



CHALMERS



Mjukvaruutveckling för pick-to-light station

Examensarbete inom högskoleprogrammet mekatronik

Robert Justh De Neczpal

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2026

www.chalmers.se

EXAMENSARBETE 2026

Mjukvaruutveckling för pick-to-light station

Robert Justh De Neczpal



CHALMERS

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2026

Mjukvaruutveckling för pick-to-light station
Robert Justh De Neczpal

© Robert Justh De Neczpal, 2026.

Handledare: Johan Karlsson, PK Data AB
Handledare: Hanna Ek, Chalmers Tekniska Högskola
Examinator: Jonas Almström Duregård, Chalmers Tekniska Högskola

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

Omslagsbild: Den slutgiltiga monteringsstationen.

Skriven i L^AT_EX
Göteborg 2026

Mjukvaruutveckling för pick-to-light station
Robert Justh De Neczpal
Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola

Sammanfattning

Elskåp används för att styra och skydda elektriska system inom industri och infrastruktur och består av ett stort antal elektriska komponenter. Den höga visuella likheten mellan komponenter ställer stora krav på noggrannhet vid plockning och montering av rätt komponent, och skapar behov av tekniska hjälpmedel vid produktion av dessa elskåp.

Detta examensarbete behandlar mjukvaruutveckling för en pick-to-light monteringsstation avsedd för att effektivisera och verifiera arbetet vid montering av elkomponenter på DIN-skenor. En ursprunglig prototyp skapad av PK Data låg till grund för arbetet, men saknade en fullständig mjukvaruarkitektur och verifieringsmetoder för praktisk användning.

Arbetet resulterade i två huvuddelar. Den första är ett webbaserat lagerkonfigurationsverktyg som möjliggör flexibel konfiguration av lagerlayout och komponentplacering för lokala lager vid monteringsstationer. Den andra är ett kamerabaserat verifieringssystem med You Only Look Once (YOLO)-modeller från Ultralytics, tillsammans med Optical Character Recognition (OCR)-modeller för identifiering av exakta komponentmodeller. I samband med verifiering med datorseende togs också en metod fram för strukturerad bildinsamling.

Modellerna utvärderades på begränsad hårdvara i form av en Raspberry Pi 4, där körning med specifikt modellformat visade möjlig realtidsverifiering. Däremot påverkades arbetet av tidsbegränsningar vilket gjorde att ett fullständigt verifieringsflöde inte hann implementeras.

Resultatet utgör därför ett proof of concept som visar att vision-baserad verifiering kompletterad med OCR fungerar som koncept. Fullständig integration av verifieringsflödet i realtid, samt vidareutveckling av hur man verifierar tätt monterade komponenter, återstår som nästa steg mot en produktionsnära lösning.

Nyckelord: pick-to-light, lagerkonfiguration, bildklassificering, YOLO, OCR, Raspberry Pi, verifiering, elskåp, DIN-skena, montering.

Förord

Detta arbete har genomförts som ett examensarbete inom högskoleingenjörsprogrammet mekatronik på Chalmers Tekniska Högskola.

Först och främst vill jag tacka min handledare från PK Data, Johan Karlsson, för möjligheten att genomföra mitt examensarbete med dem, samt för allt stöd och hjälp under projektets gång. Jag vill även rikta ett stort tack till John J. Camilleri som är den huvudansvarige för examensarbeten hos institutet för data- och informationsteknik för värdefull hjälp att hitta en lokal för prototypen och för allmänt stöd med examensarbetet. Ytterligare vill jag även tacka min handledare från Chalmers, Hanna Ek, som har varit en stor hjälp med strukturering av det akademiska arbetet. Slutligen vill jag tacka Jonas Almström Duregård som examinerade arbetet och gav slutlig feedback.

Robert Justh De Neczpal, Göteborg, Juni 2026

Innehåll

1	Inledning	1
1.1	Bakgrund	1
1.2	Syfte och Mål	1
1.3	Avgränsningar	2
2	Teknisk Bakgrund	3
2.1	Pick-to-light	3
2.2	Hårdvara	3
2.2.1	DIN-skena och elkomponenter	3
2.2.2	LED-list och adressering	4
2.2.3	Raspberry Pi och kamera	4
2.3	Mjukvara	4
2.3.1	Comma-Separated Values	4
2.3.2	AI och maskininlärning	5
2.3.2.1	YOLO	5
2.3.2.2	Träningsdatasets	5
2.3.2.3	Modellformat	5
2.3.2.4	Optical Character Recognition	5
2.3.2.5	Region of Interest	5
3	Förstudie	6
3.1	Analys av ursprunglig prototyp	6
3.1.1	Hårdvara	6
3.1.2	Mjukvara	7
3.2	Användarstudie	9
3.2.1	Företag 1	9
3.2.2	Företag 2	10
3.3	Begränsningar	10
4	Genomförande	12
4.1	Systemdesign	12
4.1.1	Val av kamera och positionering	12
4.2	Mjukvaruutveckling och implementering	17
4.2.1	Databas för DIN-artiklar	17
4.2.2	Lagerkonfiguration	17
4.2.2.1	Första versionen	18

4.2.2.2	Val av implementation	18
4.2.2.3	Uppbyggnad av lagerlayout	18
4.2.2.4	Komponentval och databasuppslagning	20
4.2.2.5	Generering av konfiguration	20
4.2.3	Dynamisk ROI-beräkning	21
4.2.3.1	Kalibrering	22
4.2.4	Teach Mode	24
4.2.5	Bilder för uppritning	25
4.2.6	Verifiering av montage	26
4.2.6.1	Val av vision-modell	27
4.2.6.2	Dataset och första modellträning	27
4.2.6.3	Justering av ROI och träningsdata	29
4.2.6.4	Slutlig träningsstrategi	31
4.2.6.5	Testmiljö	32
4.2.6.6	OCR för verifiering	33
5	Resultat	36
5.1	Lagerkonfigurationsverktyg	36
5.1.1	Stationsvy	36
5.1.2	Konfiguration av lagerzon	37
5.1.3	Komponentplacering	38
5.1.4	Sparring och LED-mappning	39
5.1.5	Användarflöde	40
5.2	Teach Mode och bildinsamling	40
5.3	Monteringsvy	42
5.3.1	Bakgrunds borttagning	42
5.4	Verifiering av montage	43
5.4.1	Testade klassificeringsmodeller	43
5.4.2	OCR Komplettering	45
6	Diskussion	48
6.1	Diskussion av resultat	48
6.2	Begränsningar	48
6.3	Etiska aspekter	49
6.4	Licens och kommersiell användning	49
6.5	För vidare arbete	49
7	Slutsats	51
	Bibliography	51
A	Träningskript för slutlig modell	II
B	Exempel på sparad lagerkonfiguration	III
C	Modellresultat på Pi 4 med PyTorch format	VI
D	Modellresultat på Pi 4 med NCNN format	VII

1

Inledning

Elskåp används för att styra och skydda elektriska system inom industri och infrastruktur. Dessa elskåp består av en strukturerad placering utav särskilda elektriska komponenter som monteras på DIN-skenor. DIN-skenan är en standardiserad metallskena som används för att fästa elektriska komponenter. Vid monteringen av elskåp fäster operatören komponenter på DIN-skenan utifrån en specifik utskriven ritning och komponentlista.

Ett elskåp innehåller typiskt ett stort antal komponenter som kan vara visuellt lika varandra, vilket ställer ett högt krav på att operatören är noggrann vid tolkning av vilken komponent som ska monteras, och vid själva monteringen. Trots detta sker monteringsarbetet till stor del manuellt, vilket gör processen beroende av operatörens individuella erfarenhet. Detta skapar behov av tekniska lösningar som stödjer operatörer i monteringsprocessen samt verifiering av korrekt montage.

1.1 Bakgrund

PK Data utvecklar elritningsprogram som används för att automatisera dokumentation och skapa produktionsunderlag för centraler, ställverk och apparatskåp. PK Data har skapat en prototyp för en modulär pick-to-light station med funktionen att montera elkomponenter längs DIN-skenor. Pick-to-light är ett system som används i lager för att underlätta och öka noggrannheten vid plockning av varor med hjälp av ljussignaler som indikerar för operatören vilken komponent som ska plockas.

Stationen har en skärm som visar vad som ska monteras, en lasersensor som mäter det monterade avståndet av komponenter längs DIN-skenan, en LED-list som visar nästa komponent som ska monteras från lagerhyllan och en kamera för att verifiera korrekt montage.

Den befintliga prototypen utgör grundläggande proof-of-concept som saknar en fullständig mjukvaruarkitektur för praktisk användning vid montering av elkomponenter i produktionsmiljö.

1.2 Syfte och Mål

Syftet med examensarbetet är att vidareutveckla den befintliga pick-to-light stationen för att möjliggöra en enklare och mer tillförlitlig montering av elkomponenter

genom att tydligt visa operatören vilken komponent som ska monteras näst och möjliggöra kontroll av att det är rätt komponenter som har monterats.

Målet är att implementera och utvärdera de centrala funktionerna för pick-to-light monteringsstationen som ett proof of concept, för att visa den tekniska genomförbarheten och skapa en grund för vidare produktutveckling. Detta uppnås genom att:

- Utveckla ett lagerkonfigurationsverktyg som möjliggör flexibel konfiguration av lagerlayout, komponentplacering och LED-mappning för pick-to-light.
- Implementera och utvärdera kamerabaserat verifieringssystem med YOLO bildklassificering för identifiering av monterade komponenter på DIN-skena.
- Utvärdera OCR som komplement till klassificeringsmodellen för att särskilja visuellt liknande komponenter inom samma komponentfamilj.

1.3 Avgränsningar

För att se till att arbetet inte överskrider resurs- eller tidsbegränsningar så görs följande avgränsningar:

- Konstruktion av station ingår inte.
- Fullständig integration av verifieringsflödet i realtid ingår inte.
- Arbetet utgör inte en produktionsklar lösning, utan syftar till att visa teknisk genomförbarhet med prototypen.

2

Teknisk Bakgrund

Detta kapitel presenterar de komponenter, teoretiska koncept och den mjukvaruar-kitektur som ligger till grund för projektet.

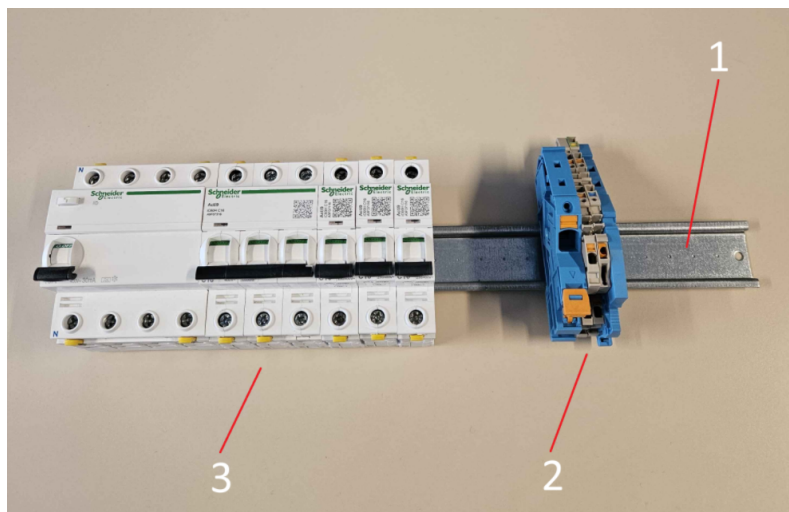
2.1 Pick-to-light

Pick-to-light är ett system som används i lager- och produktionsmiljöer för att guide operatörer med ljussignaler vid plockning eller montering av varor. Systemet indikerar med hjälp av ljus var nästa artikel finns eller ska placeras, vilket minskar risken för fel och ökar effektiviteten i manuella flöden. Pick-to-light gör det möjligt för en operatör att fokusera på själva monteringsuppgiften utan att slösa tid på att leta efter nödvändiga artiklar. Det är ett system som överlag förbättrar monteringsprocessen och minskar den kognitiva belastningen för operatörer, men kan kräva att det finns distinkta system i bakgrunden som håller koll på vad som ska plockas näst för att göra det till en smidig upplevelse [1].

2.2 Hårdvara

2.2.1 DIN-skena och elkomponenter

En DIN-skena är en standardiserad metallskena som används för montering av elektriska komponenter. Standarden möjliggör enkel montering av olika tillverkare inom Europa. DIN-skenans syfte är att bidra till en flexibel uppbyggnad av elsystem. Vanliga komponenter som kan fästas på DIN-skenor är bland annat dvärgbrytare, jordfelsbrytare och plintar.



Figur 2.1: Bild på 1) DIN-skena, 2) plintar och 3) brytare.

2.2.2 LED-list och adressering

En adresserbar LED-list består av individuellt styrda lysdioder där varje diod kan aktiveras separat med en specifik färg och ljusstyrka. Varje diod har en unik adress i listan, vilket gör det möjligt att tända enstaka segment utan att påverka övriga dioder. Vid stationen används en WS2812B-baserad LED-list. WS2812B är ett vanligt förekommande protokoll för adresserbara LED-lister där styrdata och strömförsörjning överförs via en enda signalledare [2].

2.2.3 Raspberry Pi och kamera

Raspberry Pi är en serie av kompakta enkortsdatorer. Det finns olika varianter beroende på vad man har för behov gällande processor, arbetsminne och anslutningsmöjligheter. Generellt för de olika modellerna är stöd för nätverkskommunikation, GPIO (General purpose input/output) och USB-anslutningar. De nyare modellerna är även utrustade med stöd för bluetooth och Wi-Fi för att möjliggöra trådlös uppkoppling. Operativsystemet som körs på Raspberry Pi:s är oftast baserat på Linux [3].

Raspberry Pi-kameror ansluts via Camera Serial Interface (CSI) direkt till Raspberry Pi enheter. Det finns olika kameramodeller som exempelvis Raspberry Pi AI, HQ och standardkamera. Modellerna skiljer sig i upplösning och optiska egenskaper, där vissa också stödjer utbytbara objektiv [3].

2.3 Mjukvara

2.3.1 Comma-Separated Values

Comma-Separated Values (CSV) är ett enkelt textbaserat format för att lagra tabelldata där värdena separeras med kommatecken [4].

2.3.2 AI och maskininlärning

Maskininlärning är en gren inom artificiell intelligens där modeller tränas på data för att känna igen mönster och göra förutsägelser. Inom bildanalys används djupinlärning, där neurala nätverk tränas på stora mängder bilder för att lära sig särskilja olika klasser. En viktig distinktion är skillnaden mellan bildklassificering, som avgör vilken klass ett objekt tillhör i en bild, och objekt-detektering, som även lokaliserar objektet i bilden [5].

2.3.2.1 YOLO

You Only Look Once (YOLO) är ett ramverk för datorseende som ursprungligen utvecklades för objekt-detektering men även stödjer bildklassificering. Som det låter bygger YOLO på att bilder hanteras på en gång, snarare än i flera steg. Ramverket erbjuder modeller i olika storlekar för olika behov av precision och beräkningshastighet [6].

2.3.2.2 Träningsdatasets

Ett träningsdataset består av märkt data som används för att träna en maskininlärningsmodell. Datasetet delas vanligtvis upp i träningsdata och valideringsdata, där träningsdatan används för att optimera modellens vikter och valideringsdatan används för att utvärdera modellens prestanda på data den inte har sett under träningen [7]. Augmentation är en teknik där träningsbilderna varieras slumpmässigt, exempelvis genom rotation eller ljusförändring, för att göra modellen mer robust mot variationer i verklig data [8].

2.3.2.3 Modellformat

Maskininlärningsmodeller tränas vanligtvis i ramverket PyTorch, vilket gör det till ett standardformat vid export [9]. För körning av modeller på enklare system kan de exporteras till format optimerade för specifik hårdvara. NCNN är ett sådant inferensramverk optimerat för ARM-baserade system utan GPU, som exempelvis Raspberry Pi [10].

2.3.2.4 Optical Character Recognition

Optical Character Recognition (OCR) är en teknik för att tolka och extrahera text från bilder. Det används inom en mängd olika tillämpningar såsom dokumentdigitalisering och automatisk textigenkänning. Moderna OCR-modeller använder djupinlärning och kan hantera text i olika typsnitt, storlekar och orientering [11].

2.3.2.5 Region of Interest

En Region of Interest (ROI) är ett avgränsat område i en bild som används för analys. Det kan användas för olika syften som exempelvis för att isolera objekt för att antingen spara på beräkningskraft eller öka bildkvaliteten.

3

Förstudie

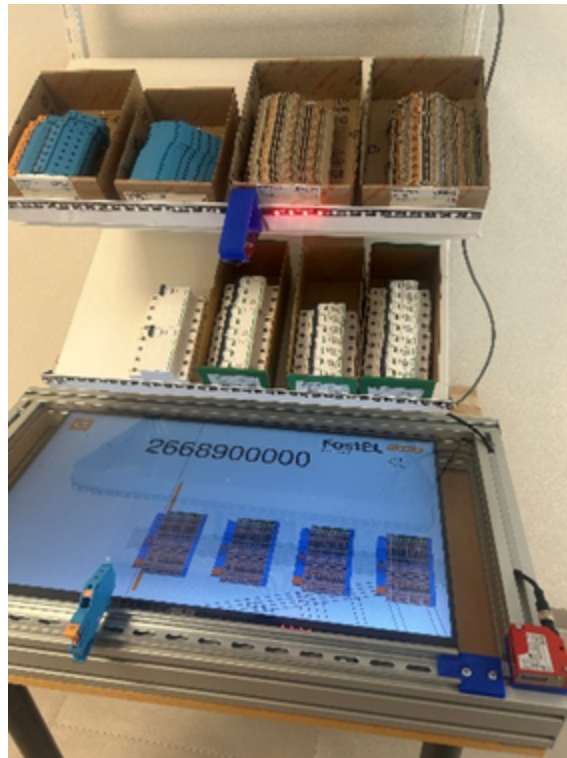
Detta kapitel presenterar utgångspunkten för arbetet genom en analys av den ursprungliga prototypen och en användarstudie hos kunder. Tillsammans ger det en bild av förutsättningarna som låg till grund för behoven och kraven som användes i det fortsatta utvecklingsarbetet.

3.1 Analys av ursprunglig prototyp

Den ursprungliga prototypen analyserades för att skapa en förståelse för systemets uppbyggnad och själva tanken bakom produkten överlag. Analysen baserades på den ursprungliga systemdokumentationen och källkoden.

3.1.1 Hårdvara

Den ursprungliga prototypen bestod av en ram med fyra stycken 40x80 millimeter aluminiumprofiler som bas med 2 stycken 800 millimeter fram och bak, samt 2 stycken 380 millimeter på sidorna. Inuti ramen låg en 27 tums skärm med 1920x1080 pixelupplösning med en plastskiva ovanpå som skydd. På höger sida av framsidan fanns ett fäste för DIN-skenor som möjliggjorde att skenan fästes parallellt med skärmen och placeras i linje med skärmens högra bildkant. Till sidan av fästet fanns en optisk lasersensor monterad där lasern var riktad längs DIN-skenan. Bakom den bakre ramen fanns en hylla för lådor med elkomponenter. På varje hyllrad var en LED-strip monterad riktad mot operatören. I mitten av den översta av två hyllor fanns också ett fäste för en Raspberry Pi AI-kamera som satt installerad och var riktad snett nedåt mot DIN-skenan. Som dator för beräkningar och styrning användes en Raspberry Pi 4 som var direkt kopplad till LED-listen, kameran, skärmen och till lasersensorn genom en Raspberry Pi kompatibel Input/Output Link(IOL)-HAT. Ytterligare fanns en handhållen streckkodsläsare kopplad direkt till Raspberry Pi 4:an. Till sist fanns en snabbspännare monterad på vänster sida av den främre ramen för att fästa DIN-skenor vid montering.



Figur 3.1: Bild på den ursprungliga prototypen.

3.1.2 Mjukvara

Den ursprungliga mjukvaran var utvecklad i Python och kördes på Raspberry Pi 4:an. Systemet var uppbyggt modulärt där separata script ansvarade för olika funktioner såsom användargränssnitt, positionsläsning, LED-styrning, och avläsning av QR-kod. Hela systemets logik baserades på att det lästes in en ritningsfil som genereras för varje DIN-skena i samband när man skapar sitt elsystem i PK Datas CAD-program. I ritningsfilen ingår var komponenten sitter på DIN-skenan från vänster till höger angett i start- och slutposition i millimeter. Nedan presenteras ett exempel på hur en ritningsfil kan se ut.

DIN = 486.000			
10.000	82.000	92.000	A9Z21440
82.000	136.000	92.000	A9F07316
136.000	154.000	92.000	A9F07110
154.000	172.000	92.000	A9F07110
172.000	190.000	92.000	A9F07110
190.000	208.000	92.000	A9F07116
208.000	226.000	92.000	A9F07110
226.000	244.000	92.000	A9F07110

Det första som framkommer i ritningen är längden på DIN-skenan i millimeter. Formatets struktur sammanfattas på en överskådligt sätt i Tabell 3.1.

Tabell 3.1: Ritningsfil format.

Startposition	Slutposition	Höjd	Artikelnummer
---------------	--------------	------	---------------

Där *Slutposition* – *Startposition* direkt motsvarar komponentens bredd och höjden är måttet på dess höjd sett ovanifrån i 2D. Denna representation är typisk för komponenter som monteras på DIN-skenor.

Gränssnittet var implementerat i Pygame och användes för att visuellt presentera ritningen på skärmen. Systemet visualiserade DIN-skenan, komponenternas placering baserat på den inlästa ritningsdatan samt skrev upp nästa komponent som skulle monteras. För att rita i verklig skala användes en skalningsfunktion som konverterade millimeter till pixlar. Detta gjorde det möjligt att översätta fysiska positioner till grafiska koordinater. DIN-skenan var uppritad med Pygames inbyggda ritningsfunktioner medan komponenterna representerades med bildfiler placerade enligt respektive koordinat i ritningsdatan. Dessa bilder lagrades lokalt och hämtades ursprungligen från tillverkarens mediabilder eller från motsvarande CAD-ritning som fanns publicerad. På så sätt skapades en visuell representation av den verkliga monteringslayouten, vilket gjorde det möjligt för operatören att jämföra den fysiska uppbyggnaden med den digitala visualiseringen.

Positionsläsningen hanterades via lasersensorn som var kopplad via en IOL-HAT. Sensorn läste kontinuerligt av den aktuella positionen längs DIN-skenan och returnerade ett avståndsvärde i millimeter som representerade hur mycket som var monterat längs DIN-skenan, vilket jämfördes mot komponenternas position i ritningsfilen för att avgöra nästa komponent att montera. Det returnerade avståndsvärdet användes också för att visuellt visa på skärmen var man aktuellt befinner sig längs DIN-skenan.

LED-styrningen implementerades genom HTTP-anrop till en lokal Flask server som ansvarade för kommunikationen med den adresserbara LED-listen. Från huvudprogrammet skickades kommandon till servern för att aktivera specifika positioner på lagerhyllan motsvarande lådan som innehöll nästa komponent att montera. Servern var baserad på Python biblioteket NeoPixel för att tillåta styrning av LED-listen direkt via Raspberry Pi:s GPIO-pinnar. Funktionen användes som grund till pick-to-light systemet genom att guida operatören visuellt till rätt komponentlåda innan montering på DIN-skenan. Varje artikelnummer var kopplat till ett fördefinierat startindex på LED-listen samt antalet efterföljande LEDs som skulle tändas. Till samma position kopplades även streckkoden för komponentlådan och i vissa fall även streckkoden som fanns på komponenten. På så sätt kunde rätt lådposition tändas upp vid behov. Dessa positioner var fördefinierade i en textfil som programmet läste från. Systemet kunde även styra vilken färg segmentet skulle lysa med, vilket användes för att skilja mellan guidning under exempelvis montering eller om man skannade streckkoden på en komponentlåda för att veta dess position.

Avläsning av QR-kod användes som verifikation av rätt montage för elkomponenterna som hade en avläsbar QR-kod på ovansidan som var synlig för Raspberry Pi AI-kameran. Bildbehandlingen genomfördes med hjälp av OpenCV där bilden analyserades för att lokalisera och tolka QR-koder. Utifrån komponentens position

i ritningsfilen samt information om att komponenten förväntades innehålla en QR-kod beräknades ett förväntat område i kamerabilden där QR-koden skulle befinna sig. Det identifierade områden transformerades därefter för att kompensera för kamerans perspektiv och göra QR-koden lättare att avläsa. Det avlästa innehållet i QR-koden analyserades därefter för att extrahera komponentens artikelnummer och jämföra mot det förväntade artikelnummret enligt ritningen.

3.2 Användarstudie

En annan viktig del av förstudien var en användarstudie och fabriksbesök hos kund. Detta genomfördes för att ge en inblick hur det i praktiken såg ut vid monteringsstationer för elkomponenter. Under besöket studerades hur operatörerna arbetar, vilka moment i monteringsprocessen som fungerar bra och vad det finns för förbättringspotential. Det kritiska att undersöka för projektet var just vad användarna efterfrågar av systemet, samt få en bild av hur lageruppsättningen kan se ut.

Följande frågor ställdes huvudsakligen under besöket:

- Hur jobbar ni idag med att plocka material till ett skåp?
- Hur ser möjligheterna för en anpassning av lagersystemet ut?
- Vad är det som tar mest tid vid montering på DIN-skenor idag?
- Vad krävs för att systemet ska vara lätt att använda?

Frågorna hjälpte mycket i att konkret få höra från montörer som arbetar med bland annat montering av elkomponenter på DIN-skenor dagligen, samt var det viktigt att få uppleva miljön och observera mycket genom att bara vara på plats och se det framför sig. Det besöktes två stycken fabriker som beskrivs nedan med Företag 1 och Företag 2 för att hålla det konfidentiellt.

3.2.1 Företag 1

Företag 1 var en mindre produktionsmiljö där material till varje elskåp levererades projektvis på separata pallar tillsammans med övriga komponenter för ordern. Montering genomfördes direkt i elskåpet där DIN-skenan först installerades och därefter monterades komponenterna på plats. Operatören arbetade utifrån en utskrivna ritning som bland annat innehöll artikelnummer och komponenternas placering längs DIN-skenan. Utifrån ritningen hämtades rätt komponentlådor från den levererade pallan och monterades manuellt i rätt ordning.

Vid diskussion med operatörerna framkom att verifiering av rätt komponent var särskilt viktigt, samtidigt som arbetsflödet behövde vara snabbt och ergonomiskt. De var öppna för förslaget att lägga till en separat monteringsstation i flödet om det inte bidrog med tidsfördröjningar. Gällande lagersystem var det diskuterat att till en början ha vanligt förekommande komponenter permanent placerade på hyllor vid monteringsstationen, medan mer ovanliga eller projektspecifika komponenter fortsatt levererades separat för respektive order.

3.2.2 Företag 2

Företag 2 var en större produktionsmiljö där flera elskåpsprojekt pågick parallellt och lagerhanteringen var mer strukturerad. Material hämtades från ett centralt lager och sorterades i individuella lådor för respektive elskåp innan montering påbörjades. Operatören arbetade utifrån ritningar som innehöll information om komponentplacering, artikelnummer och plintrader, där materialet plockades och monterades enligt specifikation.

För vissa typer av elskåp där plintmontering på DIN-skena förekom i större omfattning, användes separata monteringsstationer där DIN-skenor byggdes färdigt innan de monterades i själva elskåpet. Operatören arbetade då vid en station med ritning och komponentlådor placerade på hyllor runt arbetsytan, främst framför sig, vilket gjorde arbetsflödet mer likt den ursprungliga prototypen i projektet.

Vid diskussion med operatörerna framkom att ett tydligt pick-to-light system och snabb verifiering av rätt komponent var viktiga funktioner. Även om monteringsflödet redan var strukturerat behövde operatören fortfarande kontrollera artikelnummer manuellt mot ritningen, vilket skapade onödiga avbrott i arbetsflödet. Ett system med snabb bekräftelse och kort inlärningstid ansågs därför vara värdefullt.

3.3 Begränsningar

Baserat på analysen av den ursprungliga prototypen samt användarstudien identifierades följande begränsningar och förbättringsområden:

Hårdvara

- Lasersensorn var kostsam och känslig för störningar, exempelvis om operatörens hand befann sig i vägen. Den mätte endast absolut avstånd i en dimension längs DIN-skenan, vilket inte bidrog med meningsfull verifiering av vad som faktiskt hade monterats. Ytterligare problem uppstod vid mätning av vissa plintar med oregelbunden textur på sidan, vilket medförde att det uppmätta avståndet tekniskt sett hamnade inuti komponenten.
- Systemet hade endast en kamera med begränsad synvinkel, användbar enbart för avläsning av QR-kod på de få komponenter där sådan fanns placerad på ovansidan. Metoden var beroende av att komponenttillverkare använde QR-koder på detta sätt, vilket i praktiken är ovanligt och utgör en osäkerhet. Kameran var dessutom monterad på lagerhyllan, vilket egentligen inte ingår som en del av stationen utan tillhörde kundens befintliga miljö. Detta medförde att lösningen blev beroende av kundens fysiska uppsättning, såsom hyllans placering, höjd och kamerans vinkel, vilket försvårade en generell implementation hos olika kunder.

Konfiguration

- Komponentdata och LED-mappning lagrades i hårdkodade textfiler, vilket gjorde systemet svårt att underhålla och uppdatera. Förändringar i artikel-

nummer, komponentlådaplacering eller LED-segment krävde manuell redigering av filer direkt i systemet. Nya komponenter kunde inte heller läggas till utan att manuellt definiera deras koppling till artikelnummer och streckkod samt kalibrera vilket område på LED-listen som skulle representera komponentlådan. Detta gjorde förändringar tidskrävande att implementera.

- Systemet var uppbyggt utifrån den ursprungliga prototypens specifika lageruppsättning där LED-segment, hyllplaceringar och komponentlådor var fördefinierade för just denna station. Lösningen saknade stöd för att enkelt konfigurera om lagerlayouten för andra arbetsmiljöer eller framtida implementation hos kund. Detta begränsade systemets skalbarhet.

Mjukvara

- Systemet bestod av flera separata script för funktioner såsom GUI, LED-styrning, QR-avläsning och sensorsystem. Kommunikation mellan delar av systemet skedde delvis via textfiler och lokala HTTP-anrop, vilket gjorde strukturen svår att överblicka och försvårade underhåll av systemet.
- Mjukvaran var inte anpassad för fjärruppdatering, vilket innebar att förändringar och underhåll krävde fysisk åtkomst. Vid implementation hos kund skulle detta försvåra support och göra uppdateringar mer tidskrävande.

Verifiering

- QR-baserad verifiering täckte endast ett fåtal komponenter med QR-kod på ovansidan, vilket innebar att majoriteten av komponenterna saknade verifiering av korrekt montage.
- Komponenter av samma typ kan vara visuellt identiska med samma fysiska mått men olika specifikationer, vilket inte gick att särskilja i den ursprungliga lösningen utan en mer avancerad bildanalys.

4

Genomförande

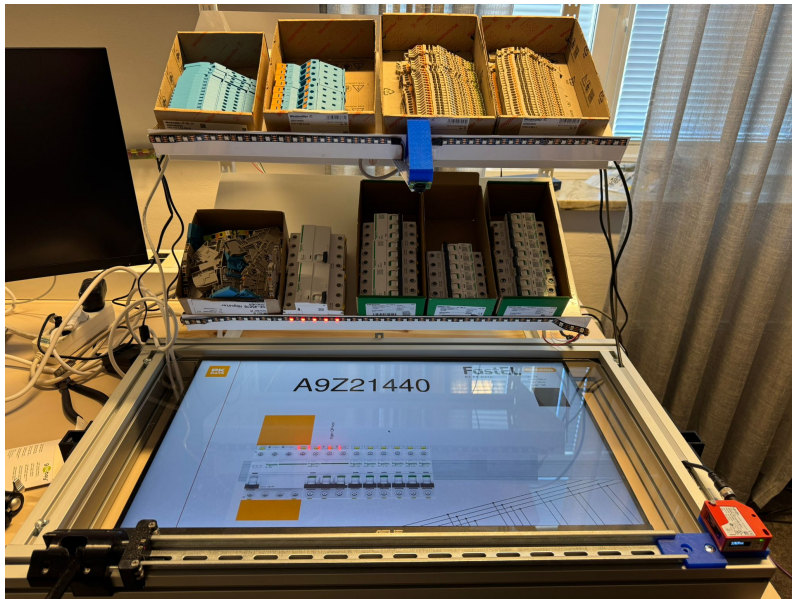
Detta kapitel presenterar det utvecklingsarbete som har genomförts under projektets gång utifrån de förutsättningar och behov som framkom under förstudien. Fokus ligger på tolkning av problemet, tankegångar för lösningar och metodik vid utveckling.

4.1 Systemdesign

Den ursprungliga prototypens hårdvara reviderades under arbetets gång där konstruktionsmässiga förändringar såsom nya fästen och monteringslösningar togs fram i ett parallellt examensarbete. En central del av förändringarna var hur kameror integrerades i stationen, vilket även påverkade arbetet inom mjukvaruutvecklingen. I kapitel 4.1.1 beskrivs val av kameran specifikation och positionering på stationen.

4.1.1 Val av kamera och positionering

I den ursprungliga prototypen var kameran monterad på lagerhyllan bakom stationen, riktad snett nedåt mot DIN-skenan. Som konstaterades i förstudien innebar detta att kameran i sig inte var en integrerad del av stationen utan var beroende av kundens befintliga hylluppsättning, vilket försvårade en generell implementation hos olika kunder. Ytterligare framkom det under arbetets gång att målet skiftade mot en implementation av verifiering med endast kameror genom AI-baserade vision-modeller, vilket stärkte behovet av att behålla kameran men positioneringen behövde ändras.



Figur 4.1: Ursprunglig kameraposition.

Det första steget var därför att flytta den befintliga Raspberry Pi AI-kameran till sidan av stationen på en egen ram, vilket gjorde att den blev en del av stationens egen konstruktion. För att minska risker för störningar under verifiering i form av armar eller händer i vägen för bilden, införskaffades ytterligare en kamera av samma modell så att det fanns två kameror från respektive sida av stationen. Eftersom det fanns två kameror som var separerade från Raspberry Pi 4 modulen, användes en Raspberry Pi Zero 2W för respektive kameranod, som kommunicerade med Pi 4:an genom att använda USB On-The-Go (OTG) och en gadget-drivrutin för att skapa nätverksanslutning över USB-kabeln.



Figur 4.2: Kameror monterade vid sidan av stationen.

Det visade sig dock att bildkvaliteten med tanke på synvinkeln man fick på komponenterna från sidovyn inte var tillräcklig för att kunna avläsa text på vissa minibrytare som fanns vid stationen. Detta gjorde det även komplicerat att verifiera att komponenter sitter monterade vid rätt avstånd längs skenan eftersom det hade krävt precisa uträkningar i ett 3D-rum. Dessutom ansågs det betydelselöst att köra verifiering endast baserat på form när många komponentfamiljer är snarlika visuellt, och skiljer sig endast åt med text på ovansidan vilket var oläsligt med den nuvarande uppsättning av kameror.

Detta ledde till beslutet att placera kameran ovanifrån för att få en top-down vy av DIN-skenan, vilket också förenklade positionsberäkningar till ett tvådimensionellt problem jämfört med den mer komplexa geometrin från sidovyn.

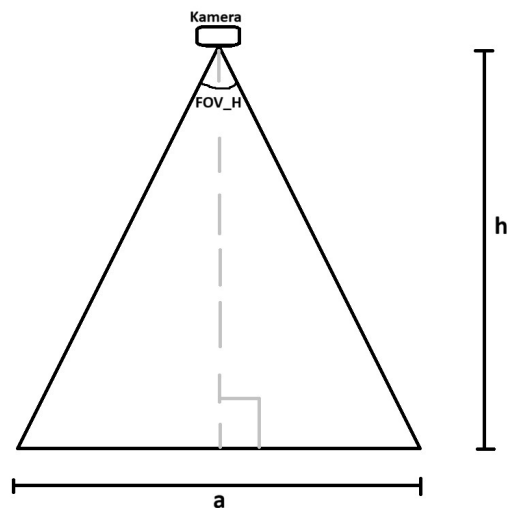


Figur 4.3: Kameran positionerad ovanifrån.

För att bestämma vilken höjd kameran behövde monteras för att en hel 600 mm lång DIN-skena skulle vara synlig beräknades höjden utifrån

$$h = \frac{\frac{a}{2}}{\tan(\frac{FOV_H}{2})}, \quad (4.1)$$

där h är den sökta höjden, a är DIN-skenans längd som ska vara synlig (600 mm) och FOV_H är kamerans horisontella synfältsvinkel som gällande Raspberry Pi kameror kan hämtas direkt från produktbladet. För Raspberry Pi AI-kameran gav detta en ungefärlig höjd på 460 mm som kameran behöver vara ovanför DIN-skenan. Den geometriska uppställningen för denna beräkning illustreras i Figur 4.4.



Figur 4.4: Illustration av kamerans horisontella synfält.

Vid test av bildkvaliteten konstaterades att Raspberry Pi AI-kameran inte gav tillräcklig upplösning för att kunna avläsa den text som skiljer snarlika komponenter åt på den höjd som krävdes för att få hela DIN-skenan i bild. Samtidigt var höjden vid gränsen för vad som är praktiskt utan att kameran är i vägen för operatören, vilket innebär att höjden inte kunde sänkas för att uppnå tillräcklig bildkvalitet för avläsning av de små texterna.

För att lösa detta införskaffades en Raspberry Pi HQ-kamera med en 16 mm lins, vilket gav möjligheten att effektivt zooma in på DIN-skenan och uppnå tillräcklig bildkvalitet på en höjd där kameran inte är i vägen. En avvägning gjordes dock mellan bildkvalitet och synfält, då den valda linsen och monteringshöjden resulterade i att cirka 400 mm av DIN-skenan är synlig i bilden istället för hela 600 mm. Beslutet togs därför att avgränsa systemet till en kamera som täcker 400 mm, med möjligheten att senare kunna utöka till två kameror för att täcka en hel 600 mm DIN-skena.



Figur 4.5: Bilder på den slutgiltiga kamerauppsättningen.

4.2 Mjukvaruutveckling och implementering

4.2.1 Databas för DIN-artiklar

PK Data delade med sig av en Excel-fil med alla elkomponenter som är kompatibla med DIN-skenor. Excel-filen innehöll fälten E-nummer, fabrikat, artikelnummer, beskrivning, typ, data, höjd och bredd per komponent. E-nummer är en unik sju-siffrig kod som fungerar för att identifiera elektriska produkter i Sverige [12]. Beskrivning används för att beskriva komponentkategorin. Typ och data används för att beskriva komponentens egenskaper, såsom karakteristik, antal poler eller vilken ström den är klassad för.

Excel-filen konverterades först till CSV-format och importerades sedan till en SQLite-databas med hjälp av ett Python script. SQLite valdes eftersom formatet är filbaserat och inte kräver en separat databasserver, vilket gjorde det enkelt att använda databasen för att snabbt söka upp relevant information om elkomponenter vid behov.

4.2.2 Lagerkonfiguration

Som det presenterades i förstudien var den ursprungliga prototypens lager- och LED-konfiguration starkt kopplad till den specifika uppbyggnaden av konceptstationen. För att minska behovet av manuell redigering av konfigurationsfiler utvecklades ett separat lagerkonfigurationsverktyg. Verktøjets roll är att fungera som ett setup läge där stationens lagerlayout, komponentplaceringar och pick-to-light information kan definieras inför att stationen tas i bruk.

Lagerkonfigurationsverktyget avgränsades till att hantera endast en stationslayout åt gången, där lagret som definieras är lokalt för en specifik station och inte kopplat

till ett övergripande lagerhanteringssystem.

4.2.2.1 Första versionen

En första version av lagerkonfigurationsverktyget utvecklades i Pygame för att snabbt kunna testa hur lagerlayout och komponentplacering kunde visualiseras på Raspberry Pi:n eftersom den ursprungliga prototypen redan använde Python och Pygame för sitt gränssnitt. Verktöget presenterade visuellt lagret som ett rutnät av lådor där man till en början skrev in sin lagerlayout i antalet rader och kolumner vilket genererade ett rutnät, där man efteråt manuellt kunde expandera med lådor. När man var färdig med att bygga upp sitt lager, kunde man därefter allokera komponenter baserat på artikelnummer till vardera låda. Artikelnumren lästes in som unika komponenter från ritningsfiler som fanns sparade som textfiler.

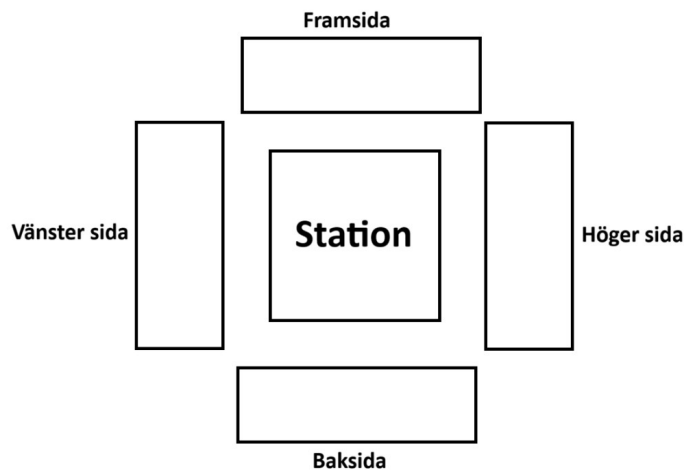
Denna versionen presenterades för handledare på PK Data för att få tidig feedback på konceptet. Feedbacken som framkom var att verktyget upplevdes som för flexibelt eftersom den utvecklades i tanke med att kunna köras på Raspberry Pi:n i samband med andra funktioner som använder samma gränssnittsbibliotek. Det betonades att konfigurationen skulle vara fast och definieras i ett separat setup-läge, skilt från själva monteringsflödet. Dessutom påpekades att ett rutnätsbaserat gränssnitt inte var lämpligt eftersom komponentlådor varierar i bredd, vilket gav en missvisande visuell representation och kunde inte möjliggöra beräkning av LED-segment för komponentlådorna. Det framkom också att stationer inte enbart har lagerhyllor framför sig, beroende på arbetsmiljö kan hyllor finnas på flera sidor om operatören, och inte alla hyllor behöver nödvändigtvis ha pick-to-light. Komponenter som är ovanliga eller projektspecifika kan behövas levereras separat per order och finns då inte på en fast hyllplats.

4.2.2.2 Val av implementation

Den slutliga versionen av lagerkonfigurationsverktyget implementerades som en webbaserad applikation med HTML, CSS och JavaScript. Valet gjordes för att separera verktyget från den ursprungliga Pygame-baserade miljön och för att göra lösningen mer tillgänglig på olika system. Det viktigaste var att verktyget kunde användas för att skapa en strukturerad konfigurationsfil som senare kan läsas in av stationens övriga mjukvara. En webbaserad implementation ger också möjlighet att vid behov hosta verktyget på en webbserver eller distribuera det som en fristående applikation.

4.2.2.3 Uppbyggnad av lagerlayout

I den slutliga versionen byggdes lagerlayouten upp kring en stationsvy där lagret delas in i fyra möjliga sidor, se konceptskissen på layouten i Figur 4.6.



Figur 4.6: Konceptskiss på lagerlayouten.

I stationsvyn delas lagret upp i fyra möjliga sidor: fram, bak, vänster och höger. Uppdelningen gjordes för att verktyget inte skulle vara låst till en station där all lagerförvaring endast finns framför operatören och för att bättre motsvara en fysisk arbetsstation. Varje sida hanteras därför som en separat lagerzon, vilket gör det möjligt att konfigurera olika delar av stationens lager oberoende av varandra.

Stationsvyn utformades som en översiktlig struktur för att dela upp lagret i separata lagerzoner. Inom varje zon skapas ett rutnät av lådor som kan byggas ut stegvis. Uppbyggnaden av lagret baserades på att det från en början ritas upp en tom möjlig lådposition med streckade linjer och ett plus-tecken i mitten. Efter att man har placerat den första lådan, fastställs den med heldragna linjer och en text som skriver ut tom" eller senare artikelnumret av komponenten som tilldelas till den lådan. Därefter ritas tomma möjliga lådpositioner upp både uppåt och åt höger sida för vidare uppbyggnad av lagret. Syftet med detta var att göra det tydligt var nya lådor kan läggas till utan att behöva ange hela lagerlayoutens storlek från början.

Genom att använda en rutnätsstruktur där uppbyggnaden utgår från det nedre vänstra hörnet kan verktyget internt spara koordinater för varje låda inom den specifika lagerzonen. Koordinaterna används för att hålla ordning på lådornas relativa placering i gränssnittet.

Till skillnad från den första versionen tilldelas varje låda initialt en standardbredd på 150 mm, eftersom detta motsvarade en ungefärlig genomsnittlig bredd för lådorna som användes vid prototypen. För att bredden inte skulle vara fast eller vara ett avbrott i uppbyggnaden av lagerlayouten, skapades även funktionalitet för att ändra bredden på en individuell låda vid behov.

Implementeringen av varje låda som ett objekt med position, bredd och eventuell komponentplacering gör att lagerlayouten kan sparas som strukturerad data. På så sätt behöver stationens lageruppbyggnad inte vara hårdkodad i mjukvaran, utan kan ändras genom att skapa en ny konfiguration i verktyget.

4.2.2.4 Komponentval och databasuppslagning

För att undvika manuell inmatning av komponentinformation eller en begränsning till komponenter som finns i redan skapade ritningar, kopplades lagerkonfigurations-verktyget till SQLite-databasen med DIN-artiklar med hjälp av biblioteket `sql.js`, som gör det möjligt att snabbt läsa från en SQLite-databas direkt i webbläsaren. Databasen används i verktyget för att söka fram komponenter baserat på fabrikat. Sökresultatet visar de artiklar som matchar det angivna fabrikatet och presenterar relevant information såsom fabrikat, artikelnummer och typ. För att underlätta sökningen hämtas resultat först när minst tre bokstäver har skrivits in. Dessa bokstäver matchas därefter mot fabrikat i databasen.

För att flera relevanta artiklar skulle kunna väljas ut vid samma sökning skapades funktionalitet för att markera flera komponenter i sökresultatet och avmarkera dem vid behov. Genom att spara alla markerade artiklar dynamiskt under sökning är det möjligt att söka efter ett annat fabrikat och kontinuerligt lägga till komponenter tills urvalet är färdigt. När urvalet är färdigt kan de markerade artiklarna läggas till i verktygets komponentlista som är synlig inom varje lagerzon och innehåller de artikelnummer som har valts ut från databasen. Därifrån kan en artikel väljas och tilldelas en tom lagerlåda i den aktuella lagerzonen.

Uppdelningen mellan databassökning och placering i lagerlayouten gjordes för att hålla verktyget mer överskådligt. Databasen används först för att söka fram och välja ut relevanta artiklar, som därefter läggs till i komponentlistan. Komponentlistan fungerar sedan som det urval av artiklar som kan placeras ut i den aktuella lagerlayouten, samtidigt som databasen kan användas igen om fler komponenter behöver läggas till.

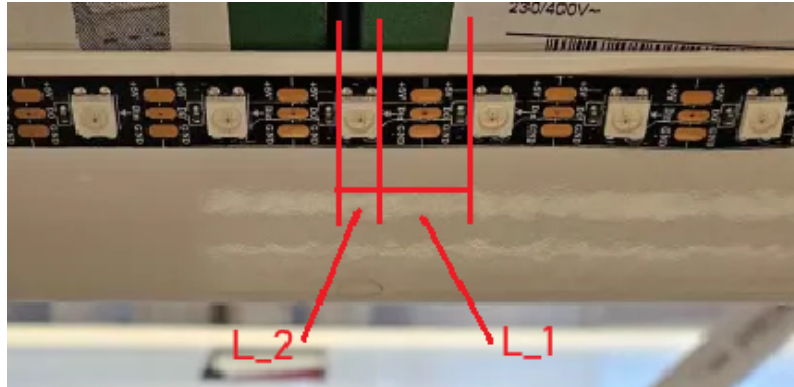
4.2.2.5 Generering av konfiguration

När lagerlayouten är byggd och komponenter är tilldelade kan konfigurationen sparas via en spara-knapp i huvudvyn. För att göra det möjligt att välja vilka lagerzoner som ska ha pick-to-light öppnas ett fönster där pick-to-light kan aktiveras eller inaktiveras individuellt för varje lagerzon. Detta gjordes eftersom det i feedback till första versionen framkom att alla hyllor runt arbetsstationen inte behöver ha en LED-list för pick-to-light, men det kan fortfarande vara viktigt att veta om det finns komponenter där.

För zoner med pick-to-light aktiverat beräknas ett LED-segment för varje låda i zonen. Beräkningen utgår från att LED-listen är installerad på så sätt att den börjar i det nedre högra hörnet av lagerzonen och fortsätter rad för rad uppåt i alternerande riktning eftersom det var så LED-listen var konfigurerad fysiskt vid den ursprungliga prototypen. Om flera zoner har pick-to-light aktiverat antas en och samma LED-list löpa genom samtliga aktiverade zoner. LED-listen antas börja i fram-zonen och fortsätter medurs enligt layouten i Figur 4.6, där LED-segment endast beräknas för zoner där pick-to-light är aktiverat.

Segmentet för varje låda definieras av en start-LED och ett antal efterföljande LED:er som ska tändas, vilket beräknas utifrån lådans bredd. Eftersom varje lådas

position och bredd är känd från lagerlayouten beräknas segmenten utifrån lådans bredd, LED-diodernas bredd och avståndet mellan dem, vilket illustreras i Figur 4.7



Figur 4.7: Bild för att illustrera LED:ns bredd och deras avstånd mellan varandra.

Antalet LED:er för varje låda beräknas enligt

$$n = \frac{\text{bredd}_{\text{låda}} + L_2}{L_2 + L_1}, \quad (4.2)$$

där n är antalet LED:er för den specifika lådan, L_2 är en LED-diods bredd och L_1 är avståndet mellan dioderna i mm enligt Figur 4.7. Start-LED för varje låda ges av n_{tot} , som räknas upp med n efter varje låda i sekvensen. L_2 adderas i täljaren med bredden eftersom den sista LED:en i segmentet inte behöver ha något avstånd efter sig och i slutändan avrundas n nedåt. Avrundningen nedåt valdes för att säkerställa att LED-segmentet aldrig överskrider lådans faktiska bredd, även om det innebär att ett litet överskott kan ignoreras. Detta gjordes som en förenkling av modellen där varje låda antas börja exakt på en LED-position.

4.2.3 Dynamisk ROI-beräkning

För att kamerabilderna ska vara användbara för både datainsamling och verifiering behöver varje bild vara beskuren till det område där komponenten befinner sig, snarare än att använda hela kamerabilden. Detta uppnåddes genom att använda ScalerCrop i Picamera2 biblioteket som beskär bilden direkt i kameran sensorn innan den sparas [13]. Fördelen med denna metod jämfört med att beskära en fullskärmsbild i efterhand är att ScalerCrop definierar det specifika sensorområde som ska användas innan bilden skalas upp till målupplösningen. Eftersom den fulla sensorupplösningen utnyttjas inom det valda området, undviks den kvalitetsförlust som digital beskärning innebär. Detta resulterar i en mer detaljerad bild av komponenten. Kvadratiska bilder eftersöktes för att undvika att komponenter behövde sträckas eller komprimeras vid bildbehandling eller uppskalning med ScalerCrop, vilket hade kunnat påverka bilddetaljer som text negativt. Kameran konfigurerades även att spara bilder i kvadratisk upplösning, 3040 x 3040, vilket i kombination med en kvadratisk ScalerCrop säkerställer att bilden varken sträcks eller komprimeras.

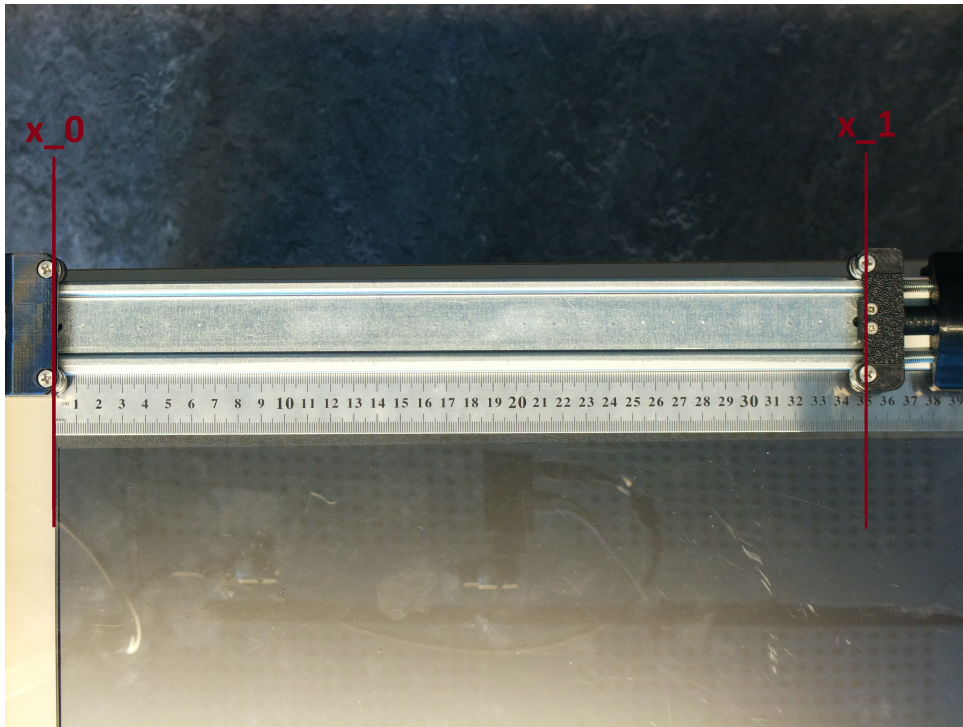
4.2.3.1 Kalibrering

För att kunna räkna på vad en position i verkligheten motsvarar på en kamerabild behövdes relationen mellan pixlar på bilden och millimeter i verkligheten kalibreras. Kalibrering av uträkningen av var i bilden en komponent kan sitta användes en fullskärmsbild från kameran i dess fulla upplösning, vilket för HQ-kameran är 4056x3040 pixlar. Detta illustreras i Figur 4.8.



Figur 4.8: Illustration av kamerans upplösning.

I bilden definieras koordinatsystemet med pixeln (0,0) i det övre vänstra hörnet, vilket innebär att det nedre högra hörnet motsvarar pixeln (4056,3040). Eftersom komponenter monteras längs DIN-skenan i horisontalled behövde endast relationen mellan pixlar och millimeter i x-led kalibreras. Detta gjordes genom att placera en linjal längs DIN-skenan och ta en fullskärmsbild, varifrån pixelpositionerna för kända millimetermärken på linjalen kunde avläsas.



Figur 4.9: Illustration av relationen mellan pixlar och millimeter i horisontalled.

Utifrån detta beräknades en skalningsfaktor i pixlar per millimeter enligt

$$s = \frac{x_1 - x_0}{d}, \quad (4.3)$$

där x_0 och x_1 är pixelpositionen för två kända punkter längs skenan, d är det verkliga avståndet mellan dem i millimeter horisontellt och s blir den slutliga skalfaktorn. I den aktuella kamerauppsättningen med en synlig DIN-skena på 350 mm avlästes startpositionen till $x_0 = 205$ pixlar och slutpositionen till $x_1 = 3613$ pixlar. Detta gav en skalningsfaktor på

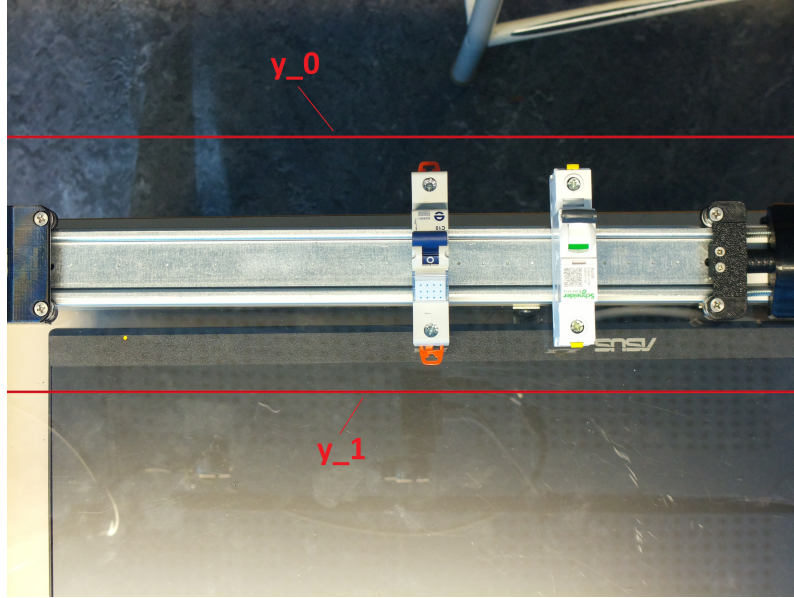
$$s = \frac{3613 - 205}{350} \approx 9,74 \text{ px/mm}. \quad (4.4)$$

Däremot monteras komponenter vanligtvis från vänster till höger längs skenan, vilket innebär att DIN-skenans startposition i verkligheten motsvarar $x_1 = 3613$ i kamerabilden och motsvarar millimeter 0 längs DIN-skenan. Skalningsfaktorn påverkas dock inte av riktningen, då relationen mellan pixlar och millimeter är densamma oavsett åt vilket håll man rör sig längs skenan.

Med skalningsfaktorn känd kan pixelpositionen för en given position längs DIN-skenan beräknas. För att beskära en kvadratisk bild centrerad kring en komponent vid position p mm från skenans start beräknas mittpunktens pixelposition enligt

$$x_{mid} = x_1 - p \cdot s, \quad (4.5)$$

där x_1 är skenans startposition i pixlar och s är skalningsfaktorn. Därifrån beräknas ROI:ns vänstra kant genom att subtrahera halva bredden från mittpunkten. För att bilden ska bli kvadratisk bestämdes först höjden på ROI genom att i en fullskärmsbild identifiera de y-värden i kamerabilden som garanterat fångar alla komponenters höjd oavsett komponenttyp.



Figur 4.10: Intervall av y-värden på pixlar som fångar alla komponenters höjd.

Som visas i Figur 4.10 sattes $y_0 = 700$ och $y_1 = 2000$, vilket ger en höjd på 1300 pixlar. Eftersom komponenter typiskt sett är högre än de är breda sattes bredden på ROI till samma värde, 1300 pixlar, med mittpunkten centrerad kring komponentens position längs skenan.

ScalerCrop-parametrarna för varje position definieras av fyra värden: x , y , bredd och höjd, där x är ROI:ns vänstra kant i pixlar. Med känd mittpunkt beräknas x enligt

$$x = x_{mid} - \frac{1300}{2}, \quad (4.6)$$

medan y hålls konstant vid 700 pixlar samt bredd och höjd vid 1300 pixlar. För att hantera fall där ROI hamnar utanför sensorns upplösning ges x slutligen enligt

$$x = \begin{cases} 0 & \text{om } x_{mid} - \frac{1300}{2} < 0 \\ 4056 - 1300 & \text{om } x_{mid} + \frac{1300}{2} > 4056 \\ x_{mid} - \frac{1300}{2} & \text{annars} \end{cases} \quad (4.7)$$

4.2.4 Teach Mode

För att systemet ska kunna verifiera korrekt montage med hjälp av endast kameror, behöver varje komponent som planeras användas vid stationen vara känd för

vision-modellen som används för igenkänning. Teach Mode utvecklades med syftet att samla in den bilddata som krävs för att träna dessa modeller. Eftersom användningsområdet är specifikt för elkomponenter och nya komponenter kontinuerligt tillkommer på marknaden som kan vara önskvärda att ha vid stationen, så finns det ingen förtränad modell som kan identifiera vilken komponent som sitter monterad på DIN-skenan. Den närmsta tillämpningen som hade varit applicabel hade varit generell objekt- eller konturdetektering som kan upptäcka att något finns på DIN-skenan, men kan inte avgöra vilken komponent det är samt riskeras att andra föremål som själva skenan eller fästen på stationen upptäcks också.

Teach Mode implementerades i Pygame för att hålla en liknande visuell uppvisning som vid montering. Programmet är uppbyggt kring en lista av artikelnummer som ska läras in. För varje komponent som ska läras in visas fem positioner längs DIN-skenan i tur och ordning. För att guida operatören till rätt placering användes till en början ett enkelt mittstreck på skärmen som komponenten skulle centreras runt, men detta upplevdes som otydligt i praktiken. Istället valdes två vertikala linjer som markerar komponentens vänstra och högra kant, vilket visade exakt hur bred komponenten är och var den ska placeras, som i sin tur minskar risken för felaktig placering vid bildinsamlingen. Bredden på området som ritas upp hämtas från komponentens bredd i databasen och skalas om till skärmkoordinater.

Bildtagningen hanteras av en Flask-server som körs på kameranoden och tar emot förfrågningar från Teach Mode programmet via HTTP. För varje position längs DIN-skenan skapades ett fördefinierat ROI som beskär kamerabilden till det område där komponenten befinner sig, vilket gör att bilden som sparas endast innehåller den relevanta delen av DIN-skenan.

När komponenten är placerad enligt instruktion på skärmen valdes en manuell bekräftelse med Enter-tangenten för att trigga bildtagningen. Detta valdes över ett automatiskt flöde som hade krävt en separat detektionsmodell som inväntar att komponenten är korrekt placerad, vilket också hade medfört behov av en kvalitetskontroll på bilden innan man gick vidare. Den manuella bekräftelsen bidrar inte med någon betydelsefull tidsförlust och säkerställer också att operatören kan trigga bildtagningen först när inget är i vägen.

4.2.5 Bilder för uppritning

I den ursprungliga prototypen representerades komponenter i monteringsvyn med bilder hämtade från antingen tillverkares eller återförsäljares mediamaterial, eller CAD-ritningar. För att undersöka om detta tillvägagångssätt var lämpligt att bygga vidare på undersöktes tillgängligheten av sådant material för ett antal av komponenter som planerades användas vid stationen hos en av kunderna.

För vissa komponenter fanns CAD-modeller tillgängliga i STEP-format, men dessa var ofta enfärgade utan ytdetaljer eller märkning eftersom STEP-formatet i huvudsak beskriver geometri och inte yttre utseende. Mediabilder från tillverkare förekom också ofta i produktbladet för komponenten, men dessa var sällan tagna rakt uppiifrån vilket krävdes för monteringsvyn. Även om det troligtvis hade varit möjligt att

kontakta tillverkare för att få tillgång till lämpligt bildmaterial, innebar det ett beroende av svarstider och ingen garanti på att materialet faktiskt fanns tillgängligt i önskat format. Dessutom varierar texten och märkningen på komponenternas ovasida mellan modeller, vilket innebär att generiska bilder inte alltid ger en tillräckligt korrekt representation av den specifika komponenten.

Det upplevdes därför som mer praktiskt att istället använda bilder från Teach Mode för uppritning i monteringsvyn. I Teach Mode valdes en av positionerna till att vara centrerad i kameravyn, vilket resulterar i en bild på komponenten ovanifrån som kan användas för uppritning. Även om kameran inte är placerad rakt ovanifrån innebär detta att samtliga komponenter fotograferas under samma förhållanden och med samma kameravinkel, vilket ger en konsekvent visuell representation jämfört med att blanda CAD-modeller, mediabilder och egna bilder från olika källor.

För att bilderna skulle passa i monteringsvyn behövde bakgrunden tas bort så att endast komponenten var synlig. Detta gjordes med hjälp av biblioteket rembg [14]. Biblioteket erbjuder stöd för flera olika förtränade AI-modeller för segmentering, varav bria-rembg och BiRefNet-general valdes ut för projektet. Vid tester med olika komponenter som fanns vid stationen, så lyckades modellerna korrekt identifiera komponenten i bilden varje gång. Däremot var det inte felfri borttagning av bakgrunden vid varje test, i vissa fall misstogs starka skuggor på komponenten för att vara en del av bakgrunden, och i andra fall kvarstod visuella artefakter efter bearbetningen. Den bästa bilden från någon av dessa två modeller valdes ut för att användas för uppritning.

Efter bakgrunds borttagningen innehöll bilden en stor mängd transparenta pixlar där bakgrunden hade varit, därför användes Python Imaging Library (PIL) för att beskära bilden till komponentens faktiska område genom att ta bort rader och kolumner som enbart bestod av transparenta pixlar. Slutligen skalades bilden så att bredden i pixlar motsvarade komponentens verkliga bredd på skärmen, vilket gör att komponenten ritas upp i korrekt skala i monteringsvyn.

4.2.6 Verifiering av montage

En del av arbetet var att undersöka hur montage av komponenter på DIN-skenan kunde verifieras med hjälp av kamerabaserad bildanalys. Målet var att ta fram och testa en möjlig metod för att avgöra om rätt komponenter har placerats under montering enligt ritningen som följs.

Eftersom ritningsfilen redan innehåller information om var varje komponent ska placeras på DIN-skenan kunde verifieringen utgå från en förväntad position. Därför fokuserade arbetet på att klassificera en bild från ett beräknat ROI-område snarare än att först detektera komponenten i hela kamerabilden. Utöver detta ansågs det lämpligt med klassificering eftersom det kräver mindre datorkraft för att utföra bedömningar i realtid på den Raspberry Pi 4 som fanns vid stationen.

4.2.6.1 Val av vision-modell

För verifieringen valdes en bildklassificeringsmodell i stället för en objektidentifieringsmodell. Anledningen var att komponentens förväntade position redan kan hämtas från ritningsfilen, vilket kunde utnyttjas för verifieringen.

Utifrån komponentens förväntade position kan ett ROI-område beräknas i kameran bilden. Bilden som skickas till modellen behöver därför inte innehålla hela DIN-skenan, utan endast det område där komponenten förväntas sitta. Modellens uppgift blir då att avgöra vilken komponent som finns i detta område, snarare än att både lokalisera och klassificera komponenten i en större bild.

Detta gjorde bildklassificering mer lämpligt än objektidentifiering för den här tillämpningen. En objektidentifieringsmodell hade krävt att komponenternas position även behöver markeras manuellt i träningsbilderna, exempelvis med bounding boxes, eller i vissa fall segmenteringsmasker, vilket är metoder för att ange var objektet finns i bilden. Med en klassificeringsmodell räcker det istället att bilderna sparas i en mappstruktur där varje klass motsvarar ett artikelnummer. Detta passade bättre ihop med Teach Mode, eftersom systemet redan vet vilken komponent som fotograferades vid varje tillfälle, vilket gjorde filhanteringen enklare. En ytterligare anledning var att en manuell annoteringsprocess hade gjort systemet svårare att använda i praktiken. Om en kund själv ska kunna skapa ett dataset för sin station är det fördelaktigt om bilderna kan samlas in och sorteras automatiskt, utan att varje bild behöver efterbearbetas manuellt. Annotering av bilderna kan dessutom påverkas av hur personen som skapar datasetet tolkar objektets gränser, vilket kan ge variation i träningsmaterialet. Genom att använda klassificering kunde datainsamlingen därför göras enklare och mer konsekvent.

Som första modell valdes YOLO11n-cls via Ultralytics. Nano-varianten valdes eftersom den är en lätt modell, vilket var viktigt eftersom stationens huvuddator var en Raspberry Pi 4. Modellen laddades med förtränade vikter i början av träningen, eftersom detta rekommenderas vid träning av klassificeringsmodeller. Ultralytics användes eftersom ramverket ger ett enklare arbetsflöde för att träna, validera och testa modeller, samtidigt som träningsparametrar och bildaugmentationer som man vill använda kan anges direkt i träningskoden. Detta gjorde det möjligt att snabbt genomföra flera testträningar och justera dataset eller träningsinställningar efter observerade resultat. Enkortsdatorer som bland annat Raspberry Pi är vanliga att använda inom industriella tillämpningar, men för själva träningen av modellen användes extern dator.

Under de senare testerna undersöktes även YOLO26n-cls som ett alternativ. Den modellen valdes som jämförelse eftersom den är avsedd att vara lättare att köra på begränsad hårdvara och passade arbetets inriktning mot körning med Raspberry Pi som huvudenhet.

4.2.6.2 Dataset och första modellträning

Det första datasetet samlades in med hjälp av Teach Mode och kamerauppställningen ovanför DIN-skenan. Datasetet innehöll bilder på nio komponenter samt en klass

för bilder på tom DIN-skena. Den tomma klassen lades till för att modellen inte skulle tvingas klassificera en tom ROI som någon av komponenterna, eftersom ROI bilderna på komponenterna innehöll mycket av DIN-skenan och stationen vid sidan om.

För varje komponent togs bilder vid fem positioner längs DIN-skenan. Positionerna användes för att ge variation i var komponenten befann sig i kamerabilden. Bilderna organiserades i mappar där varje artikelnummer motsvarade en klass enligt följande struktur:

```
dataset
├── train
│   ├── tom
│   ├── artikelnummer1
│   └── artikelnummer2...
└── val
    ├── tom
    ├── artikelnummer1
    └── artikelnummer2...
```

Positionerna valdes linjärt längs DIN-skenan. Detta datasetet samlades in kort efter att HQ-kameran hade monterats ovanför stationen. Teach Mode var då ännu inte anpassat för dynamisk ROI-bildtagning, men positionerna valdes enligt samma grundlogik som senare användes i Teach Mode.

För den första modellen användes en 550 mm DIN-skena, där cirka 400 mm var synlig i kamerabilden. Startpositionen sattes till 40 mm från vänster och slutpositionen till 360 mm från vänster. Dessa värden avsåg komponentens mittpunkt. Positionerna valdes så att samtliga komponenter i testet skulle få plats inom den synliga delen av skenan. De slutliga positionerna på DIN-skenan visas i Tabell 4.1.

Tabell 4.1: Positioner längs DIN-skenan vid första datasetet

Position	Mittpunkt längs skenan [mm]
0	40
1	120
2	200
3	280
4	360

Position 2 användes som mittpunkt eftersom kameran där hade ungefär en ren toppvy. För uppdelningen mellan tränings- respektive valideringsdata, användes position 0, 2, och 4 som träningsdata, medan position 1 och 3 användes som valideringsdata. Syftet var att valideringen skulle ske på positioner som modellen inte hade tränats på, samtidigt som träningsdatan fortfarande innehöll bilder där komponenten syntes från olika vinklar. Exempelvis om position 0 och 1 hade valts som valideringsbilder,

har modellen ingen träningsdata på hur komponenten ser ut från ena sidan. Bilderna togs genom att lägga en komponent vid varje position längs DIN-skenan, ta en fullskärmsbild och sedan skapades manuella ROI-områden för varje position.

Den första modellen tränades som en klassificeringsmodell med bildstorleken 224 x 224 pixlar. Under träningen användes enklare augmentationer som passade syftet med en fast kamera och en konstant miljö. Augmentationerna som användes var:

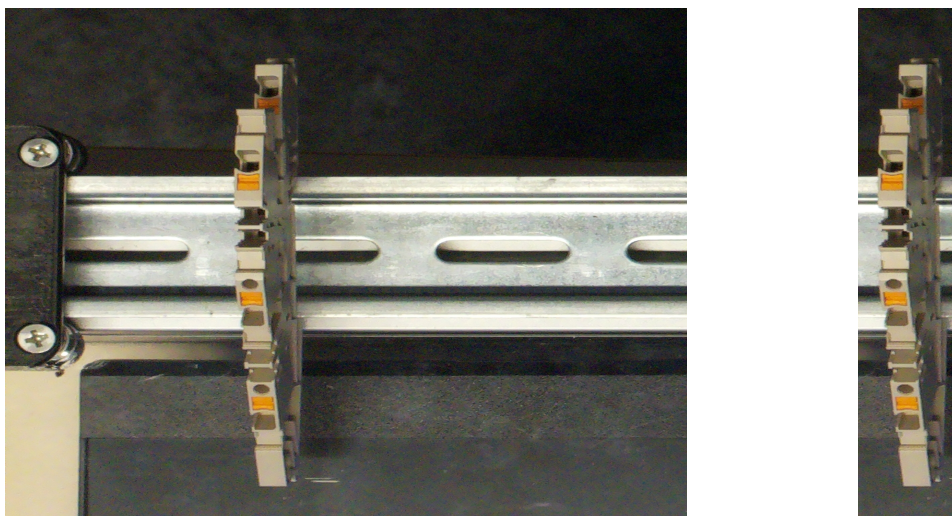
- $\pm 10\%$ ljusstyrka.
- Percentuell bildförskjutning i x och y led med 0% till 10% av bildstorleken.
- 0 till 5 graders vinkling.

Alla augmentationer appliceras slumpmässigt inom det angivna intervallet på träningsbilderna under träningen inom Ultralytics ramverk.

4.2.6.3 Justering av ROI och träningsdata

De första modellerna testades främst teoretiskt. Modellerna kördes på en extern dator utan koppling till Raspberry Pi:n och validerades på bilder som hade tagits tidigare, exempelvis från tidigare testdataset och kvalitetstester med kameran. Det var fortfarande bilder som modellen inte hade sett under träningen, vilket är viktigt vid utvärdering av modellen. Däremot hade vissa av bilderna inte samma avstånd, ljusförhållanden eller kamerauppställning som den aktuella verifieringsmiljön. Resultatet behövde därför tolkas med viss försiktighet.

Under dessa initiala tester blev det tydligt att modellens resultat påverkades mycket av hur ROI-bilderna togs och skalades. Ett tidigt antagande var att modellen skulle kunna tränas bättre om onödig bakgrund togs bort från bilderna. Syftet var att göra modellen mindre beroende av bakgrundskontext, såsom DIN-skenan och synliga delar av stationen. Därför skapades mer beskurna versioner av datasetet, där sidorna i originalbilderna redigerades bort manuellt så att nästan endast komponenten var synlig. Skillnaden visas i Figur 4.11.



Figur 4.11: Till vänster syns originalbilden och till höger den beskurna versionen.

Vid testning av modellen med det utökade datasetet visade det sig att resultatet blev överlag bättre. Orsaken till förbättringen visade sig däremot vara problematisk. Ultralytics sparar bilder från de tre första träningsbatcherna, och genom dessa bilder blev det tydligt att de mycket avlånga bildförhållandena hanterades på ett sätt som inte var önskvärt. När bilderna matades till modellen och anpassades till 224 x 224 pixlar skalades de avlånga bilderna så att sidan närmast 224 pixlar skalas till den storleken. Eftersom komponentbilderna ofta hade större höjd än bredd hamnade delar av komponenten utanför den kvadratiske ytan och skars bort, istället för att krympas extremt. Det gjorde att informationen från komponentens över- och undersida förlorades.



Figur 4.12: Den sparade bilden av den första träningsbatchen.

I figur 4.12 ser man tydligt hur vissa av bilderna behandlades inför träningen. Detta förklarade även varför modellen kunde prestera bättre på vissa testbilder som inte var tagna på samma sätt som träningsdatan. Även dessa bilder gick igenom en liknande beskärning innan klassificering. Modellen hade därför delvis tränats på att känna igen komponenter utifrån mittdelen av bilden, snarare än hela komponentens utseende. Detta gick emot syftet med veriferingen, eftersom flera komponenter skiljer sig åt genom små detaljer som inte nödvändigtvis finns vid komponentens mitt, samt att systemet ska bygga på att det går att återskapa bilderna som modellen har tränats på i aktuell verifering. Vid ytterligare analys av bilderna från de första träningsbatcherna, blev det också tydligt att viss augmentation applicerades på bilden som inte var planerat. Om man inte anger en specifik augmentationsinställning i träningskoden, kan dessa anta olika default-värden. Detta var tydligt tidigt med

augmentationer som ändrar bildens nyans eller måtttnad. Vid detta tillfälle hade dessa mestadels tagits bort, men i Figur 4.12 syns effekten av augmentationen "erasing", där delar av bilden täcks över på en av bilderna, som togs bort efter denna versionen.

En ytterligare viktig lärdom var att ROI-bilden eller andra bilder som testas med modellen måste innehålla hela komponenten och samtidigt ha ett bildformat som stämmer överens med det modellen har tränats på. För hårt beskurna bilder, eller mer generellt bilder som har för stor skillnad i bildförhållande jämfört med vad modellen förväntar sig medför detta att modellen kan tränas eller testas på ofullständig information, även om bilden vid första anblick verkar mer fokuserad på komponenten.

Det insågs också att även om kvadratiska ROI-områden definierades i ScalerCrop sparades bilderna fortfarande i kamerans fulla upplösning, 4056 x 3040 pixlar. Detta gjorde att den sparade bilden blev rektangulär, vilket syns i den vänstra bilden i Figur 4.11. Tillsammans med lärdomarna kring beskärning, bildförhållande och augmentationer ledde detta till att datasetet gjordes om med kvadratiska bilder från kameran, där bilderna sparades i rätt upplösning redan vid bildtagningen och där lösningen med hårt beskurna bilder togs bort.

4.2.6.4 Slutlig träningsstrategi

Efter de första testerna justerades träningsstrategin. Ett nytt dataset skapades med bilder som hade kvadratisk upplösning, vilket bättre motsvarade modellens kvadratiska indataformat. Lösningen med de hårt beskurna versionerna för att isolera komponenten togs bort eftersom det kunde leda till att delar av komponenten försvann vid omskalning. I stället användes ROI-bilder där hela komponenten och en del av omgivande DIN-skena fanns med.

Träningsstrategin anpassades efter att kameran och DIN-skenan är fast monterade i stationen. Därför användes endast begränsade augmentationer. Mindre variationer i ljusstyrka, rotation och position bedömdes vara rimliga eftersom komponenter kan placeras något olika och belysningen kan variera något. Däremot stängdes augmentationer av som inte motsvarade den verkliga miljön, exempelvis spegling, större skalning och kraftiga färgförändringar. Detta var särskilt viktigt för komponenter som plintar, där små färg- och formdetaljer kan vara avgörande för klassificeringen. De slutliga träningsinställningarna och augmentationerna som användes visas i Bilaga A.

För att undersöka om modellen kunde göras mer robust mot förändringar i ljusexponering skapades även en datasetvariant där bilderna kompletterades med mörkare kopior, motsvarande 85 % av den ursprungliga ljusstyrkan. Detta gjordes eftersom linsen på kameran ställdes in för att släppa in en stor mängd ljus, dels för att få bra bilder till uppritning, men också för att göra detaljer tydliga. Kompletteringsdatasetet skapades för att undersöka om det hjälpte i verklig modellprestanda.

Flera modeller tränades därefter för att kunna jämföra både klassificeringsförmåga och prestanda. Den första modellen baserades på YOLO11n-cls med bildstorleken

224 x 224 pixlar. Därefter tränades en alternativ variant, YOLO26n-cls, eftersom den ansågs vara mer intressant för körning på begränsad hårdvara och det är en nyare modell. För YOLO26n-cls testades både 224 x 224 pixlar och 320 x 320 pixlar, för att undersöka om smalare komponenter kunde klassificeras bättre när mer bildinformation bevarades. De slutliga modellerna som tränades och deras dataset presenteras i Tabell 4.2.

Tabell 4.2: Modeller som tränades och jämfördes i verifieringstestet.

Kortnamn	Modell	Bildstorlek [px]	Dataset
YOLO11n 224	YOLO11n-cls	224 x 224	Original
YOLO26n 224	YOLO26n-cls	224 x 224	Original
YOLO26n 224 mörk	YOLO26n-cls	224 x 224	Original + mörkare kopior
YOLO26n 320 mörk	YOLO26n-cls	320 x 320	Original + mörkare kopior

Kortnamnen i Tabell 4.2 används i resultatredovisningen för att underlätta dokumentation.

4.2.6.5 Testmiljö

I en fullständig integration där Raspberry Pi 4:an kontinuerligt tar emot ett kameraflöde med rätt ROI från kameranoden och ger direkt återkoppling under montering hann inte färdigställas inom arbetets tidsram. För att ändå kunna utvärdera modellen och dess prestanda på Raspberry Pi 4:an skapades en enklare testmiljö.

I testmiljön placerades komponenter på bestämda positioner längs DIN-skenan. Tio positioner mellan 40 mm och 310 mm slumpades fram. Därefter slumpades det också fram tio komponenter från de komponentklasser som modellen hade tränats på, där varje komponent tilldelades en av de slumpade positionerna. Positionerna användes för att skapa ROI-bilder på komponenter som inte nödvändigtvis motsvarade exakt de positioner som fanns i träningsdatasetet.

För varje testfall beräknades ROI-området utifrån komponentens position längs DIN-skenan enligt vad som beskrivs i avsnitt 4.2.3. Det skapades en modifierad version av Teach Mode där bildservern räknade ut ROI-området dynamiskt för varje slumpad position, medan huvudprogrammet på Raspberry Pi 4:an ritade upp var komponenten skulle placeras längs DIN-skenan för de tio komponenterna. Bilderna sparades sedan och flyttades till Raspberry Pi 4:an och kördes därefter genom de modeller som skulle jämföras. På detta sätt kunde samma testbilder användas för samtliga modeller, vilket gjorde jämförelsen mer rättvis.

Modellerna exporterades även från PyTorch-formatet vilket ultralytics vanligtvis sparar modellen i, till NCNN-format för att kunna testas mer effektivt på Raspberry Pi 4:an. Båda formaten testades för att jämföra skillnaden mellan vanlig modellkörning och ett format som är bättre anpassat för körning på begränsad hårdvara. Under testet loggades modellens toppklass, confidence, näst högsta klass och prediktionstid för varje bild. Testet genomfördes för att jämföra både klassificeringsförmåga och prestanda mellan modellerna, inte endast formaten.

I tabell 4.3 beskrivs exakt vilka positioner som slumpades fram och vilken komponent som placerades där.

Tabell 4.3: Testfall som användes vid verifieringstestet.

Testnummer	Position längs DIN-skenan [mm]	Artikelnummer
0	47	A9Z21440
1	308	A9F07316
2	195	EM103C10
3	57	2668950000
4	106	EM103C16
5	62	2668900000
6	50	2669140000
7	118	A9F07110
8	92	EM101C10
9	135	EPBR216C30

4.2.6.6 OCR för verifiering

Tidigt i arbetet med verifiering identifierades att en vision-modell ensam inte skulle räcka för att särskilja komponenter som tillhör samma komponentfamilj men har olika specifikationer. Ofta är det som skiljer dessa komponenter åt en text på ovansidan av komponenten av dess specifika variant eller ibland dess artikelnummer. För att hantera detta ansågs någon form av textigenkänning, OCR, vara nödvändig som komplement till vision-modellen.

Metoden liknar den QR-kodsbaserade verifieringen i den ursprungliga prototypen, där kameran använde ett förväntat område i bilden för att lokalisera och avläsa koden. En viktig skillnad är dock att den text som skiljer komponentvarianter åt nästan alltid finns på komponentens ovansida, vilket innebär att den aktuella kameravyn passar väl för denna typ av avläsning till skillnad från QR-koder som sällan placeras där.

Eftersom komplett verifieringsflöde inte implementerades under projektets gång valdes ändå att utvärdera en OCR-modell separat på bilder från det insamlade datasetet, körades på en extern dator. På så sätt kunde OCR-metodens lämplighet för verifiering bedömas utan att ett fullständigt integrerat system krävdes.

För att OCR-resultat ska kunna användas för verifiering behöver den avlästa texten kopplas till förväntad komponent. Eftersom artikelnumret för varje komponent är känt från ritningen kan databasen användas för att hämta fälten Typ och Data för det förväntade artikelnumret. Dessa fält innehåller nyckelord som beskriver komponentens egenskaper. Om den text som OCR-modellen läser av från komponentens ovansida matchar dessa nyckelord kan komponenten anses vara korrekt. I tabell 4.4 visas exempel på två artikelnummer och informationen från deras Typ och Data från databasen.

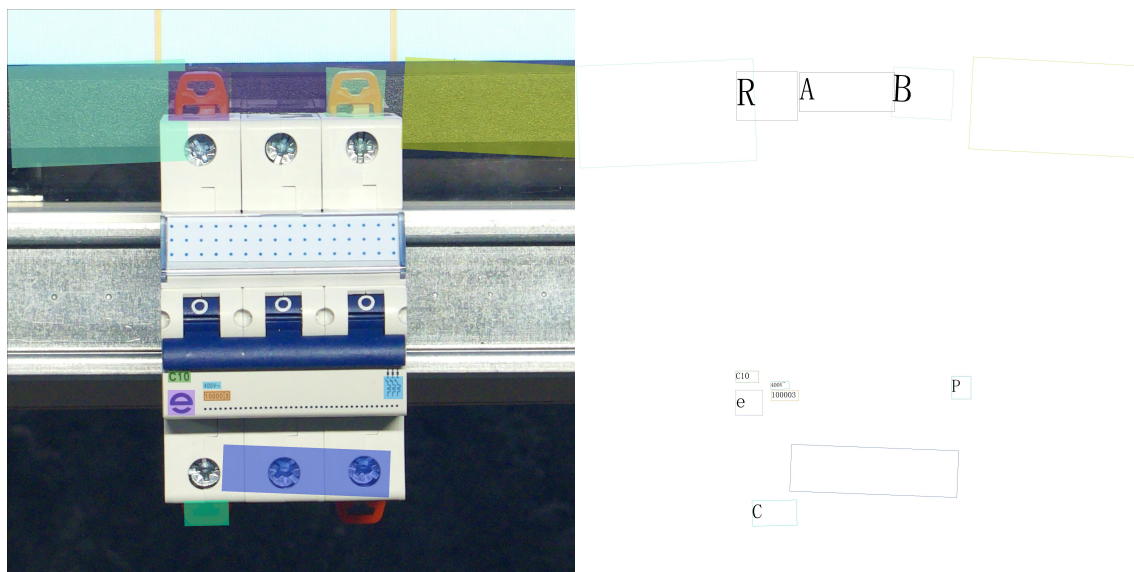
Tabell 4.4: Exempel Typ och Data för två artiklar från databasen.

Artikelnummer	Typ	Data
EM103C10		10kA 3P C10A
A9F07110	i60H 1P C10A	10A 1P C 10KA

Som visas i tabellen saknar vissa artiklar information i antingen Typ eller Data, vilket innebär att båda fälten behöver läsas och kombineras för att få en fullständig bild av komponentens egenskaper. Ytterligare en utmaning är att nyckelorden i databasen inte är helt konsekventa, utan exempelvis kan värden separeras med snedstreck istället för mellanslag, vilket behöver hanteras vid jämförelsen mot OCR-resultat.

För OCR-utvärderingen valdes PaddleOCR eftersom det erbjuder kompakta modeller och har inbyggt stöd för automatisk textrotation, vilket innebär att texten på komponenten kan vara orienterad i valfri riktning utan att det påverkar avläsningen.

Vid ett initialt test på en av träningsbilderna för *EM103C10* i 4.4 genererade modellen ett stort antal detektioner, varav flera var på grund av brus och vissa texter hade specialtecken. En första körning av modellen sparade en resultat bild som visas i Figur 4.13.

**Figur 4.13:** Illustrativt resultat av OCR körning med PaddleOCR.

Till höger i Figur 4.13 kunde man se att den plockade upp många tecken eller bokstäver från former som fanns på komponenten, men den lyckades också hitta ett nyckelord som *C10* och *400V~*. Bilden i figuren är också vriden 180 grader jämfört med inmatningsbilden, vilket visade att stöd fri orientering på texten fungerade korrekt. För att rensa resultatet från specialtecken och brus, filterades resultatet först genom att endast spara ord där modellens confidence var högre än 80 %. Därefter applicerades det reguljära uttrycket $[A-Za-z0-9]^+$, vilket fungerade som ett filter för att enbart släppa igenom sammanhängande ord och tal bestående av

vanliga bokstäver och siffror. Därefter sparades också resultatet i en array istället för en bildanalys.

Utöver det initiala bildtestet kördes OCR-modellen på träningsbilder för båda komponenterna i Tabell 4.4, där resultatet för varje komponent sparades som en array av identifierade nyckelord för vidare analys mot databasen.

5

Resultat

Detta kapitel presenterar de resultat som har uppnåtts under projektets gång, uppdelat efter de olika delar av systemet som utvecklats. Fokus ligger på vad som fungerade och vad som utvärderades.

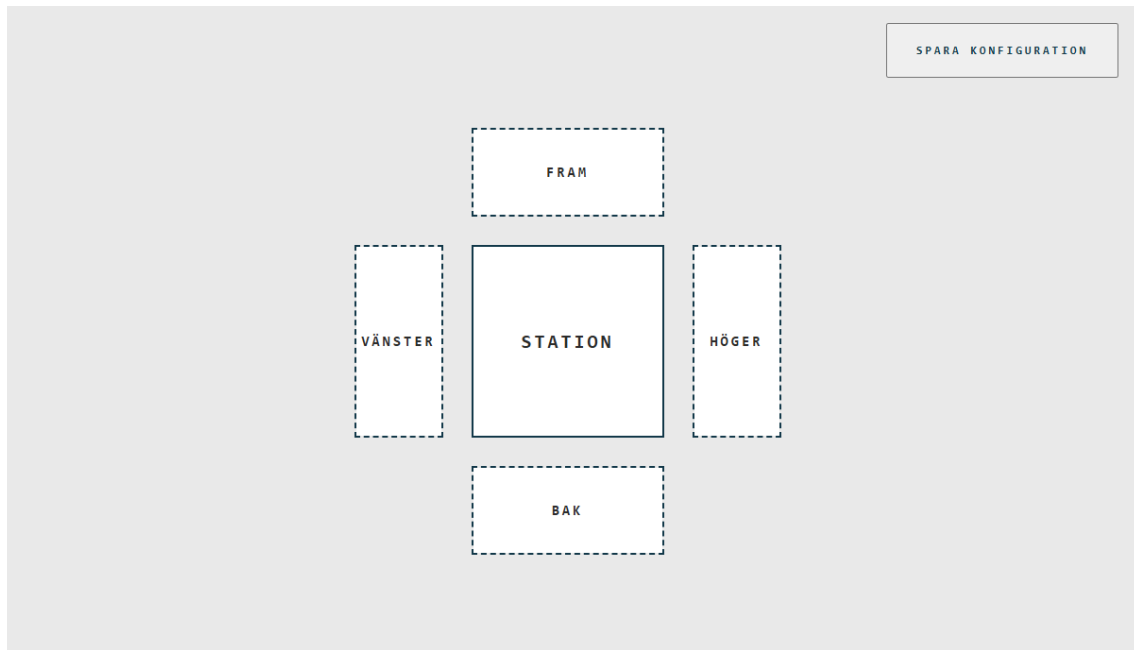
5.1 Lagerkonfigurationsverktyg

Resultatet blev ett webbaserat lagerkonfigurationsverktyg där användaren kan konfigurera en stationsnära lagerlayout. Verktöget består av en stationsvy, separata konfigurationsvyer för stationens fyra sidor och funktioner för att söka fram komponenter från DIN-artikeldatabasen.

I den färdiga versionen kan användaren bygga upp lagerzoner med komponentlådor, ange lådbredd och placera valda artiklar i respektive låda. Verktöget kan även hantera om en sida har pick-to-light eller inte.

5.1.1 Stationsvy

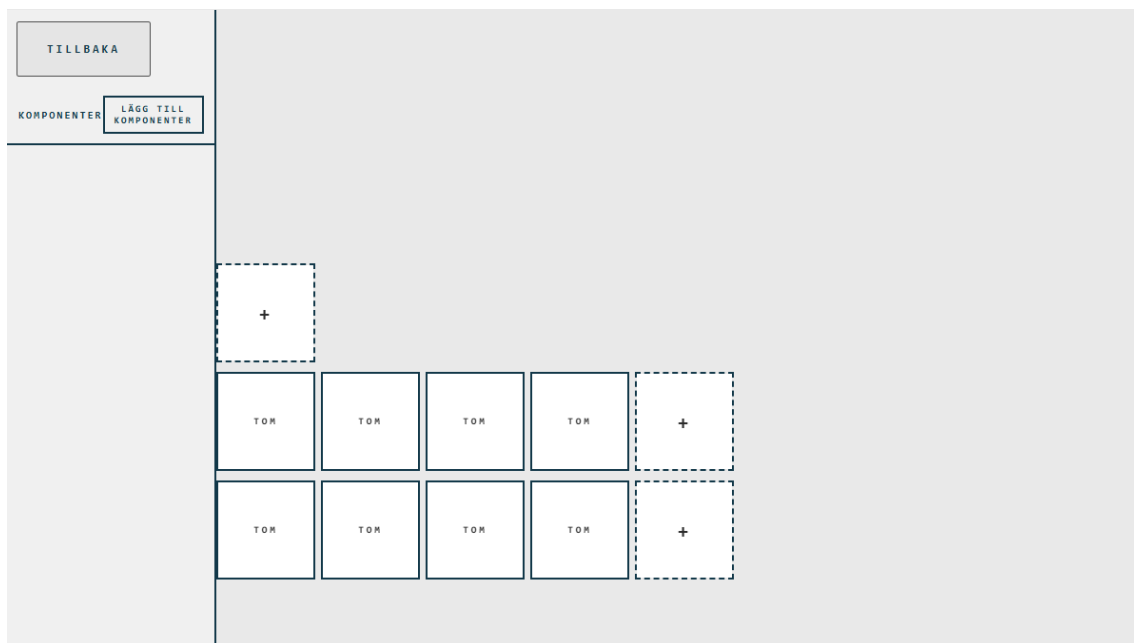
Startsidan består av en stationsvy där stationen visas i mitten och de fyra lagerzonerna är placerade runt omkring. Användaren kan välja fram-, bak-, vänster eller höger sida beroende på vilken del av stationen som ska konfigureras. Figur 5.1 visar verktygets startside.



Figur 5.1: Startside i lagerkonfigurationsverktyget med fyra valbara lagerzoner.

5.1.2 Konfiguration av lagerzon

När en lagerzon öppnas visas en separat konfigurationsvy för den valda sidan. I denna vy kan användaren bygga upp med lådor som motsvarar den fysiska hyllans uppbyggnad. Varje låda ges en individuell bredd, vilket gör att layouten kan anpassas till olika lådstorlekar.



Figur 5.2: Exempel på konfigurerad lagerzon med möjlighet att bygga ut.

Det går också att ändra bredden på en låda genom att trycka på den vilket öppnar fönstret som visas i figur 5.3.



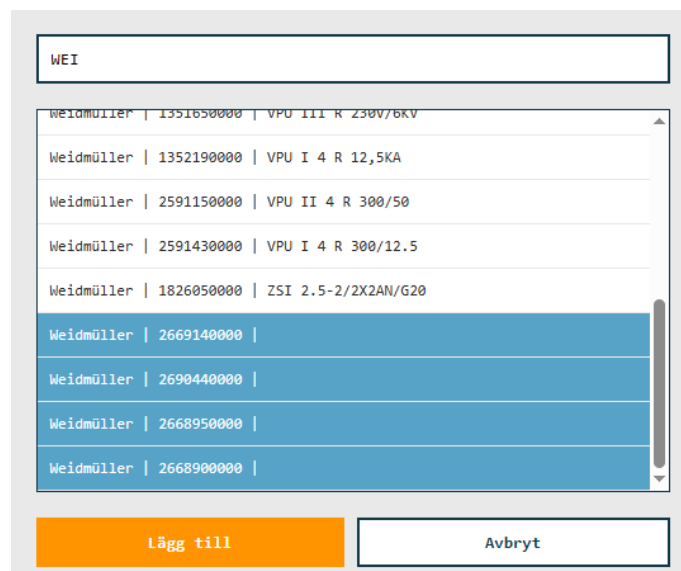
Figur 5.3: Exempel på ändring av lådbredd.

I fönstret framgår också vilken komponent som aktuellt är tilldelad lådan man har tryckt på, alternativt står det 'tom' ifall ingen komponent har placerats ännu.

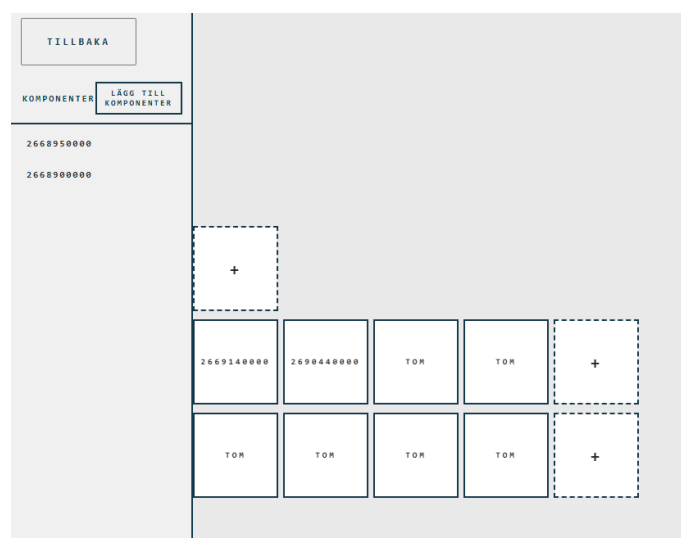
5.1.3 Komponentplacering

Komponenter väljs från SQLite-databasen via verktygets sökfunktion. När en komponent har lagts till i komponentlistan kan den placeras i en tom låda i den valda lagerzonen. Den placerade komponenten visas därefter i lådan med sitt artikelnummer, vilket gör det möjligt att visuellt kontrollera att layouten motsvarar den fysiska stationen.

Figur 5.4 visar sökfunktionen för komponenter och Figur 5.5 visar exempel på komponenter placerade i lådor.



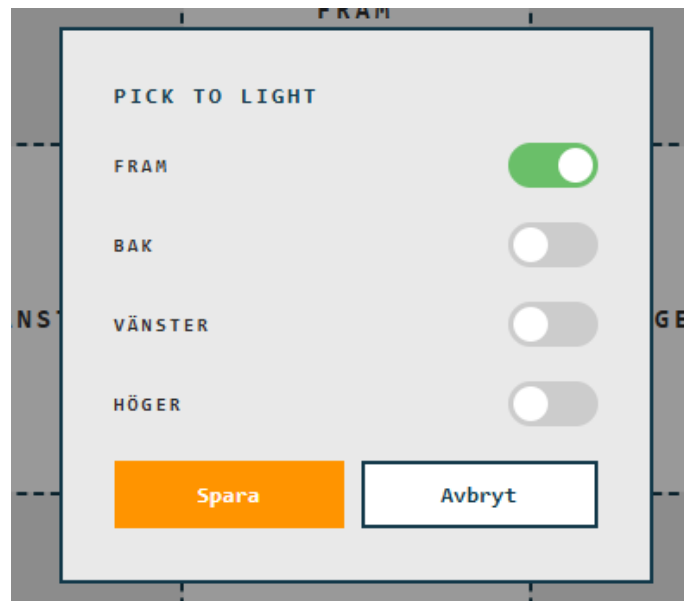
Figur 5.4: Exempel på sökning och markering av önskade komponenter.



Figur 5.5: Exempel på konfigurerad lagerzon med placerade komponenter.

5.1.4 Sparning och LED-mappning

Vid sparning via spar-knappen i stationsvyn visas ett fönster där användaren väljer vilka lagerzoner som ska ha pick-to-light aktiverat. För de valda zonerna genererar verktyget LED-information för respektive låda i form av start LED:en och antalet efterföljande LED:s. Utöver detta sparas zonen såsom fram, höger, vänster eller bak och lådans position i rutnätet också. Detta gör att alla komponenters information sparas oavsett om de ska använda pick-to-light eller inte. I Figur 5.6 visas fönstret vid sparning.



Figur 5.6: Fönstret som visas innan configurationen sparas.

Konfigurationen sparas därefter i JSON-format, vilket möjliggör en enklare hantering av configurationen än om den hade sparats i en vanlig textfil. Ett exempel på hur en sparad konfiguration visas i Bilaga B.

5.1.5 Användarflöde

Användarflödet börjar med att välja vilken sida av stationen som ska konfigureras. Därefter byggs lagerlayouten upp genom att lådor läggs till i den valda lagerzonen. För varje låda kan användaren sedan ange en fysisk bredd i millimeter.

När lagerlayouten är uppbyggd används sökfunktionen för att välja komponenter från DIN-artikeldatabasen. De valda komponenterna placeras sedan i komponentlistan och kan tilldelas specifika lådor. Komponentlistan behålls bland zonerna. När configurationen är färdig i alla zoner trycker användaren på spara i stationsvyn och anger vilka sidor som har pick-to-light aktiverat.

Efter detta genererar verktyget den information som behövs för att koppla artikelnummer till lådplacering, och där pick-to-light används, till ett beräknat LED-segment.

5.2 Teach Mode och bildinsamling

Resultatet blev ett inlärningsflöde för att samla in bilder till klassificeringsmodellen på ett mer styrt sätt. Gränssnittet visar vilket artikelnummer som ska fotograferas och vid vilken position komponenten ska placeras på DIN-skenan. Komponentens bredd används för att rita ut två markeringar på skärmen, vilket gör att komponenten kan placeras mer konsekvent vid bildtagning.

Teach Mode används tillsammans med kameranoden. När bilden tas skickas en begäran till bildservern, som beräknar rätt ROI för komponentens position och sparar bilden.



Figur 5.7: Teach Mode under användning.

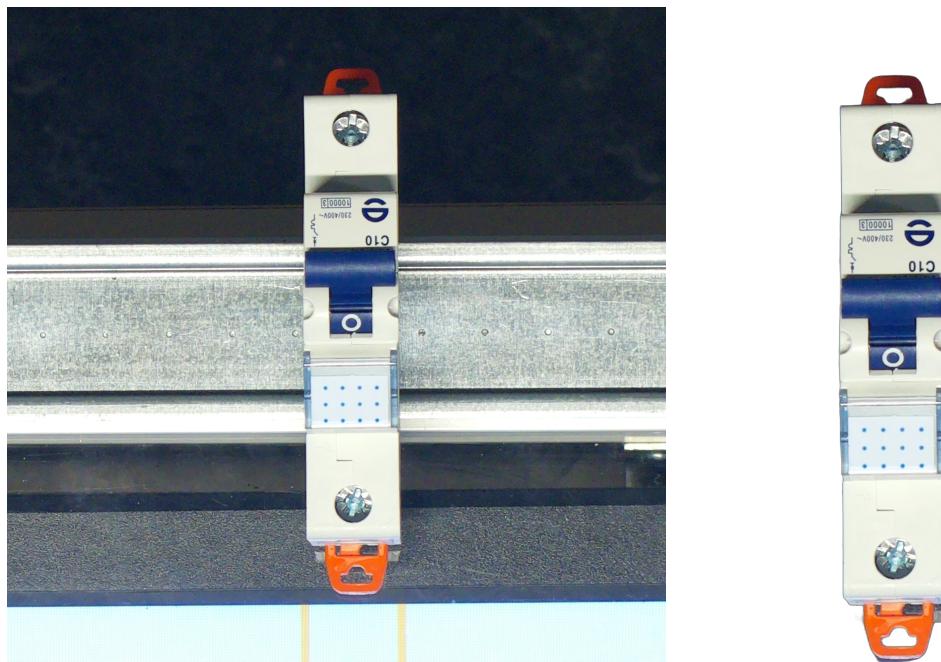
En förbättring med den första bildinsamlingen var att bilderna kunde sparas direkt i den struktur som krävs för träning av klassificeringsmodellerna. Varje artikelnummer motsvarar en klassmapp och bilderna placeras automatiskt i tränings- eller valideringsmappen beroende på vilken position som fotograferas. Detta minskar behovet av manuell sortering av datasetet och gör bildinsamlingen mer repeterbar. Filsystemet som används är den som framtoogs i avsnitt 4.2.6.2, Dataset och första modellträning.

5.3 Monteringsvy

Som ett resultat av arbetet med bilder för monteringsvyn togs en metod fram för att automatiskt framställa komponentbilder lämpade för uppritning. Metoden utgår från bilder insamlade från Teach Mode, där bakgrunden tas bort och bilden skalas till komponentens verkliga bredd på skärmen. Detta möjliggör en konsekvent visuell representation av komponenter i monteringsvyn utan att vara beroende av externa bildkällor som CAD-modeller eller tillverkares mediamaterial.

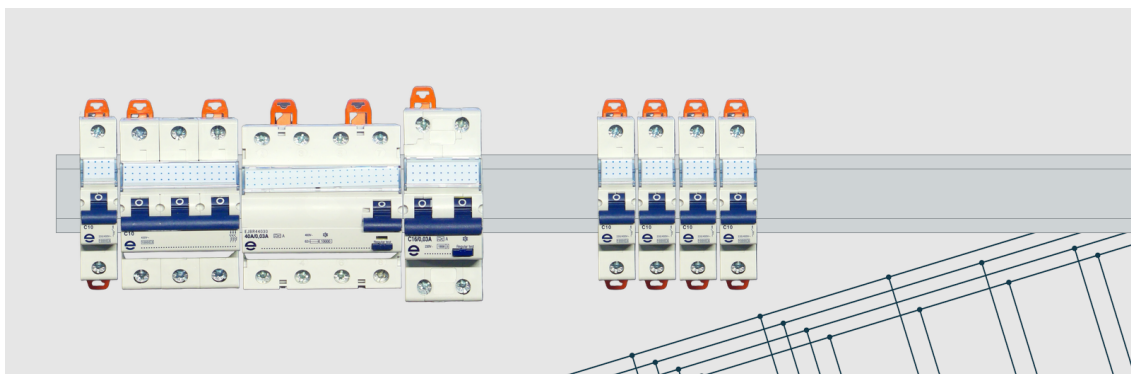
5.3.1 Bakgrunds borttagning

Figur 5.8 visar ett exempel på en komponent före och efter bearbetning med rembg, där bakgrunden ersatts med transparenta pixlar och bilden därefter beskurits till komponentens faktiska område med PIL för att ta bort transparenta pixlar som är kvar efter bakgrunds borttagning.



Figur 5.8: Komponent före och efter bakgrunds borttagning med rembg.

Resultatet visar att modellen korrekt identifierade och separerade komponenten från bakgrunden. Däremot introduceras lite brus runt om komponentens plast-fästen. Genom att testa olika modeller som fanns tillgängliga att använda inom rembg så fungerade bria-rembg bättre än BiRefNet-general modellen gällande kvarblivna artefakter, däremot betraktade den vissa starka skuggor som bakgrund. Detta visas i Figur 5.9.



Figur 5.9: En uppritning av komponenter för visuellt stöd under montering med nya komponenter som inte fanns vid stationen ursprungligen.

Resultatet visar att bakgrunds borttagning och skalningen fungerar bra nog för att möjliggöra uppritning av komponenterna som ska monteras. Det är inte perfekt, men ger ändå tillräcklig visuell feedback för att se ungefär hur den färdiga monteringen ska se ut. Detta tar bort behovet av att hitta mediabilder eller CAD-filer för specifika komponenter.

5.4 Verifiering av montage

Verifieringsdelen resulterade i ett funktionsbevis där komponenter på DIN-skenan kunde klassificeras utifrån ROI-bilder. Modellerna testades på Raspberry Pi 4:an för att undersöka både klassificeringsförmågan och praktisk prediktionstid.

5.4.1 Testade klassificeringsmodeller

Fyra modeller jämfördes i testet. De modeller som ingick visas i Tabell 4.2. Modellerna skiljde sig åt genom modellversion, bildstorlek och vilken datasetvariant som användes vid träning.

Resultatet av modellerna exporterade i PyTorch-format ges sammanfattningsvis i Tabell 5.1.

Tabell 5.1: Sammanfattat resultat från modelltester körda på Raspberry Pi 4 i PyTorch-format.

Kortnamn	Antal rätt top-1	Top-1 accuracy	Antal rätt inom top-2	Medel prediktionstid [ms]
YOLO11n 224	9	90%	10	295,18
YOLO26n 224	7	70%	10	294,52
YOLO26n 224 mörk	8	80%	10	291,80
YOLO26n 320 mörk	9	90%	10	395,13

Resultatet av modellerna exporterade till NCNN-format ges sammanfattningsvis i Tabell 5.2

Utifrån resultaten syns att export till NCNN-format gav kortare prediktionstid för samtliga modeller jämfört med PyTorch-formatet. Skillnaden var särskilt tydlig för

Tabell 5.2: Sammanfattat resultat från modelltester körda på Raspberry Pi 4 i NCNN-format.

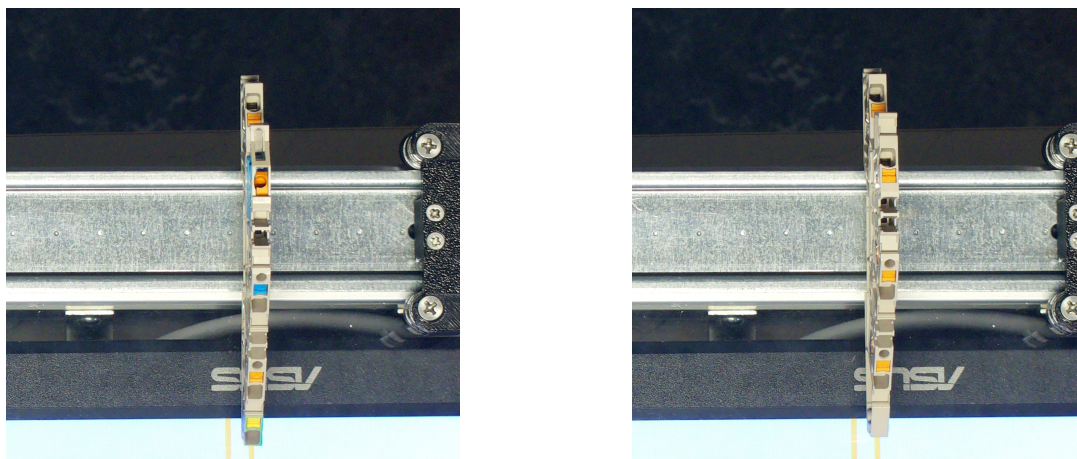
Kortnamn	Antal rätt top-1	Top-1 accuracy	Antal rätt inom top-2	Medel prediktionstid[ms]
YOLO11n 224	9	90%	10	265,86
YOLO26n 224	7	70%	10	232,03
YOLO26n 224 mörk	8	80%	10	242,75
YOLO26n 320 mörk	9	90%	10	277,41

YOLO26n mörk, där medeltiden minskade från 395,13 ms till 277,41 ms. Detta visar att modellformatet hade stor påverkan på den praktiska körningen på Raspberry Pi 4.

De modeller som gav högst top-1 accuracy var YOLO11n 224 och YOLO26n 320 mörk, där båda klassificerade 9 av 10 testbilder korrekt. Utifrån de fullständiga tabellerna med resultat i Bilagor C och D observeras också att oavsett vilket format som kördes, så predikterade modellen med samma säkerhet gällande top-1 och top-2 för testbilderna. Detta resultatet är däremot förväntat då exportprocessen bevarar modellens arkitektur och tränade vikter. Det som kan hända vid exportering är att minimala skillnader mellan modellerna uppstår i flyttalsprecision, vilket för denna appliceringen inte var tillräckligt stora för att påverka modellens slutgiltiga gissningar.

Ett intressant resultat är att samtliga modeller hade rätt komponent inom top-2 för alla testbilder. Det innebär att modellen i samtliga fall hade den korrekta komponenten bland sina två högst rankade alternativ, även när top-1-gissningen var fel. Detta tyder på att modellerna ofta kunde identifiera rätt komponentfamilj, men att det i vissa fall var svårt att välja just rätt artikelnummer som högsta gissning.

Felklassificeringarna uppstod främst för komponenter som var mycket lika varandra formmässigt. Ett förväntat svårt fall var komponenter som är fysiskt mycket lika och främst skiljer sig åt genom textmärkning, exempelvis C10- och C16-varianter. Mer oväntat var att två modeller gissade fel på plintar. I båda fallen gällde det bilden 2668950000_pos3.jpg, där modellerna YOLO26n 224 och YOLO26n 224 mörk gav 2668900000 som top-1 med med 99,6 % respektive 98,8 % confidence. Detta visar att datasetutökningen med mörkare kopior inte tillförde tillräckligt med ny information för att förbättra klassificeringen av just denna plint. Att YOLO11n 224 och YOLO26n 320 mörk klassificerade bilden rätt kan bero på skillnader i modellarkitektur eller på att den högre bildstorleken bevarade mer information om den smala plinten. Detta är dock endast en möjlig förklaring och skulle kräva djupare förståelse för den bakomliggande modellarkitekturen och bättre förståelse för dessa modeller för att utvärdera. I Figur 5.10 visas respektive bild i testdatan för dessa plintar.



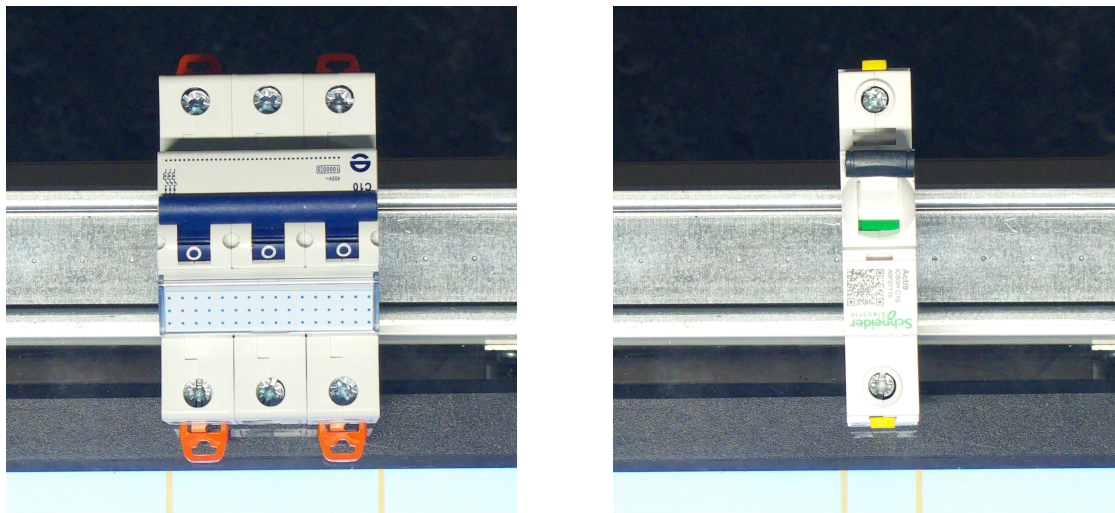
Figur 5.10: Visuellt liknande plintar, 2668950000 i vänster bild och 2668900000 i höger bild.

Plintar är de komponenter där beroendet av slutlig verifiering av klassificeringsmodellerna är högst, eftersom de inte har någon text som kan avläsas. Detta resultat gör det svårt att fullt ut förlita sig även på modellerna som klassificerade rätt på alla plintar.

För brytare inom samma komponentfamilj framkom en annan typ av osäkerhet. Även när modellen klassificerade rätt kunde top-1 gissningen ibland ligga kring 50–70 % confidence, samtidigt som top-2 ofta var en mycket liknande variant inom samma komponentfamilj. Detta syntes exempelvis för komponenter där skillnaden endast består av märkning. Resultatet tyder på att modellen i dessa fall ofta känner igen rätt komponentfamilj, men att den inte alltid har tillräckligt stark visuell grund för att säkert skilja mellan de olika varianterna.

5.4.2 OCR Komplettering

För komponenter inom samma komponentfamilj kan klassificeringsmodellen användas som ett första steg för att identifiera komponenttypen, med OCR-modellen används som komplement för att läsa av textmärkningen och bekräfta exakt variant. OCR-modellen utvärderades på träningsbilder för de två komponenterna i Tabell 4.4, vars bilder visas i Figur 5.11 bredvid varandra.



Figur 5.11: Testbilder för OCR-utvärdering, *EM103C10* till vänster och *A9F07110* till höger.

OCR-modellen kördes på en bild från Figur 5.11 för respektive komponent och resultatet efter filtrering sparades som en array. För *EM103C10* gav modellen

```
['C10', '400V', '100003']
```

och för *A9F07110* gavs

```
['A9F07110', 'IC60H', 'C10', 'Acti9']
```

För *A9F07110* återfanns hela artikelnumret direkt i resultatet, vilket innebär att en exakt matchning mot komponenten är möjlig utan vidare analys i databasen. Utöver artikelnumret identifieras *IC60H* och *C10*, vilket delvis stämmer överens med fälten Typ och Data i Tabell 4.4.

För *EM103C10* återfanns inget artikelnummer i resultatet, men nyckelorden *C10* och *400V* identifierades, vilket också delvis matchar fältet Data i databasen. Värdet 10003 bedöms vara ett tekniskt värde från komponenten som inte direkt kan matchas mot något fält i databasen och därför inte bidrar till verifieringen.

Resultaten visar att OCR-metoden har potential att komplettera klassificeringsmodellen för verifiering av komponentvarianter gällande olika brytare, men att matchningen mot databasen behöver hanteras med viss flexibilitet. För *A9F07110* räckte en direkt strängjämförelse mot artikelnumret, medan *EM103C10* kräver att nyckelord från Typ och Data används istället. Ett nyckelord som *C10* förekommer i databasen som exempelvis *C10A*, vilket innebär att en exakt strängjämförelse inte alltid är tillräcklig. En möjlig lösning är att konfigurera matchningen till att kontrollera om OCR-resultatet förekommer som en delsträng i databasens fält, snarare än att kräva exakt överensstämmelse.

En ytterligare försiktighet gäller nyckelord som *IC60H*, vilket identifierades för *A9F07110*. Detta är ett komponentfamiljenamn snarare än en variantspecifik egenskap, och skulle kunna förekomma på flera komponenter inom samma familj. Att

använda sådana nyckelord som avgörande för verifiering riskerar därför att ge missvisande resultat, och matchningen bör prioritera variantspecifika egenskaper såsom karaktistik och märkström framför familjenamn.

6

Diskussion

6.1 Diskussion av resultat

Resultatet visar att flera av de identifierade begränsningarna i den ursprungliga prototypen kunde hanteras genom mjukvaruutveckling. Lagerkonfigurationsverktyget minskar beroendet av hårdkodade konfigurationsfiler och gör det möjligt att beskriva lagerlayout, komponentplacering och LED-information på ett mer strukturerat sätt. Detta är ett steg mot en mer anpassningsbar station, även om verktyget fortfarande behöver kompletteras med fler kalibreringsmiljöer för att passa alla fysiska avvikelser i prototypen.

Ett exempel på detta är att LED-listen i den aktuella prototypen inte börjar med LED-index 0 vid lagerhyllans nollpunkt. Det innebär att den beräknade LED-mappningen behöver kunna justeras med en offset, eller en lista LED:s som inte bör tas hänsyn till vid LED-beräkningarna.

Verifieringsdelen visade att ROI-baserad klassificering kan fungera som ett proof of concept. Modellerna kunde klassificera sparade ROI-bilder på Raspberry Pi 4:an med en prediktionstid som kanske är lite för hög för att kunna ge feedback i realtid vid montering, då åtminstone 3-5 bilder i rad bör visa samma komponent för att vara säker. Samtidigt visar resultaten att lösningen inte är tillräckligt testad för att betraktas som produktionsnära. Framför allt krävdes fler tester med fler komponenter, och ett mer realistiskt monteringsflöde utifrån en ritning.

Däremot utvecklades ett strukturerat flöde för datainsamling, vilket möjliggör att snabbt kunna samla in bilder på nya komponenter som man vill använda vid stationen. Detta gör att man ständigt kan testa nya metoder för verifiering genom att exempelvis skapa utökade dataset eller lägga till fler positioner längs DIN-skenan där bilder ska tas. Detta resultat underlättar det framtida arbetet med datasetinsamling för den modellträning som används vid verifiering av montage.

6.2 Begränsningar

En tydlig begränsning i arbetet var att den kamerabaserade verifieringen inte hann integreras i ett komplett monteringsflöde. Testerna som genomfördes på Raspberry Pi 4:an gav ett underlag för modellernas klassificeringsförmåga och prestanda på

hårdvaran, men det visar inte att hela kedjan fungerar i realtid tillsammans med kameranoden, monteringsvyn och operatörens arbetsflöde.

Arbetet påverkades även av hårdvarubegränsningar. Vision-delen var beroende av att rätt kamera, lins och fysisk montering fanns tillgänglig. Eftersom kamerauppställningen ändrades i flera iterationer under arbetet behövde även datasetet, ROI-beräkning och bildinsamling justeras, samtidigt som det var stor osäkerhet angående vad för typ av resultat som kan förväntas få av en visions-baserad lösning. Detta bidrog till en tidsbrist i framtagningen av vision-modellen, eftersom flera tekniska problem behövde lösas, innan modellträningen kunde testas på ett meningsfullt sätt.

6.3 Etiska aspekter

En viktig etisk aspekt är hur mycket ansvar som bör läggas på ett automatiserat verifieringssystem. Om systemet felaktigt godkänner en komponent kan det ge operatören en falsk trygghet och leda till att ett monteringsfel inte upptäcks.

Därför bör kamerabaserad verifiering konkret vara påvisat att det fungerar i praktiskt produktion, med omfattande testning för att säkerställa att modellen är tillförlitlig för de komponenter och miljöer där den ska användas.

6.4 Licens och kommersiell användning

Ultralytics användes i arbetet eftersom ramverket gjorde det möjligt att snabbt träna, testa och exportera modeller. Detta passade bra för ett examensarbete där syftet var att undersöka möjligheten med kamerabaserad verifiering. Däremot behöver licensfrågan beaktas om lösningen ska användas kommersiellt. För kommersiell produktanvändning behöver företaget antingen säkerställa att användningen uppfyller AGPL-3.0-licensen eller teckna en kommersiell Enterprise License från Ultralytics.

I detta arbete accepterades användningen av Ultralytics som ett sätt att snabbt visa projektets tekniska möjligheter. Detta gjordes i samband med att förslag ges vid överlämning av arbete, av alternativ bildklassificeringsmodell och hur den kan användas för att uppnå liknande resultat. Övriga bibliotek och ramverk som användes i arbetet bedömdes ha licenser som tillåter kommersiell användning.

6.5 För vidare arbete

För lagerkonfigurationsverktyget bör nästa steg vara att lägga till stöd för att innan sparning, möjligtvis ange defekta eller icke-använda LED:s som då hoppas över i LED-beräkningen. Verktyget bygger i dagsläget en layout som liknar den hårdkodade variantens format, men saknar integrering av streckkoder som sitter på själva komponentlådan. Detta gör att man inte kan söka baklänges efter en lagerplats genom att skanna en låda. Ett möjligt förslag till en lösning är att antingen ha ett set-up läge där man ber användaren skanna lådorna för olika komponenter, eller be

användaren hålla upp sträckkoden nära kameran ifall det skulle räcka för att avkoda den.

För vision-delen bör nästa steg vara att integrera hela verifieringsflödet i monteringsflödet. Det innebär att kameranoden automatiskt behöver ta ROI-bilder för aktuell komponent, att modellen körs kontinuerligt eller vid rätt tillfälle, och att resultatet visas tydligt för operatören. Metoden bör därför testas med fler komponenter och fler verkliga monteringsritningar.

Vid tätt placerade komponenter bör även metoder för att avgränsa ROI-bilden undersökas, exempelvis genom att maskera bort bildområden utanför komponentens förväntade bredd för att behålla kvadratisk kameraupplösning och detaljer på komponenten som ska verifieras.

Ett ytterligare alternativ för vidare arbete för att komma närmare en produktionsredo version är att använda en fullständig YOLO objekt-detektionsmodell. Detta bidrar till att träningsdatasetet behöver bearbetas manuellt innan träning, men vid diskussioner under projektet, anses det inte riktigt vara en nackdel om det kan göras endast en gång på rimlig tid för att få en modell som fungerar i verkligheten. En fullständig YOLO modell, hade gjort det möjligt att både lokalisera komponenter, men också klassificera med en helskrämsbild, vilket undviker beroendet av ständig kommunikation gällande vilken ROI som den förväntade komponenten bör ligga inom. Detta möjliggör även att ge feedback till montör vid fallet av att en komponent monteras långt utanför det förväntade området.

OCR bör också utvärderas ytterligare som komplement till klassificeringsmodellen, eller eventuell detektionsmodell. Detta innebär att skapa en tydlig koppling mellan nyckelord som kan framkomma på komponenter i databasen, så att det blir en pålitlig verifiering. Klassificeringsmodellen kan då användas för att identifiera komponentfamiljen, medan OCR används för att kontrollera exakt märkning.

Slutligen bör ett framtida system ha stöd för fjärruppdatering och versionshantering. Detta gäller både programkod, lagerkonfigurationer och tränade modeller. Git rekommenderas som grund för versionshantering av programkoden, medan stations-specifika konfigurationer och modellfiler bör versionshanteras separat så att de kan uppdateras eller återställas utan att påverka andra stationer.

7

Slutsats

Monteringsstationen har utvecklats till att bli mer skalbar och anpassningsbar för olika kundmiljöer. Lagerkonfigurationsverktyget möjliggör att man enkelt kan konfigurera lagerlayout och komponentplacering, och tillsammans med Teach Mode skapas ett strukturerat flöde för datainsamling. Bilderna som tas på komponenterna kan sedan användas för modellträning, men också för uppritning på skärmen för visuellt stöd under montering.

För verifiering av montage visade arbetet att ROI-baserad bildklassificering med YOLO fungerar som ett proof of concept för enskilda komponenter på den begränsade hårdvaran. Användning av OCR utvärderades som ett komplement för att särskilja brytare inom samma familj. Tillsammans utgör dessa en lovande grund i form av att YOLO kan ge snabb feedback till operatören gällande vad som har monterats, medan OCR i bakgrunden kan verifiera exakt komponentmodell.

Med ett tillförlitligt verifieringsflöde är steget till en station med ett komplett monteringsflöde inte långt.

Litteraturförteckning

- [1] Elloumi, K., Triki, H., Zghal, B. & Chebbi, W. Integration of Pick by Light in the Context of Learning Factories (Assembly Workstation 4.0). Springer Nature, 349–355 (2024). doi: 10.1007/978-3-031-67152-4_37
- [2] Worldsemi. WS2812B-2020 Intelligent control LED integrated light source. Nedladdad från Electrokit. https://www.electrokit.com/upload/quick/b2/35/36bd_WS2812B-2020.pdf (Hämtad 2026-06-14)
- [3] Raspberry Pi Ltd. Raspberry Pi Documentation. <https://www.raspberrypi.com/documentation/> (Hämtad 2026-04-10)
- [4] Shafranovich, Y. RFC4180: Common Format and MIME for Comma Separated Values (CSV) Files. RFC 4080 (2005) doi: 10.17487/RFC4180
- [5] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). doi: 10.1038/nature14539
- [6] Ultralytics. Ultralytics docs. <https://docs.ultralytics.com/> (Hämtad 2026-05-18)
- [7] Ultralytics. Datasets overview. <https://docs.ultralytics.com/datasets> (Hämtad 2026-05-18)
- [8] Ultralytics. Data Augmentation using Ultralytics YOLO. <https://docs.ultralytics.com/guides/yolo-data-augmentation> (Hämtad 2026-05-18)
- [9] PyTorch. PyTorch Documentation. <https://docs.pytorch.org/docs/2.12/index.html> (Hämtad 2026-06-10)
- [10] Ni, H. et al. ncnn (2017). <https://github.com/tencent/ncnn> (Hämtad 2026-06-10)
- [11] Cui, C. et al. PaddleOCR 3.0 Technical Report. arXiv:2507.05595 (2025). doi: 10.48550/arXiv.2507.05595
- [12] Sveriges Elgrossister. E-nummerbanken, Regelverk. <https://www.seg.se/e-nummerbanken/regelverk/> (Hämtad 2026-05-05)
- [13] Raspberry Pi Ltd. The Picamera2 Library. <https://pip-assets.raspberrypi.com/categories/652-raspberry-pi-camera-module->

2/documents/RP-008156-DS-2-picamera2-manual.pdf?disposition=inline
(Hämtad 2026-05-16)

- [14] Gatis, D. rembg (2025). <https://github.com/danielgatis/rembg> (Hämtad 2026-06-10)

A

Träningsskript för slutlig modell

```
from ultralytics import YOLO
model = YOLO('yolo26n-cls.pt') # ladda modell
model.train(
    data=r'DATA_DIR',
    epochs=200,
    imgsz=320,
    batch=8,
    dropout=0.2,
    translate=0.02,
    hsv_v=0.1,
    degrees=7,
    patience=40,
    mosaic=0.0,
    erasing=0.0,
    scale=0.0,
    flip_lr=0.0,
    hsv_s=0.0,
    crop_fraction = 1.0,
    auto_augment=None,
    name = "NAMN"
)
```

B

Exempel på sparad lagerkonfiguration

Nedan visas ett exempel på hur en konfigurationsfil kan se ut. För artiklar som befinner sig i lagerzoner med pick-to-light på, fås en start LED-index, samt antal som ska tändas i följd inklusive startindexet.

```
{
  "artiklar": {
    "2668900000": {
      "hylla": "fram",
      "rad": 1,
      "kolumn": 3,
      "ptl": true,
      "start_led": 55,
      "antal_led": 9
    },
    "2668950000": {
      "hylla": "fram",
      "rad": 1,
      "kolumn": 2,
      "ptl": true,
      "start_led": 46,
      "antal_led": 9
    },
    "2669140000": {
      "hylla": "fram",
      "rad": 1,
      "kolumn": 0,
      "ptl": true,
      "start_led": 28,
      "antal_led": 9
    },
    "2690440000": {
      "hylla": "fram",
      "rad": 1,
      "kolumn": 1,
```

```
    "ptl": true ,
    "start_led": 37,
    "antal_led": 9
  },
  "A9F06125": {
    "hylla": "fram",
    "rad": 0,
    "kolumn": 3,
    "ptl": true ,
    "start_led": 0,
    "antal_led": 7
  },
  "A9F06120": {
    "hylla": "fram",
    "rad": 0,
    "kolumn": 2,
    "ptl": true ,
    "start_led": 7,
    "antal_led": 7
  },
  "A9F06116": {
    "hylla": "fram",
    "rad": 0,
    "kolumn": 1,
    "ptl": true ,
    "start_led": 14,
    "antal_led": 7
  },
  "A9F06113": {
    "hylla": "fram",
    "rad": 0,
    "kolumn": 0,
    "ptl": true ,
    "start_led": 21,
    "antal_led": 7
  },
  "EM103C10": {
    "hylla": "hoger",
    "rad": 0,
    "kolumn": 0,
    "ptl": false
  },
  "EM101C20": {
    "hylla": "hoger",
    "rad": 0,
    "kolumn": 1,
```

```
    "ptl": false  
  }  
}
```

C

Modellresultat på Pi 4 med PyTorch format

Pos motsvarar testnummer i Tabell 4.3.

image	model	top1	top1_conf	top2	top2_conf	inference_ms
2668900000_pos5.jpg	yolo11n_cls_224.pt	2668900000	0.9626	2668950000	0.0373	325.93
2668900000_pos5.jpg	yolo26n_cls_224.pt	2668900000	1.0000	2668950000	0.0000	317.32
2668900000_pos5.jpg	yolo26n_cls_224_dark.pt	2668900000	0.9996	2668950000	0.0002	293.63
2668900000_pos5.jpg	yolo26n_cls_320_dark.pt	2668900000	0.9316	2668950000	0.0679	397.26
2668950000_pos3.jpg	yolo11n_cls_224.pt	2668950000	0.9858	2668900000	0.0138	294.79
2668950000_pos3.jpg	yolo26n_cls_224.pt	2668900000	0.9960	2668950000	0.0038	293.25
2668950000_pos3.jpg	yolo26n_cls_224_dark.pt	2668900000	0.9877	2668950000	0.0107	295.78
2668950000_pos3.jpg	yolo26n_cls_320_dark.pt	2668950000	0.9999	2668900000	0.0001	397.96
2669140000_pos6.jpg	yolo11n_cls_224.pt	2669140000	0.9989	EM101C16	0.0005	294.08
2669140000_pos6.jpg	yolo26n_cls_224.pt	2669140000	0.9998	EM101C10	0.0000	294.07
2669140000_pos6.jpg	yolo26n_cls_224_dark.pt	2669140000	0.9992	EM101C16	0.0003	295.22
2669140000_pos6.jpg	yolo26n_cls_320_dark.pt	2669140000	0.9999	EM101C16	0.0001	396.94
A9F07110_pos7.jpg	yolo11n_cls_224.pt	A9F07110	0.8721	A9F07116	0.0712	292.14
A9F07110_pos7.jpg	yolo26n_cls_224.pt	A9F07110	0.8903	A9F07116	0.1088	294.28
A9F07110_pos7.jpg	yolo26n_cls_224_dark.pt	A9F07110	0.9603	A9F07116	0.0394	292.28
A9F07110_pos7.jpg	yolo26n_cls_320_dark.pt	A9F07110	0.9957	A9F07116	0.0041	395.48
A9F07316_pos1.jpg	yolo11n_cls_224.pt	A9F07316	0.9897	A9Z21440	0.0080	289.64
A9F07316_pos1.jpg	yolo26n_cls_224.pt	A9F07316	0.9989	A9Z21440	0.0006	288.17
A9F07316_pos1.jpg	yolo26n_cls_224_dark.pt	A9F07316	0.9996	A9Z21440	0.0002	288.84
A9F07316_pos1.jpg	yolo26n_cls_320_dark.pt	A9F07316	0.9966	A9Z21440	0.0023	391.05
A9Z21440_pos0.jpg	yolo11n_cls_224.pt	A9Z21440	0.9996	A9F07316	0.0001	283.50
A9Z21440_pos0.jpg	yolo26n_cls_224.pt	A9Z21440	0.9992	A9F07316	0.0006	286.86
A9Z21440_pos0.jpg	yolo26n_cls_224_dark.pt	A9Z21440	0.9982	A9F07316	0.0011	283.75
A9Z21440_pos0.jpg	yolo26n_cls_320_dark.pt	A9Z21440	0.9996	A9F07316	0.0003	388.00
EM101C10_pos8.jpg	yolo11n_cls_224.pt	EM101C10	0.9071	EM101C16	0.0695	292.73
EM101C10_pos8.jpg	yolo26n_cls_224.pt	EM101C16	0.8947	EM101C10	0.1012	292.95
EM101C10_pos8.jpg	yolo26n_cls_224_dark.pt	EM101C16	0.7901	EM101C10	0.2052	293.20
EM101C10_pos8.jpg	yolo26n_cls_320_dark.pt	EM101C10	0.7264	EM101C16	0.2668	396.62
EM103C10_pos2.jpg	yolo11n_cls_224.pt	EM103C10	0.9976	EM103C16	0.0019	294.69
EM103C10_pos2.jpg	yolo26n_cls_224.pt	EM103C16	0.5601	EM103C10	0.4327	293.98
EM103C10_pos2.jpg	yolo26n_cls_224_dark.pt	EM103C10	0.7674	EM103C16	0.2289	291.96
EM103C10_pos2.jpg	yolo26n_cls_320_dark.pt	EM103C10	0.9997	EM103C16	0.0002	396.50
EM103C16_pos4.jpg	yolo11n_cls_224.pt	EM103C10	0.5609	EM103C16	0.4359	291.88
EM103C16_pos4.jpg	yolo26n_cls_224.pt	EM103C16	0.9988	EM103C10	0.0008	291.40
EM103C16_pos4.jpg	yolo26n_cls_224_dark.pt	EM103C16	0.9991	EM103C10	0.0006	290.23
EM103C16_pos4.jpg	yolo26n_cls_320_dark.pt	EM103C10	0.6659	EM103C16	0.3326	395.45
EPBR216C30_pos9.jpg	yolo11n_cls_224.pt	EPBR216C30	0.9977	EM103C10	0.0009	292.46
EPBR216C30_pos9.jpg	yolo26n_cls_224.pt	EPBR216C30	0.9988	EM103C16	0.0004	292.91
EPBR216C30_pos9.jpg	yolo26n_cls_224_dark.pt	EPBR216C30	0.9924	EM103C16	0.0043	293.11
EPBR216C30_pos9.jpg	yolo26n_cls_320_dark.pt	EPBR216C30	0.9961	EM103C16	0.0026	396.00

D

Modellresultat på Pi 4 med NCNN format

Pos motsvarar testnummer i Tabell 4.3.

image	model	top1	top1_conf	top2	top2_conf	inference_ms
2668900000_pos5.jpg	yolo11n_cls_224_ncnn_model	2668900000	0.9626	2668950000	0.0373	268.35
2668900000_pos5.jpg	yolo26n_cls_224_ncnn_model	2668900000	1.0000	2668950000	0.0000	252.47
2668900000_pos5.jpg	yolo26n_cls_224_dark_ncnn_model	2668900000	0.9996	2668950000	0.0002	231.24
2668900000_pos5.jpg	yolo26n_cls_320_dark_ncnn_model	2668900000	0.9316	2668950000	0.0679	276.17
2668950000_pos3.jpg	yolo11n_cls_224_ncnn_model	2668950000	0.9858	2668900000	0.0138	266.21
2668950000_pos3.jpg	yolo26n_cls_224_ncnn_model	2668900000	0.9960	2668950000	0.0038	234.06
2668950000_pos3.jpg	yolo26n_cls_224_dark_ncnn_model	2668900000	0.9877	2668950000	0.0107	293.57
2668950000_pos3.jpg	yolo26n_cls_320_dark_ncnn_model	2668950000	0.9999	2668900000	0.0001	276.37
2669140000_pos6.jpg	yolo11n_cls_224_ncnn_model	2669140000	0.9989	EM101C16	0.0005	318.45
2669140000_pos6.jpg	yolo26n_cls_224_ncnn_model	2669140000	0.9998	EM101C10	0.0000	231.57
2669140000_pos6.jpg	yolo26n_cls_224_dark_ncnn_model	2669140000	0.9992	EM101C16	0.0003	238.72
2669140000_pos6.jpg	yolo26n_cls_320_dark_ncnn_model	2669140000	0.9999	EM101C16	0.0001	269.52
A9F07110_pos7.jpg	yolo11n_cls_224_ncnn_model	A9F07110	0.8721	A9F07116	0.0712	261.38
A9F07110_pos7.jpg	yolo26n_cls_224_ncnn_model	A9F07110	0.8903	A9F07116	0.1088	234.26
A9F07110_pos7.jpg	yolo26n_cls_224_dark_ncnn_model	A9F07110	0.9603	A9F07116	0.0394	234.26
A9F07110_pos7.jpg	yolo26n_cls_320_dark_ncnn_model	A9F07110	0.9957	A9F07116	0.0041	316.24
A9F07316_pos1.jpg	yolo11n_cls_224_ncnn_model	A9F07316	0.9897	A9Z21440	0.0080	249.58
A9F07316_pos1.jpg	yolo26n_cls_224_ncnn_model	A9F07316	0.9989	A9Z21440	0.0006	226.33
A9F07316_pos1.jpg	yolo26n_cls_224_dark_ncnn_model	A9F07316	0.9996	A9Z21440	0.0002	233.06
A9F07316_pos1.jpg	yolo26n_cls_320_dark_ncnn_model	A9F07316	0.9966	A9Z21440	0.0023	264.83
A9Z21440_pos0.jpg	yolo11n_cls_224_ncnn_model	A9Z21440	0.9996	A9F07316	0.0001	249.86
A9Z21440_pos0.jpg	yolo26n_cls_224_ncnn_model	A9Z21440	0.9992	A9F07316	0.0006	221.28
A9Z21440_pos0.jpg	yolo26n_cls_224_dark_ncnn_model	A9Z21440	0.9982	A9F07316	0.0011	225.39
A9Z21440_pos0.jpg	yolo26n_cls_320_dark_ncnn_model	A9Z21440	0.9996	A9F07316	0.0003	308.36
EM101C10_pos8.jpg	yolo11n_cls_224_ncnn_model	EM101C10	0.9071	EM101C16	0.0695	265.67
EM101C10_pos8.jpg	yolo26n_cls_224_ncnn_model	EM101C16	0.8947	EM101C10	0.1012	229.68
EM101C10_pos8.jpg	yolo26n_cls_224_dark_ncnn_model	EM101C16	0.7901	EM101C10	0.2052	233.75
EM101C10_pos8.jpg	yolo26n_cls_320_dark_ncnn_model	EM101C10	0.7264	EM101C16	0.2668	268.63
EM103C10_pos2.jpg	yolo11n_cls_224_ncnn_model	EM103C10	0.9976	EM103C16	0.0019	256.39
EM103C10_pos2.jpg	yolo26n_cls_224_ncnn_model	EM103C16	0.5601	EM103C10	0.4327	226.70
EM103C10_pos2.jpg	yolo26n_cls_224_dark_ncnn_model	EM103C10	0.7674	EM103C16	0.2289	227.47
EM103C10_pos2.jpg	yolo26n_cls_320_dark_ncnn_model	EM103C10	0.9997	EM103C16	0.0002	263.13
EM103C16_pos4.jpg	yolo11n_cls_224_ncnn_model	EM103C10	0.5609	EM103C16	0.4359	258.06
EM103C16_pos4.jpg	yolo26n_cls_224_ncnn_model	EM103C16	0.9988	EM103C10	0.0008	234.68
EM103C16_pos4.jpg	yolo26n_cls_224_dark_ncnn_model	EM103C16	0.9991	EM103C10	0.0006	281.40
EM103C16_pos4.jpg	yolo26n_cls_320_dark_ncnn_model	EM103C10	0.6659	EM103C16	0.3326	264.75
EPBR216C30_pos9.jpg	yolo11n_cls_224_ncnn_model	EPBR216C30	0.9977	EM103C10	0.0009	264.60
EPBR216C30_pos9.jpg	yolo26n_cls_224_ncnn_model	EPBR216C30	0.9988	EM103C16	0.0004	229.22
EPBR216C30_pos9.jpg	yolo26n_cls_224_dark_ncnn_model	EPBR216C30	0.9924	EM103C16	0.0043	228.60
EPBR216C30_pos9.jpg	yolo26n_cls_320_dark_ncnn_model	EPBR216C30	0.9961	EM103C16	0.0026	266.07

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se



CHALMERS