C_PANEL2,
                         pygame.Rect(px, y, pw, 24))
        t = self.font_xs.render(title.upper(), True, C_DIM)
        self.screen.blit(t, (x, y + 5))
        y += 30

    def gap(n=8):
        nonlocal y
        y += n

    row_h = 32

    #  KAMERAVÄLJARE
    section("Kamera")
    bw = (rw - 6 * (len(self.indices) - 1)) // len(self.indices)
    for j, idx in enumerate(self.indices):
        lbl    = CAMERA_LABELS.get(idx, str(idx))
        active = (j == self.cam_pos)
        r      = pygame.Rect(x + j * (bw + 6), y, bw, row_h)
        hover  = r.collidepoint(mouse_pos)
        fill   = C_ACCENT2 if active else (C_BTN_HOV if hover else C_BTN)
        border = C_ACCENT2 if active else C_BORDER
        draw_rect_bordered(self.screen, r, fill, border, radius=6)
        tc     = C_BG   if active else C_TEXT
        t      = self.font_m.render(lbl, True, tc)
        self.screen.blit(t, t.get_rect(center=r.center))
        self._cam_btns[j].update(r)
    y += row_h
    gap()

    #  OVERLAYS
```

```
section("Overlay")
for (attr, label, ref) in [
    ("show_crosshair", "Kors",          self._tog_cross),
    ("show_thirds",    "Rule of thirds", self._tog_thirds),
    ("show_grid",      "Finrutnät",      self._tog_grid),
]:
    r     = pygame.Rect(x, y, rw, row_h - 4)
    hover = r.collidepoint(mouse_pos)
    draw_toggle(self.screen, r, getattr(self, attr),
                self.font_s, label, hover)
    ref.update(r)
    y += row_h

gap()

#  KAMERAINSTÄLLNINGAR
section("Kamerainst\xe4llningar")
cs = self._cs()

# ISO
iso_str = "Auto" if cs["iso"] == 0 else str(cs["iso"])
r = pygame.Rect(x, y, rw, row_h - 4)
draw_stepper(self.screen, r, iso_str, "ISO",
             self.font_m, self.font_s,
             self._iso_m, self._iso_p,
             self._iso_m.collidepoint(mouse_pos),
             self._iso_p.collidepoint(mouse_pos))
y += row_h
gap()

#  SCAN-INSTÄLLNINGAR
section("Scan-inst\xe4llningar")
```

```
r = pygame.Rect(x, y, rw, row_h - 4)
draw_stepper(self.screen, r, self.args.stops, "Antal stopp",
             self.font_m, self.font_s,
             self._stops_m, self._stops_p,
             self._stops_m.collidepoint(mouse_pos),
             self._stops_p.collidepoint(mouse_pos))
y += row_h
gap(6)

# Vinkel-info
deg = 360.0 / max(self.args.stops, 1)
info = self.font_s.render(f"{deg:.1f}\xb0 per stopp", True, C_DIM)
self.screen.blit(info, (x + 4, y))
y += 22
gap(10)

#  MAPPNAMN
section("Mappnamn")
lbl = self.font_s.render("Namn p\xe5 mapp:", True, C_TEXTDIM)
self.screen.blit(lbl, (x, y))
y += 20

field_r = pygame.Rect(x, y, rw, 34)
active  = self._folder_active
border_col = C_ACCENT if active else (C_BORDER if not field_r.collidepoint(mou
draw_rect_bordered(self.screen, field_r,
                   (35, 35, 50) if active else C_PANEL, border_col, radius=5)
display_name = self.folder_name + ("|" if active and int(time.time() * 2) % 2
if not display_name and not active:
    placeholder = self.font_s.render("(autogenererat)", True, C_DIM)
    self.screen.blit(placeholder, (field_r.x + 8, field_r.centery - placeholde
```

```
        else:
            ft = self.font_s.render(display_name, True, C_TEXT)
            self.screen.blit(ft, (field_r.x + 8, field_r.centery - ft.get_height() //
        self._folder_rect.update(field_r)
        y += 42
        gap(12)

        #  STATUS
        with self._lock:
            st = self.status
        info1 = self.font_xs.render(st, True, C_ACCENT)
        self.screen.blit(info1, (x, y))
        y += 20
        gap(4)

        #  START-KNAPP
        btn_h  = 46
        btn_r  = pygame.Rect(x, WIN_H - btn_h - 14, rw, btn_h)
        hover  = btn_r.collidepoint(mouse_pos)
        fill   = C_START_HOV if hover else C_START
        draw_rect_bordered(self.screen, btn_r, fill, C_ACCENT, radius=8, border_w=2)
        bt = self.font_l.render("  STARTA SCANNING", True, C_BG)
        self.screen.blit(bt, bt.get_rect(center=btn_r.center))
        self._start_btn.update(btn_r)

    #  Scanning-overlay

    def _draw_scan_overlay(self, msg):
        ov = pygame.Surface((PREVIEW_W, WIN_H), pygame.SRCALPHA)
        ov.fill((0, 0, 0, 150))
        self.screen.blit(ov, (0, 0))
        t = self.font_l.render(msg, True, C_WARN)
```

```
        self.screen.blit(t, t.get_rect(center=(PREVIEW_W // 2, WIN_H // 2)))

    #  Huvud-draw

    def _draw(self, mouse_pos=(0, 0), overlay_text=None):
        self.screen.fill(C_BG)

        # Kamerabild
        if self.frame:
            with self._lock:
                self.screen.blit(self.frame, (0, 0))
        else:
            pygame.draw.rect(self.screen, (20, 20, 28),
                             pygame.Rect(0, 0, PREVIEW_W, WIN_H))
            t = self.font_l.render("Väntar på kamera...", True, C_DIM)
            self.screen.blit(t, t.get_rect(center=(PREVIEW_W // 2, WIN_H // 2)))

        self._draw_overlay()

        if overlay_text:
            self._draw_scan_overlay(overlay_text)

        self._draw_panel(mouse_pos)
        pygame.display.flip()

    #  Knapptryckningar

    def _handle_click(self, pos):
        # Kameraväljare
        for j, btn in enumerate(self._cam_btns):
            if btn.collidepoint(pos):
                self.cam_pos = j
```

```
        return None

# Overlay-toggles
if self._tog_cross.collidepoint(pos):
    self.show_crosshair = not self.show_crosshair; return None
if self._tog_thirds.collidepoint(pos):
    self.show_thirds = not self.show_thirds; return None
if self._tog_grid.collidepoint(pos):
    self.show_grid = not self.show_grid; return None

ISO_STEPS = [0, 100, 200, 400, 800, 1600, 3200]

# Mappnamn-fält
if self._folder_rect.collidepoint(pos):
    self._folder_active = True
    return None

# Klick utanför fältet avaktiverar det
self._folder_active = False

# ISO
cs = self._cs()
if self._iso_m.collidepoint(pos):
    idx = ISO_STEPS.index(cs["iso"]) if cs["iso"] in ISO_STEPS else 0
    cs["iso"] = ISO_STEPS[max(0, idx - 1)]
elif self._iso_p.collidepoint(pos):
    idx = ISO_STEPS.index(cs["iso"]) if cs["iso"] in ISO_STEPS else 0
    cs["iso"] = ISO_STEPS[min(len(ISO_STEPS) - 1, idx + 1)]

# Stopp
elif self._stops_m.collidepoint(pos):
    self.args.stops = max(2, self.args.stops - 1)
```

```
        elif self._stops_p.collidepoint(pos):
            self.args.stops = min(72, self.args.stops + 1)

        # Start
        elif self._start_btn.collidepoint(pos):
            return "start"

        return None

    #  Event-loop

    def run(self):
        self._stop_evt.clear()
        t = threading.Thread(target=self._fetch_loop, daemon=True)
        t.start()

        clock  = pygame.time.Clock()
        mouse  = (0, 0)
        result = "quit"
        try:
            while True:
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        return "quit"
                    if event.type == pygame.MOUSEMOTION:
                        mouse = event.pos
                    if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
                        r = self._handle_click(event.pos)
                        if r == "start":
                            return "start"
                    if event.type == pygame.KEYDOWN:
                        if self._folder_active:
```

```
                        if event.key == pygame.K_ESCAPE:
                            self._folder_active = False
                        elif event.key == pygame.K_RETURN:
                            self._folder_active = False
                        elif event.key == pygame.K_BACKSPACE:
                            self.folder_name = self.folder_name[:-1]
                        else:
                            ch = event.unicode
                            # Tillåt bokstäver, siffror, bindestreck och understre
                            if ch and (ch.isalnum() or ch in "-_ "):
                                self.folder_name += ch
                    else:
                        if event.key in (pygame.K_ESCAPE, pygame.K_q):
                            return "quit"
                        if event.key == pygame.K_RETURN:
                            return "start"
                        if event.key == pygame.K_RIGHT:
                            self.cam_pos = (self.cam_pos + 1) % len(self.indices)
                        if event.key == pygame.K_LEFT:
                            self.cam_pos = (self.cam_pos - 1) % len(self.indices)
                        k = pygame.key.name(event.key).upper()
                        if k in LABEL_TO_INDEX and LABEL_TO_INDEX[k] in self.indi
                            self.cam_pos = self.indices.index(LABEL_TO_INDEX[k])

            self._draw(mouse_pos=mouse)
            clock.tick(30)
    finally:
        self._stop_evt.set()
        t.join(timeout=3)

def show_message(self, msg):
    self._draw(overlay_text=msg)
```

```
    def close(self):
        pygame.quit()


#
# Scanning-sekvens
#

def run_scan(args, base, indices, folder, preview):
    degrees_per_stop = 360.0 / args.stops
    cam_counters = {ci: 0 for ci in indices}
    total_ok = total_failed = 0
    stop = 0
    try:
        for stop in range(args.stops):
            ts = datetime.now().strftime("%H:%M:%S")
            print(f" Stopp {stop+1}/{args.stops}  [{ts}]  "
                  f"({stop * degrees_per_stop:.1f}°)")

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    raise KeyboardInterrupt

            preview.show_message(f"Stopp {stop+1}/{args.stops}  -  fotograferar...")
            pygame.display.flip()

            for i, ci in enumerate(indices):
                label = CAMERA_LABELS.get(ci, str(ci))
                cam_counters[ci] += 1
                outpath = folder / f"{label}{cam_counters[ci]}.jpg"
```

```
            # Hämta inställningar för denna kamera
            cam_pos_for_ci = preview.indices.index(ci)
            cs = dict(preview.cam_settings[cam_pos_for_ci])

            if i > 0:
                time.sleep(args.mux_settle / 1000)

            print(f"   Kamera {label}  ", end="", flush=True)
            ok, size_kb, attempts = capture_with_retry(
                base, ci, outpath, args.warmup,
                args.retries, args.retry_wait, cam_settings=cs
            )
            if ok:
                s = f"  (försök {attempts})" if attempts > 1 else ""
                print(f"  {outpath.name}  ({size_kb} KB){s}")
                total_ok += 1
            else:
                print(f"  MISSLYCKADES efter {attempts} försök")
                total_failed += 1

        if stop < args.stops - 1:
            print(f"   Roterar {degrees_per_stop:.1f}°...", end="", flush=True)
            preview.show_message(f"Roterar...  ({stop+1}/{args.stops})")
            pygame.display.flip()
            move_degrees(degrees_per_stop,
                         microsteps=args.microsteps,
                         step_delay=args.step_delay,
                         direction=True)
            print(f" klar.  Väntar {args.settle}s...")
            time.sleep(args.settle)
        print()
```

```
    except KeyboardInterrupt:
        print("\n[!] Avbruten.")
        raise

    return total_ok, total_failed, stop + 1


#
# Main
#

def main():
    parser = argparse.ArgumentParser(description="Turntable scanner med live-preview."
    parser.add_argument("--stops",       type=int,   default=8)
    parser.add_argument("--output-dir",  default=".")
    parser.add_argument("--warmup",      type=int,   default=200)
    parser.add_argument("--mux-settle",  type=int,   default=200)
    parser.add_argument("--retries",     type=int,   default=3)
    parser.add_argument("--retry-wait",  type=int,   default=1000)
    parser.add_argument("--step-delay",  type=float, default=0.001)
    parser.add_argument("--microsteps",  type=int,   default=8)
    parser.add_argument("--settle",      type=float, default=0.5)
    parser.add_argument("--preview-ms",  type=int,   default=500)
    args = parser.parse_args()

    print("=" * 58)
    print("  Turntable Scanner  ·  4× IMX477  ·  TMC2209 + NEMA 17")
    print("=" * 58)

    base = find_base()
    if not base:
        print("[!] rpicam/libcamera saknas. Kör: sudo apt install -y rpicam-apps")
```

```
    sys.exit(1)

indices = get_camera_indices(base)
if not indices:
    print("[!] Inga IMX477-kameror hittades.")
    sys.exit(1)

print(f"[] Kameror: {[CAMERA_LABELS.get(i,str(i)) for i in indices]}")
motor_setup()
preview = PreviewWindow(base, indices, args)

try:
    while True:
        action = preview.run()
        if action == "quit":
            break

        session_ts = datetime.now().strftime("%Y%m%d_%H%M%S")
        raw_name   = preview.folder_name.strip().replace(" ", "_")
        folder_name = f"{raw_name}_{session_ts}" if raw_name else f"scan_{session_
        folder = Path(args.output_dir) / folder_name
        folder.mkdir(parents=True, exist_ok=True)
        print(f"\n[] Sparar till: {folder.resolve()}\n")

        try:
            total_ok, total_failed, stops_done = run_scan(
                args, base, indices, folder, preview
            )
        except KeyboardInterrupt:
            print("[!] Scanning avbruten.")
            break
```

```
            # Återgå till startposition
            total_degrees = (360.0 / args.stops) * (args.stops - 1)
            if total_degrees > 0:
                print(f"[→] Återgår ({total_degrees:.1f}° bakåt)...")
                preview.show_message("Återgår till startposition...")
                pygame.display.flip()
                move_degrees(total_degrees,
                             microsteps=args.microsteps,
                             step_delay=args.step_delay,
                             direction=False)
                time.sleep(args.settle)
                print("[] Startposition nådd.")

            print()
            print("" * 58)
            print(f"  Stopp klara  : {stops_done}/{args.stops}")
            print(f"  Bilder tagna : {total_ok}   Misslyckade: {total_failed}")
            print(f"  Sparat till  : {folder.resolve()}")
            print("" * 58)
            print("\n[] Redo för nästa scanning.\n")

    except KeyboardInterrupt:
        print("\n[] Avslutas.")
    finally:
        motor_release()
        preview.close()
        print("[] Motor frikopplad.")


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

[16]

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige

www.chalmers.se

CHALMERS