

TPR interface

- Brugerflade til robotiseret plukning af klasetomater
-

NP

Nicklas Petersen, (070197)
nipet17@student.sdu.dk

2/1-21

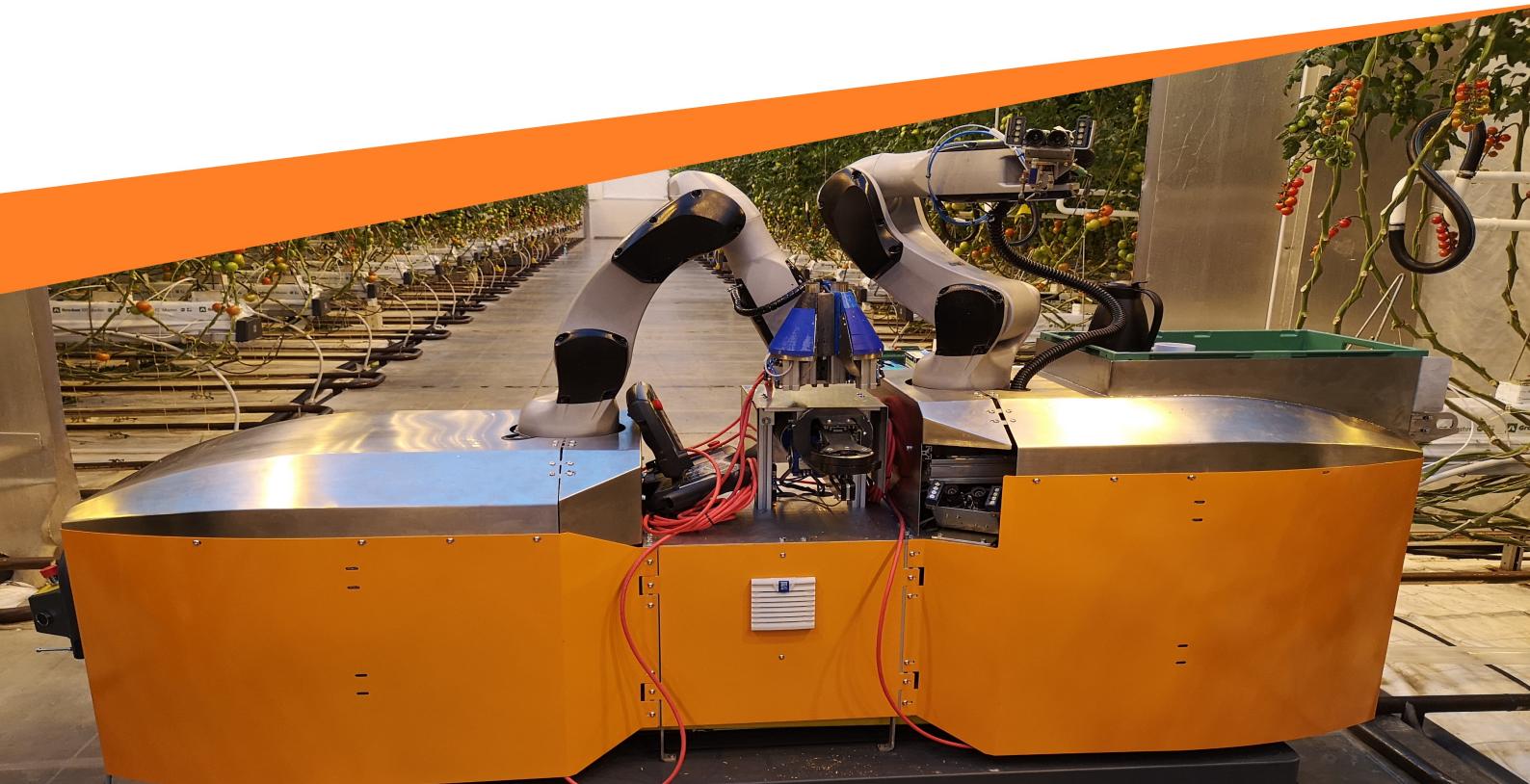
Dato

Bachelor projekt· 30 ECTS

Syddansk universitet, SDU

Vejleder · Jakob Wilm

Teknisk fakultet



TITELBLAD

Project titel	Robotiseret plukning af klasetomater – Brugerflade
Projekt periode	1. September 2020 - 4. Januar 2021
Uddannelsessted	Syddansk universitet, Odense, Teknisk fakultet
Uddannelse	Diplomingeniør Robotteknologi
Deltagere	Nicklas Petersen
Vejleder	Jakob Wilm
Side antal	70
Antal bilag	7
Abstract	<p>The focus of this project is the construction of a user interface, to control the TPR project.</p> <p>The user interface must have two user levels so that restrictions can be made for certain functionalities.</p> <p>The user interface is created as a website so that it can be accessed through a browser. For the user interface, a server with an associated database was created in order for data to be logged for further analysis.</p> <p>The robot system has been changed so that control through the user interface is possible.</p> <p>If a robot in the robotic system fails, it is possible to accept the error and continue the picking sequence. Communication between the robot-system and the PLC happens through a MQTT connection.</p> <p>With the use of sequential contrl from the interface, it is possible to execute stereo calibration of the vision cameras.</p> <p>An interface, that can control multiple aspects of the TPR robot-system, has been created. The product works as intended, however it could be improved with more time.</p>

FORORD

Denne rapport er udført, som et 30 ECTS bachelorprojekt ved Teknisk Fakultet, Diplomingeniør Robotteknologi, Syddansk Universitet. Projektet er en sammenfatning af kompetencer opnået i løbet af uddannelsens forløb. Projektet er en udarbejdelse af en webbaseret brugerflade til en automatisk tomatplukker, udarbejdet af Egatec, Dansk Teknologisk Institut, Alfred Pedersen & Søn og Bogaerts Greenhouse Logistics. Den primære udvikling af Tomat plukkeren udføres hos Egatec, mens test foregår hos Alfred Pedersen & Søn. Vejledere på projektet har været lektor Jakob Wilm ved SDU Robotics og Bjarke Colbe programmør hos Egatec.

Jeg vil gerne takke Egatec og hele Egamatic konsortiet for hjælp og opbakning i forbindelse med projektet. Det har været en stor og lærerig oplevelse at være en del af TPR projektet. Ydermere vil jeg takke Beckhoff ApS for support og assistance med kommunikation.

Som udgangspunkt antages det, at læseren har et tilsvarende teknisk kompetenceniveau, som en studerende på 7. semester på diplomingeniør uddannelsen i robotteknologi. Dette betyder, at teknisk grundlæggende viden ikke vil yderligere forklares.

Rapporten er opbygget således, at den læses fra start til slut. Rapportens struktur er opdelt i hovedsektioner og undersektioner.

Hvis nogle forkortelser eller symboler ikke intuitivt giver mening, kan en beskrivelse findes som fodnote i bunden af siden. Kilder der fremstår i rapporten er sammenfattet i litteraturlisten i slutningen af rapporten. Referencer er organiseret ved et [nummer], hvilket fører til litteraturlisten, hvor artikler er organiseret i forhold til forfatter, titel og forlægger. Internetsider er indikeret med en URL, forfatter (hvis stillet til rådighed), titel og dato for besøg.

Alle bilag findes i medfølgende zip mappe. Kode til robot-systemet vil ikke medfølge, da denne ejes af Egatec. Flowcharts og billeder af brugerfladen findes i større format i bilag.

INDHOLDSFORTEGNELSE

Titelblad	i
Forord	iii
Figurer	vii
1 INTRODUKTION	1
1.1 TPR projektet	1
1.2 Problemformulering	4
1.2.1 Problemstilling	4
1.3 Målsætning	4
1.4 Projektgrænsning	5
2 FORVENTNINGER	7
2.1 Vigtige forventninger	7
2.2 Nyttige forventninger	7
3 KONCEPT	9
3.1 Robot-system	9
3.1.1 Vogn	10
3.1.2 Lineær gribearm	10
3.1.3 Artikulerede robotarme	11
3.1.4 Vision system	11
3.1.5 Bearbejdelse af tomater	13
3.2 Brugerflade	17
3.2.1 Model	18
3.2.2 View	18
3.2.3 Controller	19
3.3 Kommunikation	19
3.3.1 CANopen	21
3.3.2 ADS-route	22
3.3.3 MQTT	22
4 IMPLEMENTERING	25
4.1 Hardware	25
4.2 Software	26
4.3 Brugerflade	26
4.3.1 Applikation	27
4.3.2 Public	41
4.4 Fjernstyring af robot-system	45
4.4.1 Kommunikation	45
4.4.2 Blok-styring	48

4.5	Design	51
4.6	Sikkerhed	54
4.6.1	Interface	54
4.6.2	Fysisk sikkerhed og præservering af robotter	55
5	ALTERNATIVER	57
5.1	Brugerflade	57
5.1.1	Monteret brugerflade	57
5.1.2	Kablet brugerflade	58
5.2	Kommunikation	58
5.2.1	OPC-UA	58
5.2.2	UDP	59
5.2.3	TCP-IP	59
6	DISKUSSION	61
6.1	Projekt udfordringer	63
7	KONKLUSION	65
8	FREMTIDIGE IMPLEMENTERINGER	67
8.1	Risikovurdering	67
8.2	Asynkron abonnement	67
8.3	Visning af vision billeder	67
8.4	Upload af data til stationær database	68
8.5	Vedligeholdelse af TPR	68
9	LITTERATUR	69

FIGURER

Fig. 1	TPR Robot-system	2
Fig. 2	Bogaerts GL Qii-Lift H-350 pipe rail trolley	10
Fig. 3	Gribearm designet af Bogaerts GL	10
Fig. 4	Robot Pick værktøj med kameraæt	11
Fig. 5	Eksempel på kalibrerings plade	12
Fig. 6	”Retrieve stem” sekvens	14
Fig. 7	”Grab stem” sekvens	14
Fig. 8	”Move on stem” sekvens	15
Fig. 9	”Cut tomatoes” sekvens	16
Fig. 10	Tomato packing sekvens	17
Fig. 11	MVC struktur	18
Fig. 12	Kommunikationskoncept	20
Fig. 13	Master/Slave terminologi	22
Fig. 14	Publish/Subscribe	23
Fig. 15	Topic struktur	23
Fig. 16	MQTT broker	24
Fig. 17	RevPi Connect	25
Fig. 18	Adjust recipe view	33
Fig. 19	Setup view	34
Fig. 20	Kalibrering ved brug af modals	37
Fig. 21	Valg af kalibreringsform	38
Fig. 22	Menu	39
Fig. 23	Dropdown menu	40

Fig. 24	URL eksempel	41
Fig. 25	Flowdiagram ved kald af MVC side	42
Fig. 26	EasySql	45
Fig. 27	Robotter til venteposition	49
Fig. 28	Alarm håndtering	50
Fig. 29	Login side	52
Fig. 30	Manuel side	52
Fig. 31	Alarm side	53
Fig. 32	Referencehak led 1	56

BILAG

BILAG 1	Kode til brugerflade (Inkl.biblioteker)
BILAG 2	Video af sekventiel kamera kalibrering
BILAG 3	Billeder af interface i landskab
BILAG 4	Billeder af interface i portræt
BILAG 5	Flowcharts
BILAG 6	Uddrag til patentsøgning TPR projekt
BILAG 7	Tidsplan

INTRODUKTION

Formålet med dette kapitel er at give et overblik over TPR¹ projektet, give en beskrivelse af det nuværende robot-system og dets virkemåde.

1.1 TPR PROJEKTET

Gartneriindustrien er hårdt ramt af mangel på arbejdskraft. I en artikel hos Greenhouse Grower fra November 2018 beskrives det, at det bliver sværere og sværere at skaffe folk, der er villig til at arbejde i drivhuse[8].

- “There just aren’t enough people willing to work in a greenhouse.”
- “Labor costs coupled with the availability of skilled labor are currently our largest concerns. Each year it becomes more difficult to obtain qualified labor and each year the skill level of the existing labor pool drops while the demand for higher wages increases.”
- “Labor issues are the most troublesome, and they seem to be getting worse.”
- “I purchased a 39-year-old business last year and don’t have the middle management in place that understands the entire business to help run the place.”
- “Labor and immigration are difficult subjects that seem to never go away or improve.”
- “Labor costs are forcing us to change our business. We are looking at mechanization, but do not know if we will be able to pay for the investment.”

Som man kan se i ovenstående citater[8], er det et stort problem at finde kompetent arbejdskraft, der er villige til at arbejde i et drivhus. Derudover stiger procentdelen af udgifter til lønninger. Af denne grund, er der stor interesse for automation i drivhus-industrien.

¹ TPR = Tomato Picking Robot

TPR projektet startede for omkring 12 år siden, hvor Egatec skulle lave en automatisk plukker til Alfred Pedersen og Søn's drivhuse. Egatec udviklede en mekanisk løsning, der kunne bevæge sig i X-Y-Z retning. Denne løsning blev ingen succes grundet diversiteten på levende planter. Trods mangel på succes havde Egatec udviklet et værktøj til bearbejdelse af planter, som de valgte at patentere sammen med arbejdsmetoden, hvorpå planterne bearbejdes nedefra.

I løbet af de næste prototyper opstod et samarbejde mellem Egatec, Bogaerts Greenhouse Logistics, der fik udviklet en gribearm, til at trække planter ind til maskinen og Dansk Teknologisk Institut, der udvikler vision system til detektionen af tomater. Derudover grundlægges virksomheden Egamatic bestående af Egatec, Alfred Pedersen og Søn, Bogaerts GL og Senmatic.

I dag er TPR projektet støttet af EU fonden "Eurostar", hvor målet er, at få udviklet et produkt, der automatisk kan plukke tomater i gartnerier. Det primære fokus ved dette projekt er at automatisere plukning og pakning af cherry tomater.

Projektet bliver udarbejdet i samarbejde med Alfred Pedersen & Søn, Bogaerts Greenhouse Logistics, Dansk Teknologisk Institut og Egatec.

Den nyeste version af TPR projektet er et robot-system bestående af en vogn udviklet af Bogaerts GL, der kan køre på varmerør mellem tomat rækkerne i drivhuse, et vision system, der udvikles af DTI, to Comau Racer 5 robotarme, og PLC styring udarbejdet af Egatec.



Fig. 1: TPR Robot-system

I figur 1 ses TPR robotten. Robotten består af en vogn fra Bogaerts GL (markeret med lyserød), der kan køre på varmerør i drivhuse. Markeret med gul er to robotarme med tilhørende værktøj til håndtering af tomatplanter. Disse robotarme bliver styret af hver sin robot-kontroller, der befinder sig bag de sorte markeringer. Bag den blå markering findes en kompressor, der leverer lufttryk til styring af værktøj. Med brun markering ses en lineær gribearm, der benyttes til at trække tomatplanter til robot-systemet.

To vision kamerasæt er markeret med lilla. Disse kameraer bliver styret fra en industri-computer, der er gemt bag den grønne markering.

Bag de tyrkis markeringer findes PLC, I/O forbindelser og en industriel Raspberry Pi, hvorpå brugerfladen befinder sig.

Styring til robotsystemet er delt op i tre emner:

- Vision system med deeplearning classifiers til detektion af tomat, stilke og plante
- Robot og maskine-styring
- Brugerinterface

Vision systemet, der detekterer tomater, stilke og tomatplanter, bliver udarbejdet af DTI Robotics. Robot og maskine-styring udvikles hos Egatec.

Denne rapport vil fokusere på udarbejdelsen og implementering af brugerfladen til styring af robot-systemet. For, at brugerfladen kan kontrollere robotsystemet, vil der ydermere lægges fokus på både robot og PLC styring.

1.2 PROBLEMFORMULERING

Målet med projektet er at anvende opnået teoretisk og praktisk viden fra udannelsen Diplomingeniør i Robotteknologi, til at implementere og udarbejde en web-baseret brugerflade til Egamatic tomat-plukker-robot. Ved hjælp af denne brugerflade, skal det være muligt, at kontrollere virkemåden af robotsystem, alarmhåndtering, samt styre robotsystemets konfigurationer og vigtige information, som pakkemønstre og værktøjsstyring.

Systemet skal kunne benyttes, til opsamling af information om plukning, hvilket kan benyttes af kommende brugere af robotsystemet.

Da der skal udarbejdes en webbaseret brugerflade til styring af robotsystemet, skal virkemåden af robotsystemet ændres, således at der er mulighed for styring gennem et bruger-interface. Til dette skal der oprettes kommunikation til den PLC, der styrer robotsystemet. Til slut skal der udvikles software således at data, der kan være brugbare i analyser, kan logges.

1.2.1 *Problemstilling*

- Hvordan udarbejdes kommunikation mellem PLC og webinterface?
- Hvordan konstrueres en brugerflade med forskellige bruger-niveauer og restriktioner?
- Hvordan skal robotsystemet håndtere input fra brugerfladen?
- Hvordan ændres robot konfigurationer vha. input fra en brugerflade?
- Hvordan struktureres PLC og Robot styring, for at styring gennem en brugerflade bliver muligt?
- Hvordan kan data logges, så senere databehandling muliggøres?

1.3 MÅLSÆTNING

Målet med dette projekt er at udarbejde en webbaseret brugerflade, der kan tilgås fra alle enheder i samme netværk. Vha. bruger-niveauer skal det være muligt at definere begrænset adgang til funktionaliteter.

Målet med produktet er, at det ikke er nødvendigt at benytte en programmør med adgang til PLC og robotter, for at kunne styre robot-systemet.

1.4 PROJEKTAFGRÆNSNING

I projektet fokuseres der på, at udvikle en brugerflade, der kan kontrollere TPR robot-systemet. Der er valgt, at fokusere på, at etablere forbindelse til robot-systemet og sende simple kommandoer. Der vil ikke fokuseres på, at ændre virkemåden af robot-systemet. Der skal udarbejdes alarmhåndtering. Der vil fokuseres på at registrere og logge alarmer, og finde ud af hvor kritiske alarmerne er. Håndtering af alarmerne vil ikke være i fokus, da der findes omkring 8000 forskellige alarmer til robotterne.

Brugerfladen skal kunne hente data fra robot-systemet. Der vil ikke fokuseres på, at data skal hentes og fortolkes efter brugerfladen er loaded. For at udføre backend programmering, efter at brugerfladen er loaded, skal nodeJS benyttes. Grundet projektets længde, vil det derfor ikke være i fokus.

2

FORVENTNINGER

At udarbejde både forventninger og begrænsninger for et produkt sikrer, at resultatet, sammenlignet med ønskerne, bliver så optimalt som muligt. Dette vil hjælpe med at definere funktionaliteter for det endelige produkt, og derved gøre det lettere for udvikleren at fokusere på det væsentlige.

Forventningerne defineres som vigtige eller nyttige. De vigtige forventninger er dem, produktet skal bestå, og de nyttige er dem, der vil blive revideret, hvis tiden ikke er en faktor.

2.1 VIGTIGE FORVENTNINGER

- En database med vigtige informationer skal udarbejdes
- Simpel brugerflade
- Kan udføre driftlog og alarm-styring
- Recept håndtering
- Intuitiv kalibrering af vision og robotter

2.2 NYTTIGE FORVENTNINGER

- Udarbejdelse af nye recepter
- Styring af vision system

Et database-system er en god ide til dette projekt, da alarm og drift-log kan gemmes i denne. Derudover kan databasen udarbejdes således, at den kan tilgås af både brugerflade og PLC. På denne måde er det ikke nødvendigt, at al information sendes fra brugerflade til PLC.

Da databasen kan tilgås af både robot-systemet og brugerfladen, kan data-udveksling mellem PLC og brugerflade simplificeres. Dette vil blive udnyttet, ved at kommandoer kan sendes mellem enhederne, for at fortælle hvilken information, der skal hentes fra databasen.

At brugerfladen er simpel og intuitiv at benytte, er vigtigt da robotsystemet skal kunne styres af alle i gartneriet. Af denne grund skal det også være muligt

at ændre recepter, for at få robot-systemet til at agere efter ønske.

Opsamling af drift-data giver stor værdi for et automations-produkt. Ved hjælp af driftlog, kan gartneri-ejere få bedre kendskab til gartneriets drift. Et eksempel kan være, hvis der skiftes pærer, kan man se, hvor stor effekt dette har på væksten.

Det kan forekomme, at enten kamera eller robotværktøjer skal ændres. Hvis dette sker, skal det være muligt for en operatør, at kalibrere systemet. Denne kalibrering bør være nem at udføre, således at den kan udføres af alle.

Både styring af vision systemet og oprettelse af nye recepter direkte fra brugerfladen vil det nemmere at styre robot-systemet. Disse funktionaliteter er dog ikke nødvendige for systemets virkemåde. Derfor vil disse blot udarbejdes, hvis tiden tillader det.

3

KONCEPT

Formålet med dette kapitel er, at give et overblik over både koncept af brugerfladen, men også konceptet for, hvorledes hele robot-systemet skal virke, og kommunikation i robot-systemet. Robot-systemet beskrives, for at give en bedre forståelse for overvejelser om virkemåden af brugerfladen.

3.1 ROBOT-SYSTEM

Ifølge Peder Zoega Beiskær, projektleder af TPR projektet hos Egatec, er det vigtigt at opfylde følgende krav, for at lave en automatisk tomat plukker[6]:

- Plukke-robotten skal være en mobil enhed, der kan detektere en tomatplante og dens position korrekt.
- Robotten skal kunne bestemme modenheten af tomaterne. Den nederste hængende tomat i klasen, vil altid være den sidste tomat til at blive moden. Udover dette, vil den nederste klasse altid være den første til at blive moden. Dette er fundamental viden indenfor tomat produktion i gartnerier.
- Hvis den nederste tomat ikke er moden, bør den mobile enhed fortsætte til den næste plante.
- Hvis den nederste tomat er moden, bør robot-enheden kunne holde planten i ro, klippe tomat-klasen og pakke tomaterne, uden at beskadige tomaterne.

For at opfylde ovenstående punkter, anvender TPR projektet sig af vision-system til detektion af planter, stilke og tomater. Artikulerede robotarme bruges til skånsom bearbejdelse af planter. En vogn bruges til at bevæge robot-systemet mellem planter, og en gribearm benyttes til at trække planter ind til enheden for nemmere bearbejdelse af planten.

3.1.1 *Vogn*

Vognen, der benyttes til projektet, er en standard-vogn fra Bogaerts Greenhouse Logistics[7].



Fig. 2: Bogaerts GL Qii-Lift H-350 pipe rail trolley

Vognen (Se figur 2) er designet til kørsel på varmerør mellem tomatrækker i gartnerier. På vognen findes en sakselift, der bruges til at løfte vognen op til optimal klippe højde.

Udover at den benyttes til at flytte robot-enheden til modne tomatplanter, anvendes vognens batteripakke, til at forsyne resten af systemet. For at forsyne systemer, der ikke benytter 24V, er en inverter eftermonteret, for at konvertere 24V til hhv. 230V eller 12V.

3.1.2 *Lineær gribearm*

På vognen findes en lineær gribearm, designet af Bogaerts GL. Gribearmen trækker tomatplanter med modne tomater ind til robot-systemet.

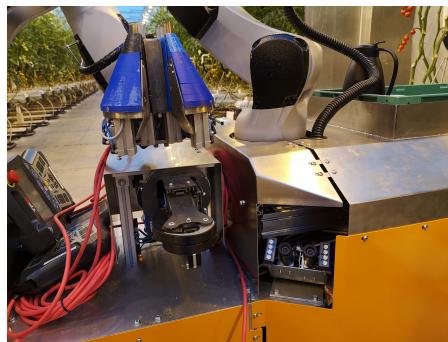


Fig. 3: Gribearm designet af Bogaerts GL

Gribarmen, som ses i figur 3, er lavet således, at en DC motor styrer bevægelse af gribarmen. En griber findes på et skinne system. Ved brug af

kædetræk kan griberen bevæge sig frem og tilbage i en lineær retning. Styring af griberen sker gennem en pneumatisk cylinder. Selve griberen har et spænd på 250mm, hvilket gør at selvom gribarmen blot kan bevæge sig lineært, sagtens kan grib om enhver plante.

3.1.3 Artikulerede robotarme

De to artikulerede robotarme er af typen Comau Racer 5[9], der har 6 led. Robotarmene har en maksimal løftekapacitet på 5kg, hvilket er tilstrækkeligt til at arbejde med mindre emner som tomater. Grundet robotarmenes størrelse og deres 6 led skabes et arbejdsmønster, der er tilstrækkeligt for TPR projektet. Hvis robotarmene var mindre, ville det ikke være muligt, at nå alle planter. Da roboterne har 6 led, er der risiko for, at roboterne ender i en konfiguration, hvor der opstår gimbal lock singularitet¹. For at mindste risikoen for gimbal lock, bør roboterne have 7 led (et redundant led), for at have flere muligheder for at komme til ønskede positioner.

3.1.4 Vision system

For at vide hvor tomatplanterne og tomater befinner sig, benyttes et vision-system, der udvikles af DTI. Vision systemet består af fire 2D kameraer.



Fig. 4: Robot Pick værktøj med kamerasæt

Ved at benytte to kameraer, der befinder sig ved siden af hinanden (som set i figur 4), kan man finde dybden i billeder. Kameraerne defineres som et slave kamera og et master kamera. Kameraerne agerer som to kamerasæt, som et 3D kamera. Disse to kamerasæt er placeret hhv. stationært på vognen og på Robot Picks værktøj. For at finde dybden i billeder, skal der først udføres en stereo-kalibrering for at finde slavekameraets position ift. masterkameraet. Stereo-kalibreringen sker ved at tage et angivet antal billeder af et veldefineret

¹ Gimbal lock er tabet af en frihedsgrad i et tredimensionelt arbejdsrum, der opstår, når to led drives i en parallel konfiguration.

mønster (Et kalibrerings-mønster kan se ud som i figur 5) fra forskellige vinkler og afstande.

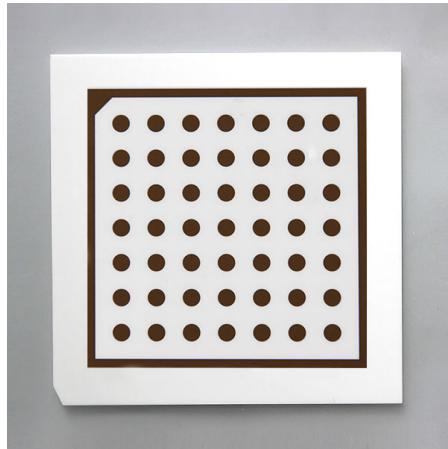


Fig. 5: Eksempel på kalibrerings plade

Når stereo-kalibrering er udført, laves en stereo-korrelation (stereo-matching) og derved fås et dybdebillede ud fra de to 2D billeder. Alle 3D punkter er i forhold til masterkameraet.

Hvis et punkt prikkes ud på et billede fra 3D kameratasættet, kan man finde et punkt defineret ift. masterkameraet.

Da der ønskes punkter defineret ift. roboternes arbejdsområde, er en hand-eye kalibrering nødvendigt. Ved en hand-eye kalibrering tages der igen billeder af et kalibreringsmønster. Kalibreringen af de to kamerasæt varierer, da det ene kamerasæt er stationært og det andet mobilt. Ved kalibrering af det mobile kamerasæt bør kalibreringsmønsteret være stationært, og kamerasættet skal flyttes. For hvert billede skal kamerasættets position og orientering i roboternes arbejdsområde gemmes. Ved kalibrering af det stationære kamerasæt skal kalibreringsmønsteret flyttes, og dets position og orientering ift. roboternes work-space gemmes.

Til vision-systemet er to deep-learning classifiers anvendt til detektion af planter, tomater og deres modenhed, stamme, stilke og samling mellem stamme og stilke.

Den ene classifier bruges ved billeder taget med det stationære kamera. Classifieren skal prikke et punkt ud nederst på stammen, et punkt på den nederste tomat, og dertil et punkt i samme højde på stammen. Inden punkterne prikkes ud, findes modenheten af den nederste tomat. Det antages, at dette gøres ved at kigge efter den røde farve i HSL² farve-formattet for at se, hvor mættet den røde farve er.

Den anden classifier bruges ved billeder taget med det mobile kamerasæt. Dette kamerasæt tager tre slags billeder. Disse tre slags billeder beskrives nedenfor.

² HSL = Hue Saturation Lightness

3.1.4.1 Billede inden Robot Cut griber om stammen

For at robot cut kan gribe om stammen, tages et billede af stammen i samme højde som Robot Cut befinner sig i. På disse billeder prikkes et punkt ud nederst på stammen, og et punkt i samme højde, som toppen af Robot Cuts værktøj. Ved hjælp af disse punkter, findes retning af tomatstammen. Retningen konverteres fra X-Y-Z rotationer til Z-Y-Z orientering, for at roboterne kan fortolke retningen.

3.1.4.2 Billede af stamme undervejs op mod stil

Robot Cut bevæger sig op mod stilken nedefra, og skal derfor prikke punkter undervejs op ad stammen. Kravene for punkterne, der prikkes ud her er, at de skal befinde sig på stammen over robot cut.

3.1.4.3 Billede af stil

Classifieren kan se når Robot Cut befinner sig ved stilken. I dette tilfælde prikkes et punkt ud ved samlingen af stamme og stil, samt et punkt på tomat stilken.

Ved brug af disse punkter, kan tomatstilkens retning findes og sendes til robot-systemet.

Ved hjælp af de to classifiers findes positioner til robot-systemet automatisk.

3.1.5 Bearbejdelse af tomater

Bearbejdelse af tomatplanter er delt op i forskellige sekvenser. Disse sekvenser er defineret som **Retrieve stem**, **Grab stem**, **Move on stem**, **Cut tomatoes** og **Tomato packing**.

Denne segmentering af sekvenser er udarbejdet for, at definere mulige komplikationer, der blot vil forekomme i en given sekvens. Ydermere, er disse sekvenser udarbejdet således, at planterne kan behandles så skånsomt som muligt.

Nedenfor er sekvenserne beskrevet.

3.1.5.1 Retrieve stem

Den første sekvens benyttes til, at trække tomatplanter ind til robot-systemet, hvis den nederste tomat er moden.

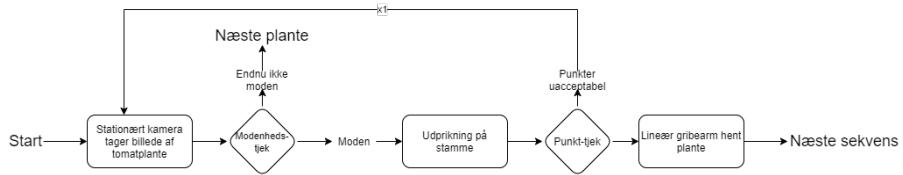


Fig. 6: "Retrieve stem" sekvens

I figur 6 ses første sekvens.

I starten af sekvensen tages et billede af tomatplanten. Der undersøges om den nederste tomat i den nederste klasse er moden. Hvis denne tomat ikke er moden, betyder det, at klasen ikke er klar til at blive plukket. Hvis tomaten er moden, prikkes punkter ud på en stamme og en tomat, for at finde positionen af planten og tomaten. Positionerne sendes til PLC, hvor afstanden til tomatplanten tilses. Hvis afstanden til planten er for stor, må den givne plante befinde sig i en anden række. Hvis dette er tilfældet tages et nyt billede.

Hvis afstanden er acceptabel, bevæges Joris griberen³ ud, griber om planten, og trækker planten ind til robot-systemet.

3.1.5.2 Grab stem

Efter at planten er trukket ind, skal Robot Cut gribe om stammen. Dette sker som vist i figur 7.

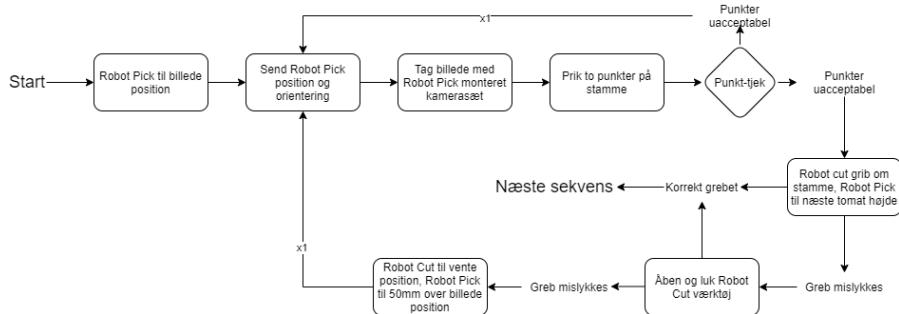


Fig. 7: "Grab stem" sekvens

For at Robot Cut kan gribe om stammen af en tomatplante, skal plantens position og retning findes. Dette findes ved at prikke to punkter ud på stammen. Robot Pick flyttes til en position, hvor både stamme og Robot Cut skal ses. For at stereo-korrelationen kan give punkter defineret i robot-workspace, sendes Robot Picks position og orientering til vision systemet.

Et billede tages med Robot Pick monteret kameraset, hvorpå to punkter på stammen prikkes ud. Vha. de to punkter, kan en retning af planten beregnes og konverteres fra X-Y-Z til Z-Y-Z orientering. Orienteringen samt det øverste punkt sendes til PLC, der undersøger om punktet er acceptabelt.

³ Lineær gribeam designet af Bogaerts GL

Hvis punktet er uacceptabelt, tages et nyt billede. Hvis punktet er acceptabelt, bevæges Robot Cut til punktet med den givet orientering og lukker sit værktøj. Mens dette sker, bevæger Robot Pick til et punkt i samme højde, som den nederste røde tomat. Dette punkt er givet fra det stationære kameraet i tidligere sekvens.

På Robot Cut værktøj er der monteret en induktiv føler, der kan se, om værktøjet er lukket korrekt. Hvis værktøjet ikke er lukket korrekt, er stammen klemt i værktøjet. Værktøjet åbnes og lukkes for at få stammen til at falde ind i værktøjet. Hvis dette ikke lykkes, forsøges sekvensen igen 50mm højere.

3.1.5.3 Move on stem

Inden "Move on stem" sekvensen startes, undersøges Robot Cuts position ift. højden på den nederste tomat. Så snart Robot Cuts position er over den nederste tomat startes **Cut tomaten**. Hvis positionen er under nederste tomat, køres sekvensen i figur 8.

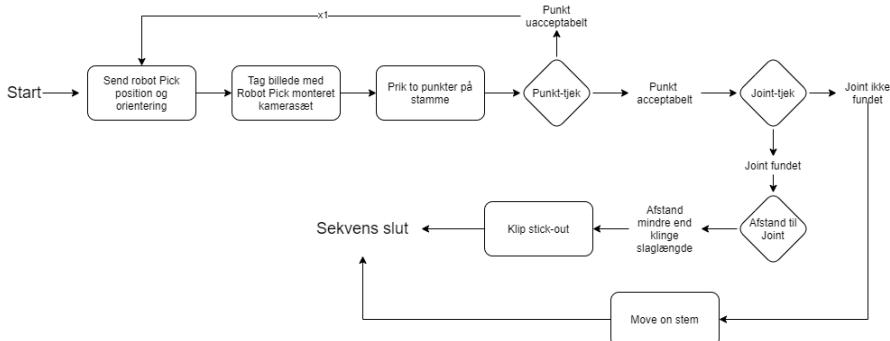


Fig. 8: "Move on stem" sekvens

For at Robot Cut kan bevæge sig op ad stammen, skal endnu et billede tages af stammen. Robot Picks position sendes til vision systemet, og et billede tages. På dette billede prikkes to punkter ud på stammen. Hvis det øverste punkt ses som et joint, må det være et stick-out. Et stick-out kan være for stor, til at Robot Cut kan komme forbi det. Hvis afstanden til stick-out er mindre end 15mm (Klingernes slaglængde), udløses klingerne. Hvis afstanden er større en 15mm, bevæger robot cut sig til stick-out.

Mens Robot Cut enten klipper eller bevæger sig, bevæger Robot Pick til 50mm over punktet Robot Cut modtager.

Så længe Robot Cuts position er lavere end den nederste tomat, køres denne sekvens.

3.1.5.4 Cut tomatoes

Så snart Robot Cut befinner sig højere end den laveste tomat, startes sekvensen i figur 9.

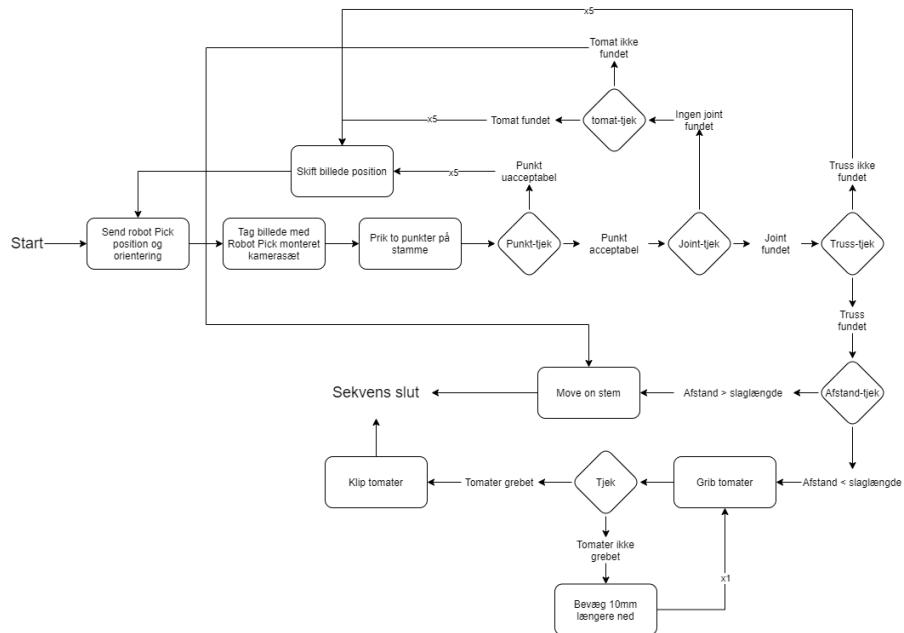


Fig. 9: "Cut tomatoes" sekvens

Efter at et billede er taget, laves en række tjek, inden roboterne bevæges. To punkter prikkes ud på stammen. Hvis joint og stilk findes på billedet, prikkes disse også ud. For at der skal udføres et klip, skal der findes både et joint og en stilk i billedet. Udenfor dette skal Robot Cut minimum have en afstand til stilken på 15mm, før klip udføres.

Tjek udføres ved, at der først undersøges om et joint findes på billedet. Hvis et joint ikke findes, undersøges det, om tomater kan ses på billedet. Hvis tomater kan ses, antages det, at et joint kan findes ved at ændre konfiguration, og derefter tages et nyt billede. Hvis tomater ikke kan ses, kaldes robot sekvensen "move-on-stem", hvor Robot Cut bevæger sig til øverste punkt prikket ud, og Robot Pick bevæger sig til 50mm over punktet.

Hvis et joint findes, laves et "truss-tjek". Her undersøges det, om en stilk kan findes på billedet. Hvis en stilk findes, og afstanden til joint er lavere end klinge slaglængden, bevæger Robot Pick over og gribber fat om stilken. Dette er muligt, da en vinkel mellem joint og punktet på stilken bliver sendt fra vision systemet.

Også på Robot Pick findes en induktiv føler (Se figur 3), og med denne vides det, om Robot Pick har fat i stilken. Hvis den ikke har fat om stilken, forsøges det at gribbe om stilken 10mm længere nede.

Når Robot Pick har fat om stilken, trækkes den 10mm væk, og Robot Cut udløser sine klinger, og klipper stilken over.

3.1.5.5 Tomato packing

Når stilken er klippet, skal tomaterne pakkes. Derudover skal det være muligt at afblade planten, så den er klar til næste gang tomater skal klippes. Dette foregår som i figur 10.

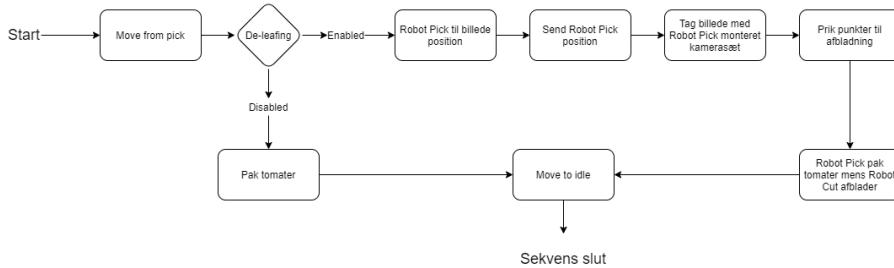


Fig. 10: Tomato packing sekvens

For at pakke tomaterne, skal Robot Pick bevæge sig væk fra planten uden at ramme, hverken plante eller Robot Cut. Derfor bevæges Robot Pick væk, ved hjælp af en rute defineret ift. Robot Picks position fra Robot Cut.

Det er muligt at vælge, at robot-systemet skal afblade planten, så det er klar til næste klip. For at afblade, skal Robot Pick ændre konfiguration, således at de næste to-tre forgreninger kan ses. For enhver forgrenning skal et billede tages, for at finde deres stick-out. Mens Robot Cut afblader planten, pakkes tomaterne i kassen bagerst på vognen (Se figur 1). Pakning sker ift. et pakkemønster defineret ud fra kassens størrelse.

Hvis afbladning ikke skal udføres, pakkes tomaterne mens Robot Cut slipper tomat planten.

Slutningsvis bevæges begge robotter til venteposition, og Joris griberen placerer planten tilbage i rækken.

3.2 BRUGERFLADE

Brugerfladen til robot-systemet skal være webbaseret, og skal derfor laves som en hjemmeside. En hjemmeside kræver intet specifikt operativ-system, og alle enheder vil kunne tilgå siden. Siden vil ikke kunne findes på internettet, derfor skal man have adgang til samme netværk som brugerfladen.

Brugerfladen skal opbygges efter en MVC⁴ arkitektur, som set i figur 11.

⁴ MVC = Model-View-Controller

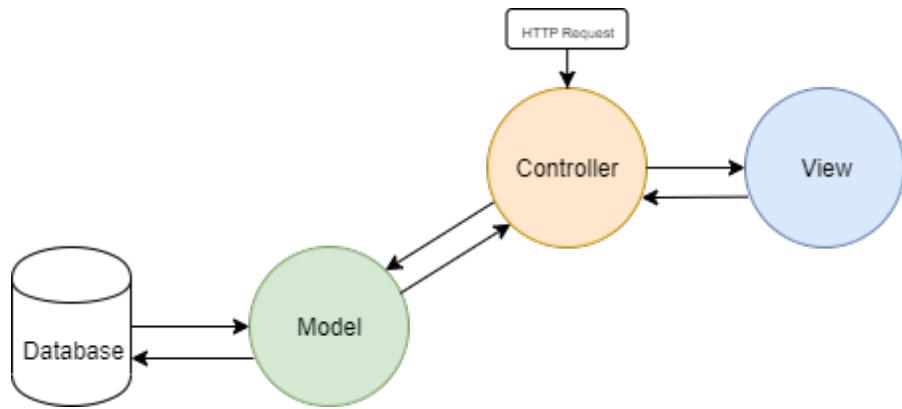


Fig. 11: MVC struktur

Arkitekturen opdeles i tre dele: model, view og controller. Komponenterne er ansvarlige for at håndtere forskellige aspekter af applikationsudvikling af både web / software. Selve karakteren af MVC arkitekturen vil være således, at der er lav kobling⁵ mellem de forskellige komponenter.

Som man kan se i figur 11, modtager controller-komponenten HTTP Requests. Kontrolleren benyttes til, at beordre model-komponenten til, at forberede de oplysninger, der kræves af view-komponenten. View-komponenten bruger data, som kontrolleren modtog af model, til at vise det endelige output[4].

3.2.1 *Model*

Model et det komponent, der har adgang til databasen. Dette niveau definerer datastruktur for applikationens dataobjekter. Modelkomponenten er koblet med kontrolleren.

3.2.2 *View*

Viewkomponenten viser modedata for brugeren. Det hjælper med at oprette en grænseflade, der viser det endelige output til brugeren. Komponenten viser output til brugeren efter modtagelse af instruktion / information fra kontrolleren. Derudover indsamler den anmodninger fra brugeren og sender anmodningerne videre til kontrolleren.

Modelkomponenten og viewkomponenterne forbides ikke til hinanden. Viewkomponenten får de nødvendige data til præsentationen fra modellen gennem kontrolleren.

⁵ Lav kobling = Et forhold, hvor et modul interagerer med et andet modul gennem en enkel og stabil grænseflade, uden bekymring over det andet moduls interne implementering

3.2.3 *Controller*

Kontroller-komponenten fungerer som den centrale enhed i MVC-arkitekturen. Komponenten fungerer, som en bro mellem brugeren og systemet. Output fortolkes, behandles og konverteres til passende meddelelser og sender det til viewkomponenten.

Fordelen ved en MVC arkitektur er, at man kan lave flere view til samme model. Dette gør det nemmere at genbruge kode. Udover dette er det nemt at lave ændringer i hjemmesiden, som layout og farver, da ændringer kun implementeres et enkelt sted.

Ved at bruge en MVC-arkitektur, er det muligt, at udvide en web-applikation, da vise funktionaliteter blot defineres en gang. Forbindelse til en database behøves blot at oprettes en gang, før den kan benyttes over hele applikationen.

Til brugerfladen skal der være to forskellige bruger-niveauer. Disse bruger-niveauer er til operatører og master brugere. Grunden til disse niveauer er, at operatører ikke skal have mulighed for at ændre på virkemåden af robotsystemet. En operatør skal have mulighed for at kontrollere basale operationer som start/stop, valg af recepter og hastighedskontrol.

For at oprette restriktioner til funktionaliteter, vil der defineres hvilke kontrollere, der må skabes adgang til, ved specifikke brugere. Ved at give alle brugere et adgangsniveau, vil det være muligt, at specificere, hvem der har restriktioner.

Ved at specificere restriktioner til kontrollere, vil det ikke være muligt, for en bruger med lav adgangsniveau, at benytte hverken kontrollerens views eller metoder.

Da forskellige brugere skal have adgang til applikationen, skal et login-system udvikles. Bruger-information gemmes i en database, hvor deres adgangsniveauer også kan findes. Da et bestemt adgangsniveau skal benyttes, for at få adgang til en kontroller, vil det ikke være muligt, at benytte brugerfladen, uden at være logget ind.

3.3 KOMMUNIKATION

Til systemet skal en server udarbejdes. Denne server skal indeholde en brugerflade, der kan kommunikere med PLC for at udveksle data. Udover dette kommunikerer PLC med både robotter og vision systemet.

På serveren kommer et database-system, webinterface og kommunikation til PLC.

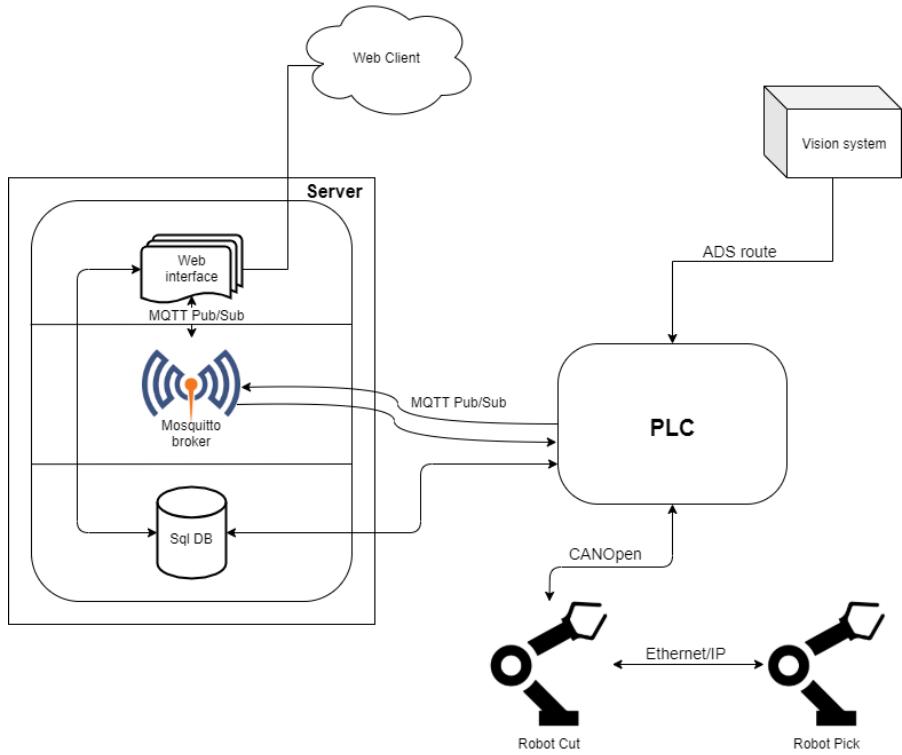


Fig. 12: Kommunikationskoncept

Som man kan se i figur 12 opsættes kommunikation i et ”star network”, hvor PLC er center enhed, der kommunikerer med alle enheder i systemet. Kommunikationen med vision-systemet sker gennem en ADS route⁶. Forbindelsen fungerer ved, at vision-systemet læser og skriver i variable på PLCen. Dette betyder, at vision-systemet kommer til at have forbindelse til PLCen, men ikke den anden vej.

Kommunikationen til Robot Cut skal ske gennem en CANopen forbindelse. De to robotarme (Robot Cut og Robot Pick) vil have en Master/Slave rollefordeling, hvor Robot Cut er Master. Dette vil derfor sige, at al kommunikation til Robot Pick vil ske igennem Robot Cut.

For at kommunikere med PLC fra den webbaserede brugerflade, skal det ske gennem en MQTT forbindelse⁷.

Både web-interface og PLC vil have forbindelse til en MQTT broker, hvor relevante topics bliver udgivet og abonneret på. Derfor kommer PLC og web-interface ikke til at have forbindelse til hinanden, men til samme mægler, hvor der abonneres på relevante emner.

Brugerfladen og PLC bliver forbundet til den samme SQL database. Som beskrevet i kapitel 2, betyder dette, at de samme data vil kunne hentes af PLC

⁶ ADS (Automation Device Specification) er et enheds-uafhængig og fieldbus-uafhængig kommunikations-interface

⁷ MQTT = Message Queuing Telemetry Transport

og brugerflade. Kommandoer der sendes med MQTT forbindelsen fortæller, at databasen er opdateret og den anden enhed skal hente data fra databasen. Opstillingen er smart, da kommunikation mellem brugerflade og robot-system kommer til, at ske hurtigt. Da kommandoer, sendt mellem brugerflade og robot-system, er små, vil responstiden altid forblive hurtig. Ydermere vil dette sikre mod MITM angreb⁸. Da følsom data ikke sendes til den offentlige MQTT mægler, vil der ikke være mulighed for, at hente eller ændre data.

Kommunikationskonceptet vil benyttes for at adskille robot-styring og brugerflade, samt for at mindske kommunikation mellem de forskellige enheder. Konceptet er en letvægt, sikker og effektiv styring til robot-systemet. Ved kommunikation mellem brugerflade og PLC kan alle enheder i systemet kontrolleres.

Som beskrevet i afgrænsingen, vil der ikke fokuseres på, at ændre i virkemåden af robot-systemet. Derfor fokuseres der kun på, at udarbejde server med database, MQTT forbindelse, brugerflade og kommunikation mellem robot-system og brugerflade.

3.3.1 CANopen

CANopen er en kommunikationsprotokol for embeddede systemer, der er meget brugt indenfor automation. CANopen implementerer lag over, og inkluderer, netværkslaget i OSI modellen. Kommunikationsprotokollen bliver brugt meget i motor kontrol, men bliver også brugt i stort omfang indenfor styring af robot-systemer og landbrug.

CANopen er en "higher layer" protokol baseret på CANbus. Dette betyder, at CANbus står for transport, mens CANopen dataformattet af data, der sendes[2].

CANopen-standarden består af et adresserings skema, flere små kommunikationsprotokoller og et applikationslag defineret af en enhedsprofil. Kommunikationsprotokollerne har understøttelse af netværksadministration, enhedsovervågning og kommunikation mellem noder. Protokollen på lower level er normalt "Controller Area Network" (CAN), dog kan enheder, der benytter et andet kommunikationsmiddel (Såsom Ethernet Powerlink eller EtherCAT), også implementere CANopen.

CANopen forbindelsen i dette projekt virker, som en Master/Slave forbindelse, hvor PLC er Master og Robot Cut er slaven.

⁸ man-in-the-middle

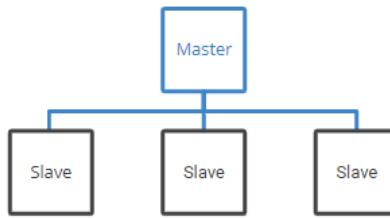


Fig. 13: Master/Slave terminologi

I en master/slave forbindelsen sker al kommunikation gennem master, som ses i figur 13. I dette projekt, hvor PLC, Robot Cut og Robot Pick er tilsluttet i en Daisy chain, er denne master/slave rollefordeling underordnet, da Robot Pick ikke er slave for PLC. For at kommunikere med Robot Pick skal det ske gennem Robot Cut.

En rollefordeling, hvor både Robot Cut og Robot Pick er slave til PLC, vil være optimalt, da kommunikation til begge robotter kan ske optimalt.

Denne måde at kommunikere på i projektet sker grundet begrænsninger i robotterne.

3.3.2 *ADS-route*

ADS kommunikationsprotokollen er udviklet til TwinCAT-systemarkitekturen. ADS gør det muligt at behandle enkelte moduler i software som uafhængige enheder. Kommunikation mellem disse enheder udveksles gennem en ADS-grænseflade ved hjælp af en ”message router”. Routeren styrer og distribuerer alle meddeleser over TCP/IP-forbindelser[3].

3.3.3 *MQTT*

MQTT er en åben OASIS og ISO-standard (ISO/ IEC 20922) letvægts public / subscribe netværksprotokol, der transporterer meddelelser mellem enheder ved hjælp af "Topics"[1].



Fig. 14: Publish/Subscribe

MQTT er en simpel "messaging protocol", der er designet til enheder med lav båndbredde. MQTT gør det muligt at sende kommandoer til kontroloutputs, læse og publish data.

3.3.3.1 Topics

Topics er måden, hvorpå man viser interesse på indgående beskeder, og hvordan man ønsker at sende beskeder.

Topics er repræsenteret som en streng separeret af '/'. Enhver skråstreg indikerer et topic level.

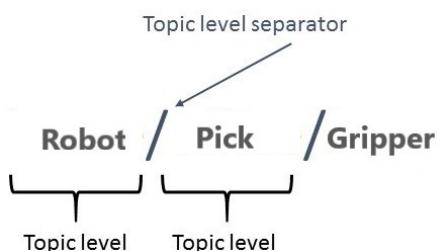


Fig. 15: Topic struktur

Klienter kan vælge at abonnere på topics (Et eksempel på et topic kan ses i figur 15). Når en besked ændres, vil klienter der subscriber topic, modtage den nye besked.

3.3.3.2 Broker

En MQTT broker er en mægler, der modtager alle beskeder. Udover at modtage beskederne, filterer mægleren derudover beskederne og sender dem videre til de klienter, der abonnerer til de bestemte topics.

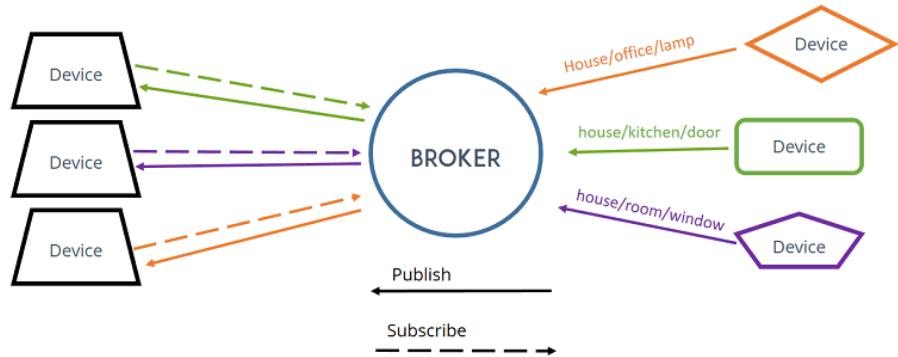


Fig. 16: MQTT broker

Som man kan se i figur 16, er en broker center af MQTT protokollen. Klienter forbinder til en broker, abonnerer til topics og sender beskeder til bestemte topics. Dertil sørger en broker for, at beskederne kommer til de klienter beskeden vedkommer.

4

IMPLEMENTERING

I dette kapitel findes information om implementering af, og tanker bag, konstruktion af en brugerflade til TPR robot-systemet.

4.1 HARDWARE

For, at brugerfladen altid har adgang til robot-systemet, er det vigtigt, at serveren med brugerfladen altid befinner sig på samme netværk som robot-systemet. Måden dette gøre på, er ved at placere serveren i nærheden af robot-systemet.

En industriel Raspberry Pi kaldet RevPi Connect[10](Se figur 17), benyttes til lagring af server. Denne Raspberry Pi findes i robot-systemet, og vil derfor altid være i nærheden.

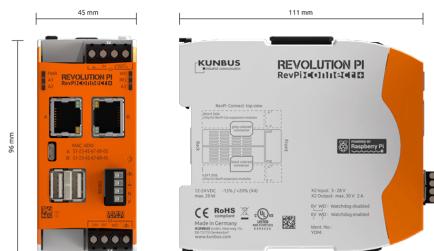


Fig. 17: RevPi Connect

RevPi Connect har 1GB RAM og 4GB lager. Dette er tilstrækkeligt, da letvægts kommunikationsformen MQTT benyttes. Ifølge www.mqtt.org er MQTT protokollen en letvægts kommunikationsform, der kan benyttes på små microcontrollers.

Da RevPi Connect er lavet til industriel brug, kan den monteres på en DIN skinne i robot-systemet.

RevPi Connect er udstyret med to RJ45 ethernet porte. Ved hjælp af VPN på RevPi og internet forbindelse på den ene port, kan RevPi Connect bruges, som access point til fjernstyring af robot-systemet.

Dette virker ved, at port A er sat til DHCP forbindelse med adgang til internettet. Port B har en fast IP-adresse i samme netværk, som resten af robot-systemet, hvilket gør det muligt for enhederne at kommunikere. Ved hjælp af

OpenVPN på RevPi kan en enhed på et andet netværk få adgang til enheden. Da RevPi også har en adgang til netværket til robot-systemet, kan andre enheder i robot-systemet styres.

Grunden til at en RPi er benyttet til projektet er, at et linux baseret operativ system ønskes.

Da Linux er open source, er det gratis at benytte, og derfor er det muligt at holde omkostninger til robot-systemet nede. Linux er kompatibel med mange andre operativ systemer. Derfor er det intet problem, at udvikle en web-applikation i Windows og gemme den på en Linux server.

En af de største fordele ved at benytte en linux baseret server er dens stabilitet. Ved brug af biblioteker på en linux baseret server, vil serveren operere bedre end på en Windows baseret server. Dette sker, da en linux baseret server ikke vil blive langsommere eller fryse.

4.2 SOFTWARE

Da operativ-systemet på RevPi Connect er ”Raspbian stretch”, som er Linux baseret, opsættes en LAMP stack¹.

I serveren findes en MQTT mosquitto broker, som er mægler mellem MQTT klierterne. Når en kommando sendes fra brugerfladen, sendes den til mæglen. Herefter sender mæglen kommandoen til alle klienter, der abonnerer på det bestemte topic.

En MariaDB database befinner sig på serveren. Denne database indeholder tabeller med login-, recept-, frame-, log-, alarm-, alarmlog- og aktive data information. Databasen er til rådighed fra både PLC og brugerflade.

En mariaDB database benyttes fremfor en MySQL database, da den viser forbedret hastighed ift. MySQL databasen[16]. Til udvikling af brugerfladen, benyttes jQuery 3.5.1 til AJAX kald og Bootstrap 4.5.3 til design og styring af modals.

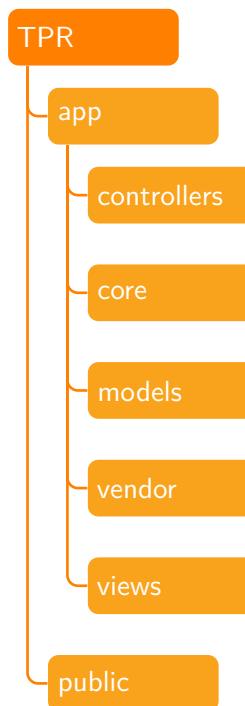
Da MQTT protokollen ikke er standard i PHP, benyttes et MQTT bibliotek fra GitHub, udarbejdet af ”Namoshek”[11]. Biblioteket giver mulighed for at oprette forbindelse til en MQTT-broker, hvor man både kan sende beskeder og abonnere på topics.

4.3 BRUGERFLADE

Som forklaret i kapitel 3.2, er brugerfladen struktureret efter en MVC arkitektur.

Den anvendte arkitektur er som følger:

¹ LAMP = Linux Apache MySql PHP



Arkitekturen er delt op i ”applikation” og ”offentlige” filer. De offentlige filer er dem, der kan tilgås direkte fra browseren. Disse JavaScript og CSS filer.

4.3.1 *Applikation*

Applikationssmappen indeholder selve applikationen. Med dette menes, at public mappen indeholder offentligt tilgængelige filer som JavaScript og CSS filer. Derudover indeholder public mappen en index.php, der er indgangen til applikationen.

Applikation indeholder **controllers**, **core**, **models**, **vendor**, **views** og en *init.php* fil.

Vendor indeholder biblioteker inkluderet til projektet og init.php benyttes til at inkludere vigtige core programmer, så de blot inkluderes et enkelt sted.

4.3.1.1 *Controllers*

Kontrollerne i MVC-arkitektur håndterer enhver indgående URL-anmodning. Controller-klassen indeholder offentlige metoder. Klassen og dens handlings-metoder håndterer indgående browseranmodninger, henter nødvendige model-data og returnerer passende svar til view.

Enhver kontroller klasse-navn, skal ende med ”Controller”. Det er vigtigt, at navnet på kontrollerne har samme struktur således, at de kan kaldes gennem

Controller.php klassen. Et eksempel på dette er, at kontrolleren til setup hedder ”setupController”. Alle kontrollere skal findes i mappen kaldet controllers, for at **Controller.php** kan referere til dem.

Brugerfladen deles op i 12 sider, derfor laves 12 kontrollere. Disse er; *Adjust, Alarm, Info, Log, Main, Manual, Recipe, Setup, Signup, Tool, User* og *Vision*. Alle kontrollere indeholder, som minimum, en metode kaldet index. metoden er den der kaldes, hvis en metode ikke defineres, når kontrolleren kaldes.

Som nævnt tidligere, benyttes kontroller-metoder til at hente nødvendige modeldata. Dette gøres vha. core programmet ”Controller.php”.

```

1  <?php
2  class Controller {
3      public function model($model) {
4          require_once './app/models/' . $model . '.php';
5          return new $model();
6      }
7      public function view($view, $viewbag = []) {
8          require_once './app/views/' . $view . '.php';
9      }
10     public function post () {
11         return $_SERVER['REQUEST_METHOD'] === 'POST';
12     }
13     public function get () {
14         return $_SERVER['REQUEST_METHOD'] === 'GET';
15     }
16 }
```

Listing 1: Controller.php

Listing 1 viser Controller klassen. Der er defineret fire simple metoder, der kan benyttes af enhver kontroller.

Hvis en kontroller skal hente modeldata, kaldes Controller metoden ”model”. Denne metode henter den gældende model i model-mappen.

```

1  <?php
2  class SetupController extends Controller {
3      public function index () {
4          $this->model('setup')->getFrames();
5          $this->model('subscribe')->subscribe1("mqtt/plc2hmi
6 /program");
7          $this->view('setup/setup.view');
8      }
9      public function sendCommand () {
10         $this->model('publish')->sendCommand($_POST);
11         if (isset($_POST['program'])) {
12             $this->model('setup')->updateCurrent($_POST);
13         }
14         public function updateFrame () {
15             $this->model('setup')->updateFrame ($_POST);
16             $this->model('publish')->sendCommand ($_POST);
17         }
}
```

Listing 2: setupController.php

Listing 2 viser ”setupController”. som kan ses i index-metoden, kaldes model metoden fra *Controller.php*. Ved at benytte modellen, skaffes der adgang til modellens metoder. På samme måde, når et view skal kaldes, søger ”controller” metoden i mappen views efter mappen ”setup”. I denne mappe findes det pågældende view ved navn ”setup.view.php”.

Ved index metoden, gemmes model data i `$_SESSION` og vises direkte i `setup.view`. En anden måde at vise modeldata, er ved at gemme modeldata i en ”`$viewbag`”, og kalde view med `viewbag`. Denne måde er at foretrække, da data gemt i en session kan tilgås i alle views.

4.3.1.2 Core

Core programmerne er til for, at MVC strukturen fungerer efter hensigten. Disse programmer er hhv. **Controller**, som er beskrevet ovenfor, **Database**, **db_config**, **mqtt_config**, **Mqttconn**, **Restricted**, **Router**.

Config

Der benyttes to config filer. Disse er konfiguration til database og MQTT kommunikation. I begge config filer findes host, username og password. Host er IP-adresse på RPi (192.168.29.56).

Filerne er defineret for at konfigurationsdata er samlet, og blot defineret et sted.

Database

`Database.php` skaber forbindelse til mariaDB databasen, der befinder sig på RPi.

`Database` er en klasse, hvor forbindelsen skabes i en constructor. På denne måde skabes SQL forbindelsen, når et database-objekt konstrueres. `Database` nedarver fra `db_config`, og har derfor adgang til de konfigurationsvariable, der findes i **db_config** klassen.

I et ”try/catch” statement forsøges det at oprette en PDO forbindelse. PDO forbindelsen anvendes i stedet for MySQLi, da den tillader overførsel af variabler og værdier direkte. Derudover kan PDO auto-detect variabel-type[12].

Mqttconn

`Mqttconn.php` fungerer på samme måde som `Database.php`. Forbindelsen skabes i constructoren af `MQTTConn` klassen. Da MQTT ikke er standard i PHP, skabes MQTT forbindelsen gennem et bibliotek[11]. For at skabe en MQTT forbindelse, benyttes både `\PhpMqtt\Client\MQTTCClient` og `\PhpMqtt\Client\ConnectionSettings`. Først skabes en MQTT klient og et `ConnectionSettings` object. Klienten kan skabe en forbindelse til MQTT broker på RPi vha. `ConnectionSettings`. Standard indstillinger til forbindelse

anvendes. Disse er som set i listing 3.

```

1  public function __construct(
2      int $qualityOfService = 0,
3      bool $retain = false,
4      bool $blockSocket = false,
5      int $socketTimeout = 5,
6      int $keepAlive = 10,
7      int $resendTimeout = 10,
8      string $lastWillTopic = null,
9      string $lastWillMessage = null,
10     bool $useTls = false,
11     bool $tlsVerifyPeer = true,
12     bool $tlsVerifyName = true,
13     string $tlsClientCertificateFile = null,
14     string $tlsClientCertificateKeyFile = null,
15     string $tlsClientCertificatePassphrase = null
16 )

```

Listing 3: Forbindelsesindstiller MQTT

Som man kan se i listing 3, er MQTT beskederne ikke ”retained”, når de sendes fra brugerfladen. Dette er fordi PLCen kun skal modtage kommandoer når robot-systemet er tændt. Hvis en besked er retained, gemmes den, og bliver sendt til enheder, når de tilslutter det pågældende topic. Derfor risikerer man, at robot-systemet modtager en gammel kommando, når systemet startes.

QoS² er sat til 0, da der kun ønskes, at en besked sendes en gang. Hvis QoS er sat til 1, bliver beskeden sendt indtil en bekræftigelse sendes retur. QoS 2 sikrer, at en besked kun modtages en enkelt gang[13]. Da QoS 1 og 2 er langsommere end QoS 0 benyttes QoS 0.

Restricted

I Restricted.php findes en metode med tre arrays, der indeholder begrænsninger til kontrollere og deres metoder. Array \$restricted_urls indeholder begrænsninger for brugere, der ikke er logged-in. De Kontrollere, der er adgang til, uden at være logget ind, er userController og infoController. Kontrolleren ”userController” indeholder en metode til at logge ind på brugerfladen. Kontrolleren ”infoController” indeholder et view, hvor man kan få kontaktinformation til Egamatic.

Som operatør (”access level” 2) har man ikke adgang til signup, adjust, vision, setup og logController. Hvis man er logget ind med ”access level” 1, har man adgang til alle kontrollere.

Hvis den ønskede metode findes i det pågældende array, returneres *false*, ellers returneres *true*.

Router

Router.php anvendes til, at læse URL for at finde den ønskede kontroller. URL hentes og separeres i et array adskilt af ”/”, hvorefter de tre første indeks

² Quality of service

negligeres. De første indeks negligeres, da disse ikke indeholder information om kontroller, ellers deres metoder. Det fjerde indeks indeholder navnet på en kontroller, og femte indeks indeholder en metode. Hvis arrayet indeholder flere indeks, er disse parametre til metoden. Ved brug af Restricted.php undersøges det, om kontroller-metoden er begrænset. Hvis Controller og metode er begrænset, nавigerer Router til mainController. Hvis brugeren ikke er logget ind, nавigeres de til userController.

4.3.1.3 *Models*

Model programmerne sørger for al den datarelaterede logik, som brugeren arbejder med. Dette er de data, der overføres mellem View og Controller. Der er defineret modeller til MQTT forbindelse og modeller til database forbindelse.

MQTT modeller

Disse programmer er hhv. publish.php og subscribe.php.

```

1  public function sendCommand($data) {
2      // Get topic and value for MQTT
3      $topic  = $data['topic'];
4      $value  = $data['value'];
5      if ($value !== NULL) {
6          $this->conn->publish($topic, $value, $this->qos);
7      } else {
8          return false;
9      }
10 }
```

Listing 4: sendCommand metode fra publish modellen

Måden, hvorpå publish virker er ved, at en metode kaldes med topic og en kommando. I listing 4 ses, hvordan en kommando sendes til MQTT mægleren. Da QoS er 0, tjekkes der ikke om, kommandoen er afsendt korrekt.

Når der abonneres på et topic, kaldes subscribe metoden ”subscribe1” eller ”subscribe2”, hvor man hhv. kan abonnere på et eller to topics. For hvert topic, der abonneres på, tages et topic som argument. Når en besked modtages, gemmes den i en `$_SESSION`, og forbindelsen afsluttes.

Database modeller

Der er defineret modeller til database kommunikation ved navn; `user.php`, `signup.php`, `setup.php`, `main.php`, `log.php`, `alarm.php`, `adjust.php`. Af disse modeller, benyttes `user.php` og `signup.php` til brugerhåndtering, `main.php` og `setup.php` til driftsstyring, `log.php` og `alarm.php` til data logning, og `adjust.php` til recepthåndtering.

Recepsthåndtering

I adjust.php findes tre metoder. For at Controller kan kalde et view, der viser recept information, findes en metode kaldet *show(\$id)*. Metoden har til formål, at hente al recept data og returnere den. Recept information er fordelt over flere tabeller, og derfor benyttes et JOIN i ”SQL Query”.

```

1   $part1 = "SELECT *, operation_mode.name as
2     operation_name, packing_pattern.name as pattern_name ";
3   $part2 = "FROM recipe ";
4   $part3 = "LEFT JOIN operation_mode ON recipe.
5     Operation_mode_id = operation_mode.id ";
6   $part4 = "LEFT JOIN packing_pattern ON recipe.
7     packing_pattern_id = packing_pattern.id WHERE recipe_id
8       = :id";
9
10  $sql = $part1 . $part2 . $part3 . $part4;

```

Listing 5: SQL Query til recept håndtering

Som ses i eksemplet i listing 5, laves en forespørgsel på at hente al data fra recept tabellen i databasen. Da navn på driftsmåden og pakkemønster findes i to andre tabeller, laves et LEFT JOIN³ Ved at bruge deres ID som ”foreign key” hentes navnene. Der hentes information for et bestemt receipt_id. Det bestemte id indsættes i SQL forespørgslen i et ”prepared statement”. En variabel bindes til forespørgslen efter den er defineret, for at modvirke SQL-injektions⁴.

Da robot-systemet skal kunne operere uden, at en operatør er logget på brugerfladen, er der lavet en tabel i databasen, hvor vigtige nuværende værdier befinner sig. For hver gang en ny recept anvendes, opdateres recipe_id i tabelen ”currentValues”.

Da det er muligt at lave nye recepter, findes en metode i ”recipe” kontrolleren kaldet; *updateRecipe(\$recipe)*. Metoden tager imod et array med recept information.

Som udgangspunkt benyttes metoden til at opdatere en eksisterende recept.

³ LEFT JOIN returnerer alle forekomster fra den venstre tabel, og de forekomster fra højre tabel, der matcher.

⁴ SQL-injektion er en kodeinjektionsteknik, der kan ødelægge din database.

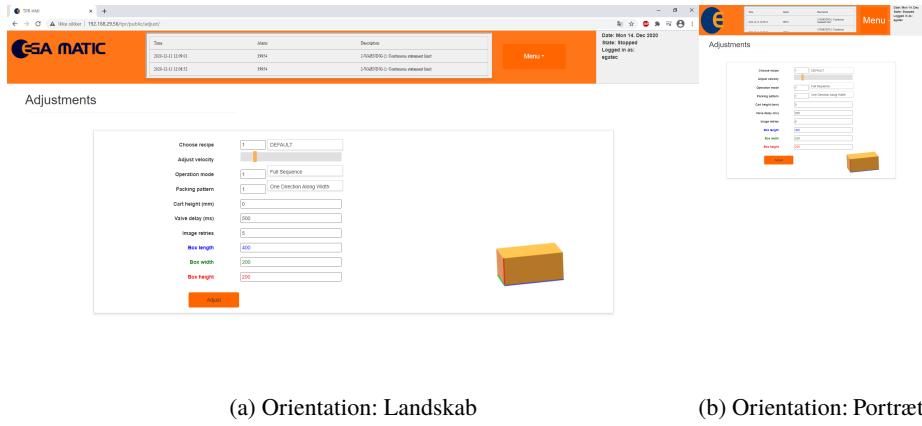


Fig. 18: Adjust recipe view

I ”recipe.view” set i figur 18, ses hvordan recepter defineres. På siden kan man ændre al recept information for en ønsket recept. Inden informationen gemmes, bliver de ”sanitized”, for at fjerne uønskede tegn. Ved at ”sanitize” variable, sikres der for, at en SQL forespørgsel kan udføres.

Inden en recept opdateres eller udarbejdes, kaldes en SQL forespørgsel, der undersøger, hvor mange recepter der findes i tabellen. Hvis antallet af eksisterende recepter er højere end det receipt_id der vælges i view, betyder det, at recepten allerede findes. Hvis receipt_id er højere end antal eksisterende recepter, findes den ikke endnu, og en ny skal oprettes. Hvis en ny recept laves, gemmes den på den næste plads i tabellen.

Driftsstyring

I *main.php* modellen findes to metoder. Disse bruges til at finde hvilket program der anvendes, og hvilken hastighed roboterne opererer med. Da viigtig nuværende driftsdata findes i database tabellen ”currentValues”, anvendes metoderne til, at hente data fra tabellen.

i *setup.php* findes metoder, til at hente og opdatere work-frame og tool-frames.

```

1 public function getFrames() {
2     $sql = "SELECT * FROM frame";
3     $stmt = $this->conn->prepare($sql);
4     $stmt->execute();
5     $result = $stmt->fetchAll();
6     if (isset($result[0]) && sizeof($result) >= 1) {
7         for ($i = 0; $i < sizeof($result); $i++) {
8             $_SESSION[$result[$i]['id']] .= "-x" = $result[$i]['x'];
9             $_SESSION[$result[$i]['id']] .= "-y" = $result[$i]['y'];
10            $_SESSION[$result[$i]['id']] .= "-z" = $result[$i]['z'];
11            $_SESSION[$result[$i]['id']] .= "-a" = $result[$i]['a'];
12            $_SESSION[$result[$i]['id']] .= "-e" = $result[$i]['e'];
13            $_SESSION[$result[$i]['id']] .= "-r" = $result[$i]['r'];
14        }
15    }
16 }
```

Listing 6: Metode getFrames() fra setup.php

Listing 6 viser, hvorledes frames hentes. Først kaldes en SQL-forespørgsel, hvor alle frames hentes. For at gemme frames benyttes en for-løkke.

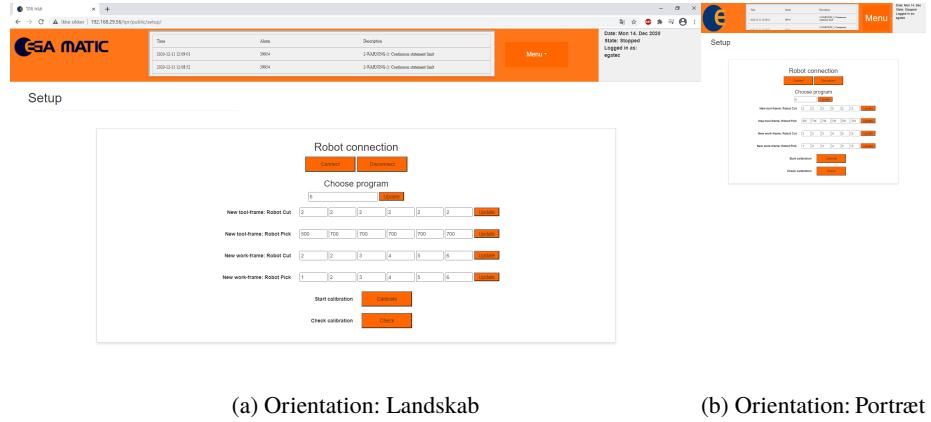


Fig. 19: Setup view

Figur 19 viser, at der er defineret fire frames. Der findes frames til Robot Picks og Robot Cuts ”workframe” og ”toolframe”. De fire frames er i ”setup.view” defineret som hhv. ”ct”, ”cw”, ”pt” og ”pw”. i database tabellen ”frames” er id defineret med samme forkortelse.

Ved brug af for-løkkens i listing 6, gemmes de fire frames, så de kan vises i ”setup.view”.

Ved at ændre roboterne toolframe og workframe, ændres måden hvorpå robotterne bevæges. Derfor bør ændring af disse frames kun kunne ændres af en master bruger. Der er risiko for, at en ændring i frames resulterer i kollision ved resten af robot-systemet.

I *setup.php* findes ligeledes metoder til at hente og opdatere nuværende program, så robot-systemet kan operere uden interface.

Brugerhåndtering

Som beskrevet ovenfor, benyttes *user.php* og *signup.php* til brugerhåndtering. For at logge på brugerfladen, er der defineret en metode i *user.php* modellen kaldet ”login”. Metoden tager imod et brugernavn og en adgangskode.

Først defineres en forespørgsel på al information om brugeren, hvor brugernavn er det argument metoden tager imod. Hvis en bruger findes med dette brugernavn, skal adgangskode undersøges. Da adgangskoden er hashed med MD5⁵ benyttes funktionen ”password_verify” til at dekryptere adgangskoden. Hvis adgangskoden matcher med adgangskoden fra databasen, sættes `$_SESSION['logged_in'] = true;` samt brugernavn og ”access-level” gemmes i SESSIONs. Hvis adgangskode er forkert, eller brugernavn ikke findes, sættes `$_SESSION['logged_in'] = "Incorrect login";`.

5 MD5 er en kryptografisk hashfunktion med en 128-bit hashværdi.

Hvis der ønskes at lave en ny bruger, kaldes signup metoden `register($username, $password, $password2, $access)`. Metoden tager imod to adgangskoder, da den ene er til at verificere, at adgangskoden er korrekt skrevet.

Efter de fire input er ”sanitized”, tjekkes der om adgangskoderne er ens. Hvis ikke, sættes `$_SESSION['message'] = "The two passwords does not match!"`.

Hvis de er ens, laves en SQL forespørgsel, der undersøger om en bruger findes med samme brugernavn.

Hvis brugernavnet endnu ikke findes, hashes adgangskoden, og en bruger ind-sættes i databasen.

Data logging

Disse modeller benyttes ikke til at logge data, men til at vise den data, robot-systemet har logget.

Både i `alarm.php` og `log.php` findes metoderne `get10()` og `get10more()`. Metoderne er ens, dog hentes data fra forskellige tabeller i databasen. For at få de 10 seneste alarmer eller driftslog benyttes nøgleordene ”ORDER BY time DESC” og ”LIMIT 10”. Ved hjælp af ”ORDER BY time DESC”, sorteres data ift. kolonnen ”time” med de højeste værdier først.

Ved `get10more()` foregår dette på samme måde, dog er LIMIT sat til en værdi defineret i et ajax kald.

Modellen `alarm.php` indeholder også en metode kaldet `getLatest()`, der finder den seneste kaldte alarm. Alarmen returneres, for senere at blive brugt til at ”godkende” nuværende alarm.

4.3.1.4 Views

View håndterer appens datapræsentation og brugerinteraktion. Der er defineret et view til alle kontrollere.

Alle views er delt op i `<div>`. HTML er et Markup Language, hvilket betyder at det bruges til at strukturere web-applikationer. Hjemmesiden udarbejdes i samspil med CSS, der benyttes til design af hjemmesiden. Brugen af `<div>` som containere, gør dette nemmere at udarbejde design med CSS.

Som beskrevet i kapitel 4.3.1.1 gemmes model-data i en `$viewbag` eller en `$_SESSION` for at kunne vises i et view, som set i listing 7.

```

1  <div class="block">
2    <label>Choose recipe</label>
3    <input class="number" type="number" pattern="[0-9]*" id="recipeNoInput" name="recipeNo" value=<?php if(isset($_SESSION['recipe'])) { echo $_SESSION['recipe']; } ?>">
4    <span class="desc" id="recipeNameSpan"><?php if (isset($viewbag['adjust'])) { echo ($viewbag['adjust'][, 'recipe_name']); } ?></span>
5  </div>

```

Listing 7: Udklip af `recipe.view.php`

Forskellen mellem at gemme data i \$viewbag og \$_SESSION er, at \$viewbag-data blot findes på den pågældende side, hvor \$_SESSION-data er gemt, indtil den nuværende session lukkes.

Det er vigtigt, at HTML elementer har enten et id eller en klasse, da disse benyttes til Ajax kald.

Styring sker gennem HTML knapper og tekst-inputs. Ved tryk på knapper, laves ajax kald, der laver en ændring i robot-systemet. Ved sekventiel styring, som kalibrering af vision kameraer, er der udarbejdet bootstrap modals, der dukker op, alt efter hvad der skal ske i sekvensen.

Bootstrap modals defineres med strukturen, som set i listing 8. En modal skal være i klassen ”modal” for at blive defineret, som en bootstrap modal.

```

1 <div class="modal fade" id="choose-calib-modal">
2   <div class="modal-dialog modal-xl">
3     <div class="modal-content">
4       <div class="modal-header">
5         </div>
6       <div class="modal-body">
7         </div>
8       <div class="modal-footer">
9         </div>
10      </div>
11    </div>
12  </div>
```

Listing 8: Bootstrap modal struktur

Kalibrering

Et eksempel på sekventiel styring ses i figur 20. I eksemplet vises hvordan kalibrering af vision kamerasæt foregår på brugerfladen. Som det kan ses i figuren, kan stereo kalibrering ikke udføres. Dette er fordi denne ikke kan kaldes fra PLC.

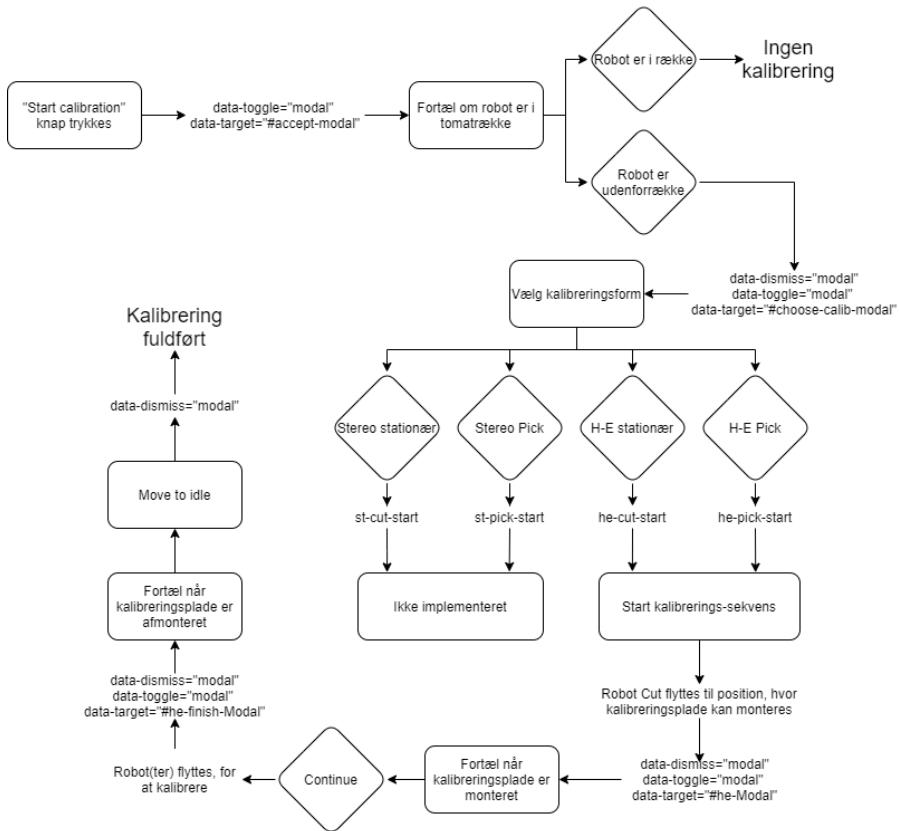


Fig. 20: Kalibrering ved brug af modals

Ved tryk på knappen ”Calibrate”, startes kalibrering af vision kamersæt(Kan ses i kapitel 4.5). Ved tryk på knappen, åbnes ”#accept-modal”. I denne modal bliver brugeren spurgt, om robot-systemet er flyttet ud fra tomat-rækkerne. Hvis dette er tilfældet, åbnes en ny modal, hvor kalibreringsform kan vælges (Se figur 21).

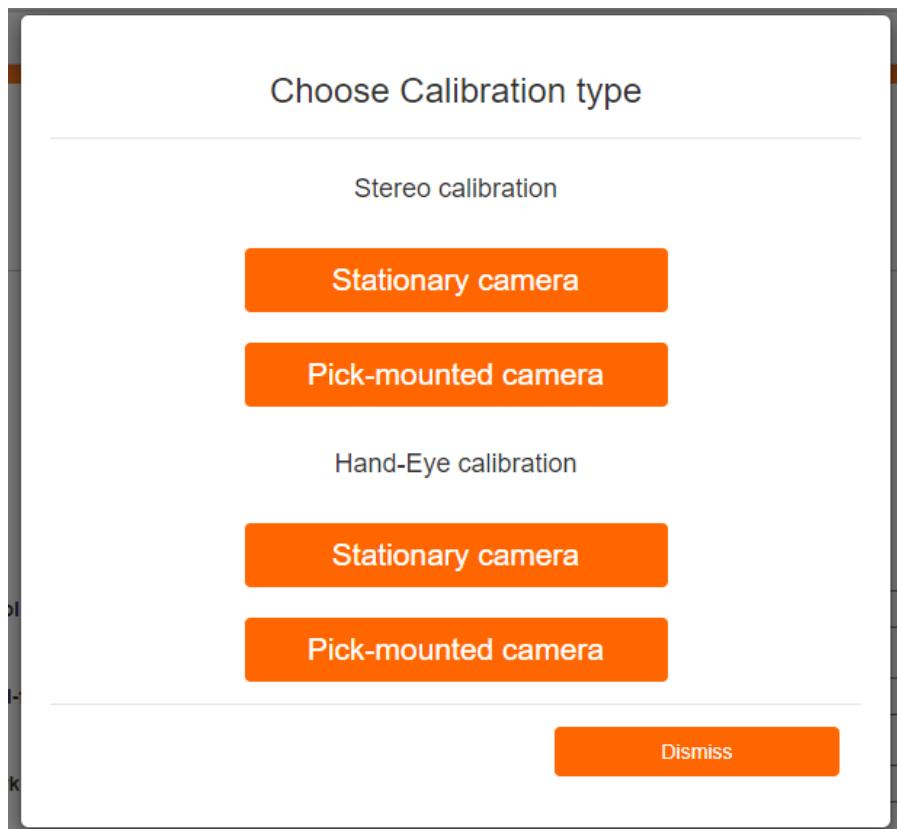


Fig. 21: Valg af kalibreringsform

Ved tryk på den valgte kalibreringsform, sendes en MQTT kommando til PLC, der starter den bestemte kalibreringssekvens. Hvis kaliberingsformen er Hand-eye, startes en sekvens, hvor Robot Cut bevæges ud, så en kaliberingsplade kan monteres. På samme tid åbnes en ny modal, der venter på, at pladen er monteret. Ved at trykke ”continue”, sendes kommandoen ”calib-mounted” vha. MQTT-forbindelsen.

Når PLC modtager ”calib-mounted”, forsættes kalibrering. Hvis det Pick monterede kamerasæt skal kalibreres, flyttes Robot Cut til en predefineret position. Robot Pick bevæges til et angivet antal konfigurationer, hvor kalibreringspladen kan ses. Robot Picks position og orientering sendes til vision-systemet, hvorefter et billede af pladen tages.

Hvis det stationære kamerasæt skal kalibreres, skal Robot Cut bevæges til et angivet antal positioner, hvor kamerasættet kan se kalibreringspladen. Positionerne er predefineret i forhold til kamerasættets position. Robot Cuts position og orientering sendes til vision-systemet, hvorefter et billede tages.

Når alle billeder er taget, bevæges robot(terne) til en position, hvor pladen kan afmonteres. Når pladen er afmonteret, og knappen ”Plate removed” er trykket, sendes MQTT kommando ”calib-unmounted” til PLC, hvorefter robot(terne) flyttes til start position.

Visning af data-log

Der vises data-log i alarm og log siden. Denne data er hentet fra databasen og vises i hhv. en `<div>` med klassen "log-table" og en `<div>` med klassen "alarm-log".

For at vise data, findes en `<div>`, der indeholder data-overskrifter. Ved hjælp af en "foreach" løkke vises, hver data-log i en `<div>` med samme struktur. Da `log.php` og `alarm.php` modellernes metoder `get10()` benyttes, vises de seneste 10 data-logninger. Begge modeller indeholder en metode `get10more()`, der viser 10 elementer mere. Disse benyttes, når knappen "moreLogs" eller "moreAlarms" trykkes, hvor et ajax kald henter 10 data elementer mere (Læs nærmere om Ajax kald i kapitel 4.3.2.3). For at vise 10 data-logninger mere, benyttes et nyt view (Enten `moreLogs.php` eller `moreAlarms.php`). Indholdet i disse views, har samme `<div>` struktur som hhv. `alarm.view.php` og `log.view.php`, hvor data-overskrifter findes i en `<div>`, og data skal vises i en `<div>` vha. en "foreach" løkke.

4.3.1.5 Partial views

Udover views, defineret til specifikke kontrollerse er der defineret "partial" views, hvilket er kode-dele, der bliver brugt i alle views, og derfor blot bør implementeres en enkelt gang.

Der er defineret tre partial views. Disse er til "menu", en "informations-bar" til menuen, og "foot".

Menuen `menu.php` er de øverste del, af alle views. Af denne grund indeholder menuen både `<head>` og `<body>`.

Elementet `<head>` er en container til metadata. HTML-metadata er data om HTML-dokumentet.

I `<head>` defineres hjemmesidetitel som "TPR HMI", samt logo defineres som Egatecs logo. I `<head>` refereres der til CSS og JavaScript filer. Da `<head>` er defineret i et partial view, der vises i alle andre views, benyttes samme style-sheets og script filer over hele brugerfladen.

I `menu.php` partial view, findes en menu, defineret som `<header>`. Menuen er struktureret som en `<div>`, der indeholder fire elementer. Disse fire elementer er; logo, "infobar", menu-knap og et "tilstand" felt (Se figur 22).



Fig. 22: Menu

Logo elementet virker som et hyperlink tag, der navigerer til login-siden. Informations baren er et iframe, der viser "infobar.php", hvor de seneste alarmer vises. Menu-knappen er lavet som en Bootstrap drop-down menu. Ved hjælp af bootstrap kan en drop-down menu defineres, som en `<div>` i klassen "dropdown". Knappen har klasserne; "btn", "btn-primary" og "dropdown-toggle". Da knappen er i klassen "btn-primary", får knappen et design defineret i Boot-

strap. Klassen ”dropdown-toggle” fortæller, at knappen bruges til at åbne en dropdown-menu.

Når knappen trykkes, åbnes den uordnede liste ”``”, der grundet klassen ”dropdown-menu” har et design, der er defineret af Bootstrap. Listen indeholder links til alle sider, som set i figur 23.

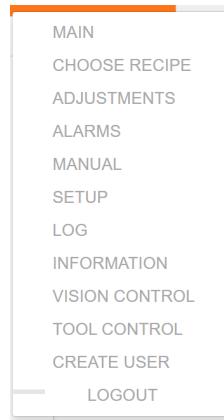


Fig. 23: Dropdown menu

Det sidste element indeholder information om systemets tilstand.

Elementet indeholder dato, driftstatus og brugernavn. Da den første side, der åbnes efter et login, er main (her hentes driftstatus), gemmes driftstatus i en `$_SESSION['opState']`. Denne session vises i ”tilstand”. På samme måde, er bruger gemt i en SESSION, hvilket også vises (Se figur 22).

Efter `<header>`, startes en `<div class="content">`. Denne div indeholder alle views.

På samme måde, som `menu.php` starter et view, benyttes `foot.php` til at lukke `<div class="content">` og `<body>`. I ethvert view, inkluderes `menu.php` i starten af filen, og `foot.php` i bunden af filen.

Det sidste partial view er `infobar.php`. Da dette view ikke er en del af MVC strukturen, har det ikke adgang til de forskellige models. Derfor indeholder `infobar.php` både en PHP klasse kaldet ”Infobar” og HTML markup.

I ”Infobar” constructor oprettes en database forbindelse, som i core klassen ”Database.php”. Udover constructor og destructor, findes en metode kaldet `get3()`, der henter de sidste alarmer, som `get10()` metoden gør i alarm modellen. For at de tre seneste alarmer kan vises i et iframe i ”menu.php”, er en `<div>` med klassen ”info-table” defineret. En `<div>` med klassen ”row” er lavet, der indeholder tre `<p>` tags til ”time”, ”alarm” og ”description”. Efter `<div>` med klassen ”row”, kaldes ”Infobar” klassens metode ”`get3()`”. Ved hjælp af et ”foreach” loop laves nye ”row” `<div>`, der indeholder en alarm med tidspunkt og beskrivelse (Se figur 22).

4.3.2 Public

I ”public” mappen findes en *.htaccess*⁶ fil. Denne fil definerer at */tpr/public* er roden af arkitekturen. Ydermere bliver *.htaccess* brugt til at definere regler for, hvis den ønskede URL ikke eksisterer.

Mappen indeholder også offentlige tilgængelige filer som CSS og JavaScript filer. I public mappen findes til sidst en *index.php* fil.

4.3.2.1 Index

Denne fil er den første fil, der åbnes, når der ønskes at anvende brugerfladen.

```

1 <?php
2 if (session_status() == PHP_SESSION_NONE) {
3     session_start();
4 }
5 require_once '../app/init.php';
6 $router = new Router();

```

Listing 9: index.php

I listing 9 ses indeholdet af ”index.php”. Siden bruges til at starte en SESSION. Efter en SESSION er startet, inkluderes init.php, som er den fil der inkluderer alle core filer. Når dette er gjort initialiseres en router, der er defineret i ”core”.

Blandt core programmerne findes *Restricted.php* og *Router.php*.

<http://192.168.29.56/public/main>

Fig. 24: URL eksempel

Grundet MCV arkitekturen ønskes fysiske stier ikke i URL. *Router.php* benyttes til, at læse URL for at finde den ønskede kontroller. Det er derefter kontrolleren, der viser et view. Figur 24 viser et eksempel på en URL, der kalder en kontroller.

⁶ HyperText Access = Konfigurationsfil, der bruges af apache-baserede webservere

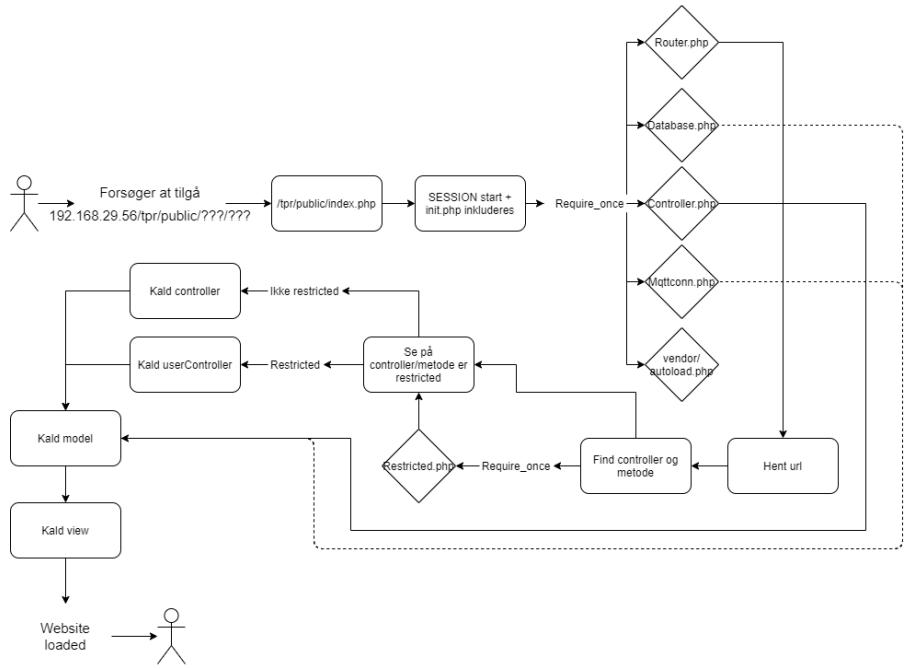


Fig. 25: Flowdiagram ved kald af MVC side

I figur 25 ses et flowdiagram over kald af kontrollere i den anvendte MVC arkitektur.

URL fra figur 24 hentes og separeres i et array adskilt af "/", hvorefter de tre første indeks negligeres. *Router.php* programmet undersøger, om det næste indeks i arrayet, er navnet på en kontroller. I eksempel URL i figur 24 er navnet på kontrolleren mainController.php. Efter dette undersøges næste indeks (Hvis tilgængelig). Hvis dette indeks ikke er tomt, indeholder indekset en metode i den pågældende kontroller. Hvis kontroller eller metode ikke findes, er en standard kontroller (index) defineret.

Efter at have undersøgt, hvilken kontroller der ønskes adgang til, tjekkes det om det er lovligt at tilgå kontrolleren. Dette gøres i *Restricted.php*.

For at undersøge om det er tilladt, undersøges det om kontrolleren findes i det gældende array. Det ene array indeholder adgangstilladelser, til hvis man ikke er logget ind. De sidste to indeholder adgangstilladelser til de to bruger-levels. Hvis kontroller og metode er tilladt returneres **true**, ellers returneres **false**. Hvis *Router.php* modtager **true**, kaldes den pågældende kontroller og metode, ellers vil brugeren blive omdirigeret til *userController*, hvilket er den kontroller, der kalder login siden.

Alle kontrollere har en standard metode (index), som er den, der kaldes, hvis en kontroller kaldes uden bestemt metode.

Efter kald af modeller, kaldes et view tilhørende den pågældende kontroller.

4.3.2.2 CSS

Cascading Style Sheets bruges til at definere, hvordan indhold af et HTML-dokument skal præsenteres. Da brugerfladen skal kunne åbnes på en mobil enhed, som en tablet, er CSS delt op i to. Dette vil sige, at generelle styles defineres og styles, der kun benyttes, hvis enheden befinder sig i ”portrait mode” defineres.

For at få indholdet på brugerfladen til at passe så mange enheder, som muligt defineres indholdets størrelse ikke, som absolutte måleenheder som pixels, cm og mm. I stedet benyttes relative måleenheder. Enheder, der skal fylde lige meget på siden, defineres med ”vh” og ”vw”, hvilket definerer størrelse i forhold til viewport højde og bredde. Her er 1 enhed det sammen, som 1% af hhv. højde og bredde.

Til skriftstørrelse benytte måleenheden ”em”, der beskriver størrelse i forhold til størrelsen på skrifttypen. I visse tilfælde ønskes en størrelse defineret i forhold til elementet, det befinner sig i. I disse tilfælde beskrives størrelsen i procent.

4.3.2.3 JavaScript

JavaScript koden benyttes til al styring, der sker efter siden er initialiseret. I dette projekt benyttes JavaScript til lave Ajax kald, når bestemte knapper trykkes. Dette gøres ved hjælp af jQuery.

Da alle Ajax kald laves på samme måde, er der defineret to metoder til udarbejdelse af kaldene. Den ene metode er til at lave et Ajax⁷ kald, og den anden laver et Ajax kald, hvor siden opdateres derefter.

```

1   function ajaxCall(url, commandData) {
2     $.ajax({
3       method: 'post',
4       url: url,
5       data: commandData,
6       error: function(e) {
7         alert("Error!! " + e);
8       }
9     );
10   }

```

Listing 10: Ajax kald

AJAX bruges til at udveksle data med en server og opdatere dele af en webside, uden at genindlæse hele siden. I listing 10 ses en asynkron post request. Metoden tager imod en URL og commandData. URL fortæller, hvilken metode i den nuværende kontroller, der skal kaldes. Variablen commandData er den data, der sendes med post request. Hvis den asynkrone post request ikke udføres succesfuldt, laves en alert med fejlen.

⁷ AJAX = Asynchronous JavaScript and XML

Metoden i listing 10 og `ajaxCallUpdate(url, commandData)` benyttes, hvis bestemte knapper i brugerfladen trykkes.

Når en side er initialiseret undersøges der, hvorvidt en knap trykkes. Et eksempel ses i listing 11.

```

1   $(".main-btn").on('click', function(event) {
2       var btnId = $(this).attr('id');
3       var url = 'main';
4       var mqttData = {
5           topic: "php-mqtt/hmi2plc/opState",
6           value: btnId,
7       }
8
9       ajaxCallUpdate(url, mqttData);
10  });

```

Listing 11: AJAX kald, der kalder main metoden i mainController

I listing 11, startes en metode, hvis en knap i ”.main-btn” trykkes. Hvis knappen trykkes, gemmes id på den bestemte knap (I `main.view.php` er id på knapperne hhv. ”start”, ”pause”, ”stop”), hvorefter URL defineres, som mainController metoden `main`.

Data der sendes som post request, gemmes i et JSON objekt⁸, hvorefter objektet sendes ved brug af metoden `ajaxCallUpdate(url, commandData)`. I eksemplet fra listing 11 kaldes mainController metoden `main`, der kalder `publish.php` metoden `mainOperation($_POST)`. Metoden sender kommandoen (id på den trykkede knap) qgennem MQTT til PLC. Efter kommandoen er sendt opdateres siden. Når siden opdaterer, kaldes mainController metoden `index()`, der eksempeltvis abonnerer på emnet ”mqtt/plc2hmi/opState”. Når PLC har modtaget en kommando med drifttilstand, sender PLC den nye driftstilstand tilbage og vises på brugerfladen.

Ajax kaldet, der bruges til at hente flere data-log, fungerer med samme princip som i listing 10. Variablerne ”logCount” og ”alarmCount” initialiseres når siden er initialiseret, og bliver sat til 10. Når en load-knap trykkes, kaldes en metode, der lægger 10 til variablen, hvorefter metoden `getMore()` kaldes på ”#log-table” med logCount så argument (Se figur 12).

```

1   var logCount = 10;
2   $(".moreLogs").mouseup(function(e) {
3       logCount = logCount + 10;
4       $("#log-table").load("getMore", {
5           logNewCount: logCount
6       });
7   });

```

Listing 12: Metode til at hente flere data-logninger

Load metoden indlæser data fra en server og placerer de returnerede data i det valgte element. Metoden `getMore()` i logController henter data, og gemmer

⁸ JSON = JavaScript Object Notation.

dem i *moreLogs.php*. Da moreLogs view returneres i det valgte element (#log-table), vil de næste 10 data-logninger vises i tabellen.

4.4 FJERNSTYRING AF ROBOT-SYSTEM

Inden udarbejdelse af brugerfladen blev påbegyndt, var udarbejdelse af robot-systemet allerede startet. Derfor skal det eksisterende robot-system ændres, således at styring kan ske gennem brugerfladen.

4.4.1 Kommunikation

For at brugerfladen kan styre robot-systemet, er det vigtigt, at der er komunikation mellem enhederne. Der benyttes en MySQL database forbindelse samt en MQTT forbindelse til at sende data. Både forbindelse til databasen og MQTT broker er udarbejdet i samarbejde Beckhoff.

4.4.1.1 Database

For at udarbejde en database forbindelse, skal et Database Server Projekt udarbejdes. Der oprettes forbindelse til den ønskede database ved hjælp af en "Connection String", som ses nedenfor:

```
Server=192.168.29.56;Database=tpr_db;Port=3306;Uid=myuser;Pwd=***;
```

Strenge indeholder addressen til databasen, samt database navn, port, brugernavn og adgangskode. For at få adgang til databasen, har der en *EasySql* funktionsblok udviklet i samarbejde med Beckhoff.

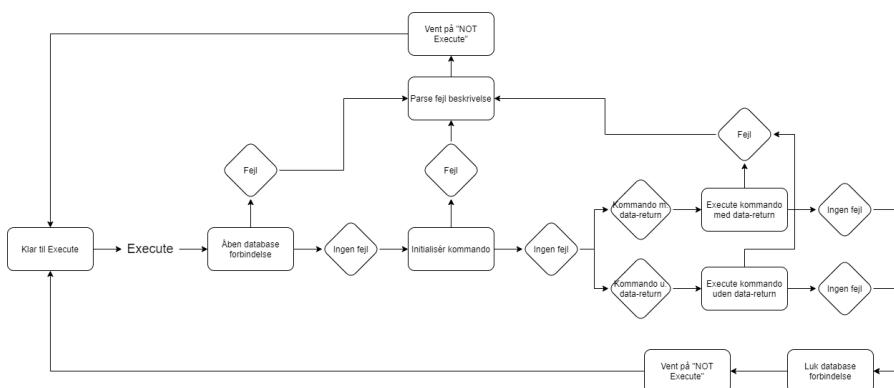


Fig. 26: EasySql

Som det ses i figur 26, åbnes en database forbindelse, efter Execute sættes høj. Ved brug af en *FB_SQLDatabaseEvt* funktionsblok forsøges der, at oprette forbindelse til databasen i Database Server projektet. Hvis en fejl forekommer, udskrives fejlen, ellers skiftes tilstand. Den næste tilstand er, hvor SQL kommandoen bliver forberedt. For at forberede kommandoen benyttes den tidligere nævnte funktionsbloks metode ”CreateCmd”, der tager imod en pointer til en kommando. Der undersøges om SQL kommandoen er en ”SELECT”, eller om den indsætter data i databasen. Når en kommando er udført, gemmes data, hvis kommandoen er ”SELECT”. Til slut lukkes database forbindelsen, hvorefter en ny kommando kan sendes.

Til at foretage SQL kald er, der udarbejdet program-blokke til ”INSERT”, ”UPDATE” og ”SELECT” kald. Disse program-blokke indeholder en SQL streng samt en EasySql instans. Hvis det er nødvendigt, benyttes en funktionsblok til at indsætte en værdi fra en variabel i strengen, således der kan laves et ”prepared statement”.

En program-blok, hvor databasen manipuleres kan se ud som i listing 13.

```

1 sSql := 'INSERT INTO tpr_db.alarmLog (alarmId, deviceWhich)
          VALUES ( 1 , 2 );';
2 F_Replace(pString:= ADR(sSql), cbStringLen:= SIZEOF(sSql),
            sSearch:= ' 1 ', sReplace:= TO_STRING(robotAlarmId));
3 F_Replace(pString:= ADR(sSql), cbStringLen:= SIZEOF(sSql),
            sSearch:= ' 2 ', sReplace:= TO_STRING(sDeviceWhich));
4 fbSql(
5   bExecute:= bInsert,
6   wDBID:= 1,
7   pSql:= ADR(sSql),
8   cbSql:= SIZEOF(sSql),
9   pData:= 0,
10  cbData:= 0,
11  udiMaxOfRecord:= 0,
12  udiNoOfRecords=> ,
13  bDone=> bDone,
14  bBusy=> ,
15  bError=> bError,
16  sError=> sError);

```

Listing 13: ”Insert” alarm logging

Her ses en udarbejdet SQL query, hvor værdien fra ”robotAlarmId” og ”sDeviceWhich” er indsat i strengen, som et ”prepared statement”. I funktionsblokken defineres, hvilken database, der arbejdes med, samt en pointer til kommando strengen. De input, der er sat til ”0” benyttes, hvis en SELECT kommando anvendes. I tilfældet, at retur data skal modtages, indsættes adressen til en struct, der skal modtage data. Det er vigtigt, at denne struct har den eksakte struktur, som det data der modtages.

4.4.1.2 MQTT forbindelse

For at skabe en MQTT forbindelse er et script, kaldet *P_Mqtt*, udviklet i samarbejde med Beckhoff. Dette script benyttes til, at skabe forbindelse til MQTT mægleren samt sende og modtage meddelelser. Programmet indeholder fem funktionsblokke: *A_Init()*, *A_Connect()*, *A_SendMessage()*, *A_Subscribe()* og *A_AdapterInfo()*.

A_Init()

Denne funktionsblok indeholder al information til at skabe forbindelse. Her defineres host, port og hvor lang tid en forbindelse skal være åben inden ”timeout”. Udeover dette defineres en *fbMessageQueue*. Denne benyttes, hvis data ikke kan nå at blive bearbejdet inden modtagelse af ny data. Til slut defineres, hvilke de topic, der ønskes at abonnere på og hvilke kommandoer skal sendes til.

A_AdapterInfo()

I denne blok undersøges forbindelsen til netværk. En funktionsblok ”FB_GetAdaptersInfo” benyttes til at finde alle netværk som PLCen har adgang til. Hvis et indeks i arrayet med netværk er det samme som MQTT mæglerens, sættes *bHaveNetwork* := TRUE;.

Funktionsblokken kaldes uafbrudt, resulterende i at *bHaveNetwork* skifter fra TRUE til FALSE.

A_Connect()

Connect benyttes til, at kalde *fbMqttClient*, hver gang *bHaveNetwork* er TRUE. Hvis en fejl opstår, sætte *bError* := TRUE;.

A_Subscribe()

Denne funktionsblok benyttes til at abonnere på ønskede topics. Der aboneseres på de topics, der er defineret i *A_Init()*. Hvis en meddelelse modtages, gemmes den i *fbMessageQueue*. Meddelser håndteres efter FIFO. Den første meddelse behandles først. Alt efter, hvilket topic meddelelsen modtages på, er specifikke handlinger defineret.

```

1 IF sTopicRcv = 'php-mqtt/hmi2plc/recipe' THEN
2   PROGRAM_DATA.usiNewRecipe := STRING_TO_USINT(sPayloadRcv)
3   ;
4   P_SqlSelectRecipe.bRead := TRUE;
5 END_IF

```

Listing 14: Subscribe eksempel

Listing 14 viser, at hvis en meddelelse modtages på topic ”recipe”, gemmes værdien i program data, hvorefter *P_SqlSelectRecipe.bRead* flaget sættes høj. Dette sørger for, at PLC henter resten af recept-data fra databasen.

A_SendMessage()

For at sende meddelelser, skal MQTT klienten have adgang til mægleren. Hvis dette er tilfældet, læser PLC om data, der er vigtig for brugerfladen, er ændret. For at sende en meddeelse benyttes metode "Publish" i funktionsblokken fbMqttClient. Metoden har brug for topic, som meddelelsen skal sendes til, en adresse til meddelelsen og størrelsen på den. For at sende meddelelsen sættes QoS og retain. For at sikre, at en brugerflade modtager alle meddelelser, sættes QoS til 2 (At most once) og retain sættes til TRUE. Grunden til at meddelelser fra PLCCen retained, er da det skal være muligt for en bruger, at logge på brugerfladen, og stadig modtage det nyeste data.

4.4.2 *Blok-styring*

For nemmere at styre robot-systemet gennem et interface, er vigtig kode, der benyttes flere steder, udarbejdet som funktionsblokke og separate programmer.

4.4.2.1 *Tool control*

Til styring af værktøj er en funktionsblok kaldet *ValveControl_1* udarbejdet. Funktionsblokken tager imod fire input: ControlType, Ack, Execute og ExecuteHmi. Der findes fire måder at benytte denne funktionsblok. Hvis ControlType er 1, styres en ventil tilsluttet PLCCen. ControlType 2 styrer en ventil på en af robotterne. Hvis en ventil på en robot manipuleres, modtages en "acknowledgement" fra robotten. Når en ventil tilsluttet PLCCen manipuleres benyttes en timer, for at være sikker på, at manipulationen er udført. ControlType 3 og 4 benyttes, hvis flere ventiler skal manipuleres sekventiel. Dette kan være når Robot Cut værktøjet skal lades eller aflades.

Input flaget Execute benyttes i plukke sekvenserne, hvor ExecuteHmi manipuleres af brugerfladen.

4.4.2.2 *Robotter til vente position*

Det kan forekomme, at en fejl er opstået i løbet af en plukke sekvens, og robotterne derfor står uhensigtsmæssigt. Hvis en operatør mener, at det ikke er forsvarligt at fortsætte sekvensen, skal det være muligt, at trykke på en knap, for at få robotterne tilbage til en vente position. Dette skal kunne gøres, uden at robotterne kolliderer uanset, hvilke konfigurationer robotterne befinner sig i (Se figur 27).

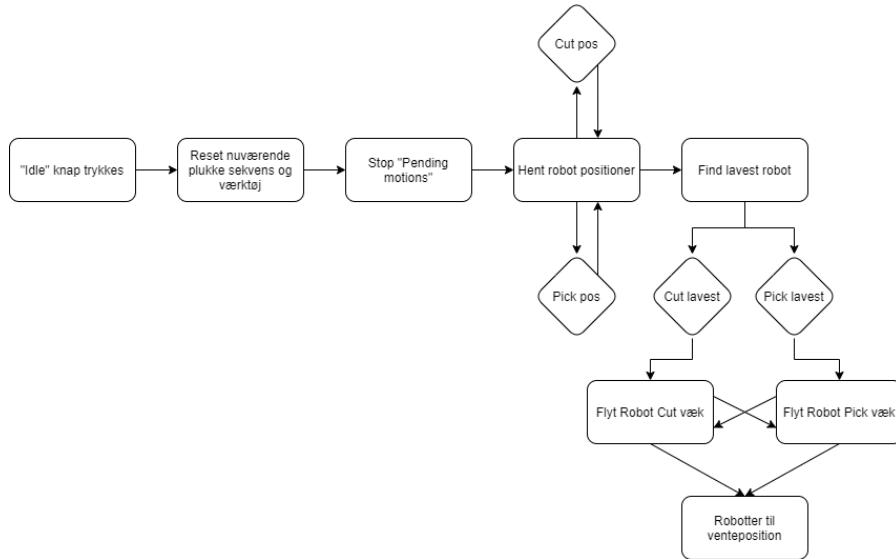


Fig. 27: Robotter til venteposition

Inden robotterne kan bevæge sig til venteposition, skal nuværende sekvens stoppes. Derudover skal værktøj nulstilles, så planter ikke trækkes med, når robotterne flyttes. Hvis robotterne var i gang med en bevægelse, stoppes denne og de næste i det nuværende robot-program.

Når alle bevægelser er stoppet, hentes begge robotters position, for at finde ud af, hvilken robot skal flyttes først. Det er bestemt, at den nederste robot skal flyttes først. Robotterne bevæger sig væk i hver sin retning. Det kan forekomme, at Robot Pick befinner sig i den retning, som Robot Cut skal bevæge sig i. Hvis Robot Pick er på denne side af Robot Cut, er det fordi, den var undervejs til tomatstilk, og vil derfor befinde sig over Robot Cut.

Når den nederste robot er flyttet, flyttes den sidste robot, inden de kan bevæge sig til venteposition, hvor de er klar til start af ny sekvens.

Ventepositionen for robotterne befinner sig indenfor vognen, og derfor vil det være muligt for vognen at køre videre, uden at robotterne fanger planter.

4.4.2.3 Kamera kalibrering

Inden implementering af brugerfladen, var det muligt, at kalibrere de to kamerasæt. Ved hjælp af en brugerflade er, der mulighed for, at personer uden teknisk viden om systemet, kan udføre kalibreringen. I figur 20 ses, hvordan kalibrering af kamerasæt fungerer med brug af bruger-interface.

Måden hvorpå sekvensen foregår er ved, at der abonneres på et MQTT topic "php-mqtt/hmi2plc/calib". Hvis der modtages en meddelelse på dette topic, gemmes den i en variabel kaldet `sNewCalib`. Hvis `sNewCalib` bliver "he-pick-start" og ingen andre sekvenser er i gang, startes hand-eye kalibrering for det Pick monterede kamerasæt. Når kalibrering startes, sættes et flag høj, der ændrer ventepositionen på Robot Cut til et punkt, hvor en kalibreringsplade

kan monteres. Når Robot Cut har bevæget sig til denne nye venteposition lukkes Cut, og pladen kan monteres. Så snart en MQTT meddelelsen ”calib-mounted” modtages, sendes robot kommandoen *FB_CALIB_CAM_ONE*. Da variablen til at ændre ventepositionen stadig er sat høj, betyder dette, at Robot Cut skal flyttes til en predefineret position, hvor Robot Pick kan se pladen og udføre kalibreringen i forhold til.

Efter kalibreringen, bevæges Robot Pick til venteposition, mens Robot Cut flyttes til den nye venteposition. Når meddelelsen ”calib-unmounted” modtages, bevæger Robot Cut tilbage til den originale venteposition.

4.4.2.4 Alarmhåndtering

Hvis en alarm-kode opstår på en af robotterne, sendes denne til PLCen. Udover alarmkoden sender Robot Cut information om, hvilken robot fejlen kommer fra. Figur 28 viser håndteringen af alarmer.

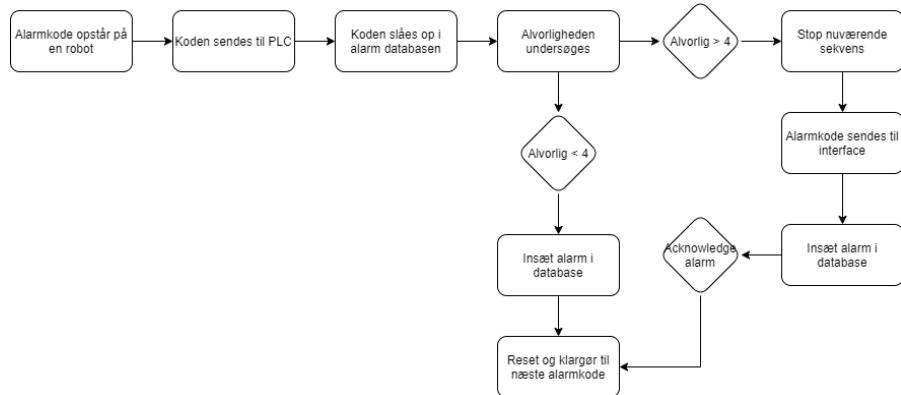


Fig. 28: Alarm håndtering

I databasen findes en tabel med alle robot alarm koder. I denne søges information om den nuværende kode. Tabellen indeholder [alarmId] [alarmDesc] [alarmCause] [alarmRemedy]. Alvorligheden af alarmen findes i ”alarmDesc”. En alarm beskrivelse har følgende struktur: ”4-PAUSE Cannot VP2 add to oneself”

Hvis tallet inden ”-” er over 4, vil det sige, at alarmen er en error.

For at finde tallet, skal det første mellemrum fjernes. Udover dette skal alt fra og med ”-” slettes. Når dette er gjort, kan strengen konverteres til en integer.

Hvis koden er en error kode, sættes robotter til `NoDriveOff := FALSE;` og kan derfor ikke køre. På PLC sættes bAlarm flaget høj.

Uanset om alarmen er en fejl eller advarsel, gemmes alarmen i database tabellen ”alarmLog”. Sammen med alarmen gemmes information, om hvilken enhed alarmen opstod på, og tidspunktet for alarmen.

Når en godkendelse modtages fra brugerfladen, sættes bAlarm lav igen, og robotterne får `NoDriveOff := TRUE;`

4.4.2.5 *Robotter til service position*

Hvis noget skal ændres på robotternes værktøj, skal robotterne flyttes til en position, og ændringerne kan udføres. Hvis en meddelelse sendes til topic ”php-mqtt/hmi2plc/serviceStart”, startes en sekvens, der får robotterne i en service position. Sekvensen stopper, når en meddelelse modtages på ”php-mqtt/hmi2plc/serviceStop”.

For at få robotterne i service position, nulstilles værktøj, så der ikke er luft i ventilerne. Robot-programmet *FB_ROBOT_SERVICE* startes, hvor robotterne bevæger sig til joint positionen {-169, 0, -90, 0, 0, 0} på Robot Pick og {169, 0, -90, 0, 0, 0} på Robot Cut. Her har de retning langs vognen, og det er også muligt at arbejde med værktøjet inde i tomatrækkerne. Når bearbejdelse af værktøj er færdiggjort, flyttes robotterne til venteposition.

4.4.2.6 *Data logning*

Data der logges er hhv., hvor eksekveringstiden pr. klip, antal billeder taget gennem klippe sekvensen, antal stickouts klippet undervejs, antal nye billede konfigurationer benyttet, antal forsøg, og hvorvidt sekvensen lykkes. I starten af enhver sekvens startes en timer til at definere eksekveringstiden. Timeren stoppes i slutningen af sekvensen. Hver gang et af de andre log-emner udføres inkrementeres en tæller til disse. Hvis et klip er succesfuld, sættes et flag højt i slutningen af sekvensen. Hvis denne del af sekvensen ikke nåes, har klippet ikke været succesfuld.

Når tilstandsmaskinen i kontrol sekvensen bliver 0, gemmes log-data og lægges i databasen.

4.5 DESIGN

I dette kapitel vil information findes om valg af design og opdeling af sider. Det er vigtigt at applikationen er overskuelig og nem at håndtere, og derfor er det vigtigt, at applikationen er designet efter dens formål. Da formålet er at betjene et robot-system, er det vigtigt, at knapper og tekst nemme at finde. Enheden der skal vise applikationen, vil i de fleste tidspunkter være håndholdt. Derfor er det vigtigt, at hver enkelt side ikke indeholder for meget information, således elementer kan blive skygget. Det er også vigtigt, at siderne er ændres efter hvilken orientering enheden har, således at siderne forbliver overskuelige. I bilag 3 og 4 kan skærbilleder af alle brugerfladens sider. Det er vigtigt, at applikationen passer til virksomhedens branding. Da Egamatic benytter farven orange, benyttes denne, som den primære farve i applikationen. Da arbejde i drivhuse er et ”beskidt” arbejde, benyttes den hvide nuance, da det giver et ”clean” udseende.

Som ses i figur 29 overholder design valg, ved kun at vise det mest nødvendige på siden.

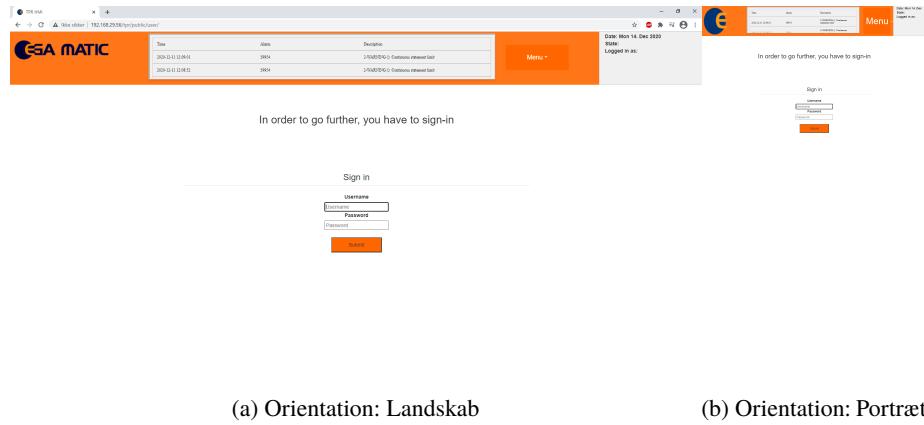


Fig. 29: Login side

Som ses i figur 30, ændres siden efter hvilken orientering enheden har. For at siden ikke kan scrollle til siden, er det valgt, at flytte vognstyrings-knapperne ned, for at udnytte, at skærmen er længere i portræt orientering.

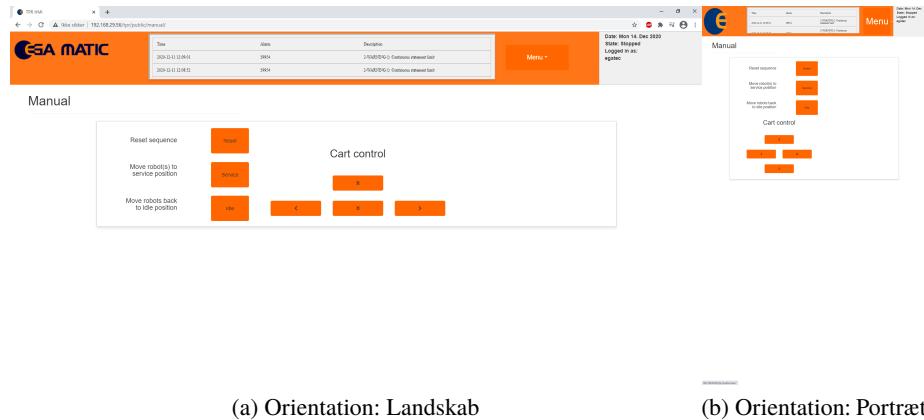


Fig. 30: Manuel side

På nogle sider, skal meget data vises, som ses i figur 31.

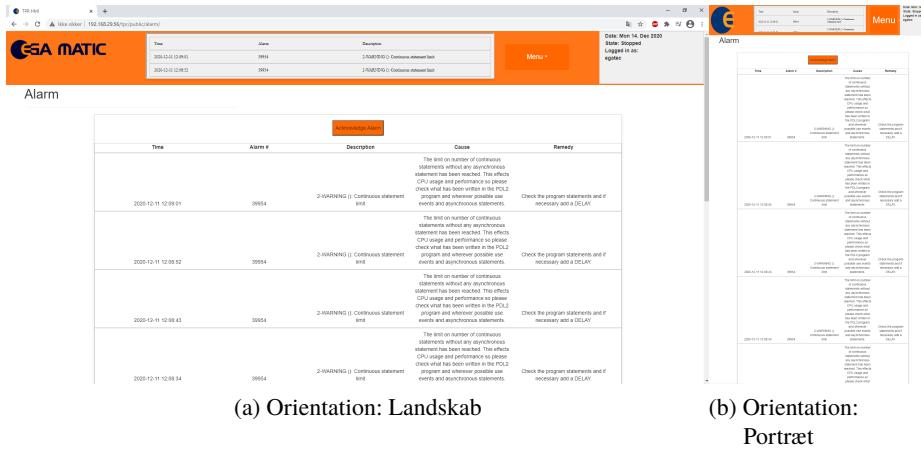


Fig. 31: Alarm side

For at gøre siden mere overskuelig, er det valgt, at data kun skal vises på 75% af sidens bredde.

For at vise data, anvendes PHP inde i HTML markup tags. For at vise alarmoverskrifterne, var disse gemt i en <div>.

```

1  <?php
2      foreach($viewbag['alarm'] as $alarm) :
3  ?>
4  <div class="row" style=" border:1px solid #cfcfcf;">
5      <p><?= $alarm['time']?></p>
6      <p><?= $alarm['alarmId']?></p>
7      <p><?= $alarm['alarmDesc']?></p>
8      <p><?= $alarm['alarmCause']?></p>
9      <p><?= $alarm['alarmRemedy']?></p>
10     </div>
11
12 <?php endforeach; ?>
```

Listing 15: For-løkke til visning af alarmer

For at vise alarm-data, benyttes en PHP for-løkke. For enhver alarm laves en <div> med samme struktur som den <div> med alarmoverskrifterne. "alarm.view" blev kaldt med en "\$viewbag". For enhver alarm i denne "\$viewbag" vises hvornår den pågældende alarm er opstået samt alarmens information. Ved tryk på en knap nederst på siden, udføres et AJAX kald, der viser de næste 10 alarmer i alarm <div>.

For at mindske indhold på siderne benyttes modals. Ved tryk på "Acknowledge alarm" knappen, åbnes en modal, der viser den sidste forekommet alarm. I denne modal er det muligt, at sende en godkendelse af alarmen.

4.6 SIKKERHED

I dette kapitel vil, der gives et overblik over, hvilke tanker der har lagt bag sikkerheden. Disse tanker er både i forhold til kommunikation og hvilke tanker der ligger bag virkemåden af brugerfladen.

4.6.1 *Interface*

Da brugerfladen benyttes til kontrollering af drift af robot-systemet, er det vigtigt, at aktører udefra ikke har adgang til brugerfladen.

Derudover ønskes det, at al interaktion med robot-systemet skal ske gennem brugerfladen. For at muliggøre dette, benyttes ingen "GET" requests, da der ikke ønskes drift-data i URL.

4.6.1.1 *Netværk*

Idéen med brugerfladen er, at der kun skal være mulighed for at kontrollere robot-systemet med adgang til robot-netværket. Dette kan enten ske med at være fysisk tilstede eller gennem en VPN forbindelse. For at dette er muligt, findes et maskine-netværk og et netværk, der giver adgang til internettet.

Alle enheder i robot-systemet har adgang til maskine-netværket og har derfor adgang til hinanden. For at få adgang til maskine-netværket, er det nødvendigt, at være fysisk til stede. Hvis adgang til maskine-netværket er etableret, er det muligt for en operatør, at få adgang til brugerfladen.

Måden hvorpå netværket er beskyttet, er ved hjælp af en adgangskode. Der antages, at personer, der har fysisk adgang til netværket, ikke har ondsindede hensigter og derfor menes det, at denne sikkerhed er tilstrækkelig.

Da det skal være muligt at tilgå maskine-netværket fra et fjernt netværk, findes et netværk på robot-systemet, der kan oprette forbindelse til internettet. RevPi har adgang til dette netværk gennem port A, og får tildelt en offentlig IP adresse gennem DHCP. På RevPi findes en OpenVPN forbindelse, der skriver til en server udarbejdet af trendlog.io, hvor RevPi er købt. Hos Trendlog kan VPN forbindelse til RevPi opnåes gennem deres interface **StmLogic Gatekeeper Manager**, og på denne måde skabe adgang til maskine-netværket gemmen RevPi.

Der antages, at Trendlog ikke giver uvedkommende adgang til VPN forbindelsen, og derfor kan det antages, at denne forbindelse også er sikker.

4.6.1.2 Database

Skulle det ske, at uvedkommende får adgang til maskine-netværket, bør foranstaltninger indføres, for at gøre uønskede handlinger besværlige at udføre. For at benytte brugerfladen, skal der være logget ind på systemet. Da al login information findes på en database på RevPi, er det vigtigt, at disse ikke kan læses direkte i databasen. Derfor, når en bruger oprettes, bliver adgangskoden hashed inden login information gemmes i databasen. På tidspunktet bliver adgangskoder hashed med hashfunktionen *MD5*.

En af de mest brugte web-angreb til at hente følsomme data er SQL-injektion. SQL-injektion er en teknik, hvor angriberen indsætter en SQL query i et input felt, der derefter behandles af den underliggende SQL-database.

For at modvirke dette, bliver variable, der sendes til databasen, ”sanitized” inden en SQL query udføres. Sanitation benyttes til at fjerne uønskede karakterer i en streng, hvilket gør, at hvis man eksempeltvis prøver at skrive brugernavnet **brugernavn'; –⁹**, så slettes ‘;–, så den ønskede datastruktur modtages.

4.6.2 Fysisk sikkerhed og præservering af robotter

Da intentionen er, at robot-systemet styres gennem brugerfladen, er det vigtigt, at kunne nulstille systemet efter en kollision eller en dårlig robot konfiguration. Det kan forekomme, at en fejl opstår på robot-systemet. Hvis en operatør mener, at robot-systemet ikke kan fortsætte igangværende sekvens, findes en knap på brugerfladen, der nulstiller alle sekvenser, og flytter robotter til venteposition, uden at de kolliderer undervejs.

Denne funktionalitet er vigtig, da det er uhensigtsmæssigt at tilslutte robot teach-pedant for at flytte robotterne manuelt.

Da både robotter og kameraer skal kalibreres, er det vigtigt, at disse kan tjekkes. Hvis kalibrering ikke er optimal, opererer systemet ikke optimalt og der er risiko for uhensigtsmæssige bevægelser.

For at modvirke potentiel ødelæggende bevægelser, er det muligt at udføre kalibrering. Ved tjek af kalibrering, startes en sekvens, der flytter Robot Pick til et punkt, der er udprikket på et billede fra vision-systemet. Her kan man se, hvor tæt på det ønskede punkt Robotten kommer.

Ved tjek af robot kalibrering, skal robotterne flyttes til en joint position, hvor man kan se om leddets referencehak er placeret korrekt ift. hinanden.

⁹ ”–” laver resten til kommentar

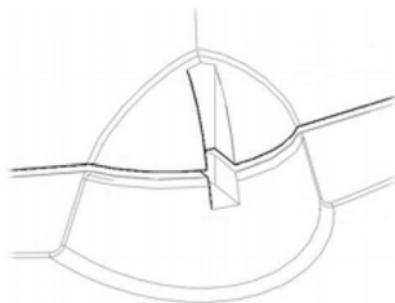


Fig. 32: Referencehak led 1

Hvis referencehak ikke er placeret som i figur 32, bør teach-pedant tilsluttes, for at kalibrere robotten.

5

ALTERNATIVER

Ved udvikling af et produkt, må man tage nogle valg, for at få den ønskede virkemåde. I dette kapitel, vil der beskrives andre alternativer, hvor deres fordele og ulemper vil beskrives.

5.1 BRUGERFLADE

Brugerflader til robot-systemer kan udarbejdes på forskellige måder. Enhver måde har sine begrænsninger og fordele. I denne sektion ville disse beskrives.

5.1.1 *Monteret brugerflade*

Hvis en brugerflade, der er monteret på et robot-system ønskes, er det vigtigt at montere brugerfladen et sted, hvor det altid kan tilgås. Dette er ikke muligt for et produkt, der skal arbejde inde i tomatrækker. Dette skyldes, at brugerfladen skal monteres i en af robot-systemets ender, og robot-systemet ikke altid vil have den korrekte ende vendende mod indgangen til rækken.

På en brugerflade, der er monteret direkte på robot-systemet, er det muligt at montere et nød-stop, så det er let tilgængeligt. Derudover kan man sikre sig, at når man befinder sig bag brugerfladen, vil man være indenfor en sikkerhedsafstand.

Hvis det ønskes, er der mulighed for at gemme driftdata i en stationær database, hvis robot-systemet er tilsluttet samme netværk. For at dette er muligt, bør robot-systemet flyttes til en position, hvor netværket, den stationære database befinder sig på, kan tilgås.

Problemet med en fastmonteret brugerflade til TPR robot-systemet er, at robot-systemet er mobilt, og derfor skal robot-systemet først lokaliseres for at kunne betjenes. Derudover, hvis robot-systemet befinner sig i en gang mellem to tomatrækker, er der risiko for ikke at have adgang til brugerfladen.

5.1.2 *Kablet brugerflade*

En Brugerflade kan uover at være fastmonteret, også være løs-monteret i robot-systemet med en kabelt forbindelse. På denne måde, kan brugerfladen have samme fordele, som den fastmonterede brugerflade. Der er ydermere mulighed for at flytte brugerfladen, så den altid kan tilgås uanset, hvordan robot-systemet befinner sig i en tomatgang. Til forskel for den fastmonterede brugerflade, kan man ikke være sikker på, at man befinder sig med korrekt sikkerhedsafstand fra systemet.

Fordelen ved, at en trådløs brugerflade, der i dette projekt er den anvendte form for brugerflade, er at brugerfladen kan tilgås fjernt, såfremt det korrekte netværk kan tilgås. Uover dette vil det være nemmere, ved brug af lokalitetsmetoder, at finde det ønskede robot-system. Ydermere vil det være nemmere at styre flere robot-systemer i et drivhus, da man ikke behøver at være fysisk tilstede.

Problemet med den trådløse løsning er sikkerheden, da systemet kan styres uden at være fysisk tilstede. Endvidere er der risiko for, at man betjener robot-systemet uden at befinde sig indenfor den angivede sikkerhedsafstand.

5.2 KOMMUNIKATION

For at kunne udarbejde styring til et robot-system, skal der benyttes en eller flere kommunikationsprotokoller. I denne sektion vil der undersøges, hvilke kommunikationsprotokoller, der bliver benyttet til automationsinterface og se hvilke fordele og ulemper disse besidder.

Der vil fokuseres på kommunikation mellem brugerflade og PLC. Kommunikation til resten af robot-systemet vil negligeres, da disse er arbitrær for brugerfladen.

5.2.1 *OPC-UA*

OPC-UA¹ er en maskine-maskine kommunikationsprotokol til industriel automation[14]. OPC er baseret på klient / server kommunikation, og er ikke begrænset af operativ system eller programmeringssprog. OPC-UA tilbyder sikkerhedsfunktioner til godkendelse, autorisation, integritet og fortrolighed. Ved brug af OPC-UA er det muligt at benytte multi-treads, hvilket vil gøre det muligt at sende og modtage flere datasæt samtidig.

Da der ikke findes en open-source PHP klient, blev denne protokol ikke anvendt.

¹ OPC-UA = Open Platform Communication Unified Architecture

5.2.2 *UDP*

UDP er en del af Internet-protokolstakken. UDP giver ingen garanti for at data modtages. Efter data er sendt, får afsenderen ikke besked hvis data ikke kommer frem. UDP er ”connectionless”, hvilket betyder, at der ikke oprettes en session, og der garanteres ikke, at data modtages. Da UDP protokollen ikke benytter bekræftelse, der sikrer at data er modtaget, er UDP protokollen hurtigere end TCP.

Da UDP protokollen ikke sikrer at data modtages, er den ikke nyttig til dette projekt. Til dette projekt er det vigtigere, at data modtages, end hvor hurtigt det sendes.

5.2.3 *TCP-IP*

TCP-IP er en af de vigtigste protokoller i internetprotokolpakken. TCP-IP giver pålidelig, sorteret og fejlkontrolleret levering af en datapakker mellem applikationer, der kommunikerer via et IP-netværk.

En TCP-IP forbindelse, er en god idé, da der sikres, at data modtages grundet TCP handshake.

En ADS-route var ønsket som kommunikations protokol mellem brugerflade og PLC. ADS protokollen er baseret på TCP-IP. Ulempen med ADS er, at en klient udarbejdes i JavaScript og ikke PHP. Dette er en ulempe, da JavaScript bliver afviklet lokalt på den enkelte browser. På grund af dette, kan der læses variabel navne og AMS ID direkte i browseren. Udeover dette risikerer man, at den lokale web browser ikke er opsat korrekt, og JavaScript koden bliver ikke afviklet.

Den valgte kommunikationsprotokol mellem brugerflade og PLC er MQTT, hvilket også normalt kører over TCP-IP.

6

DISKUSSION

I dette kapitel vil projektets resultater blive drøftet i henhold til projektets mål og de funktionaliteter, der blev stillet til systemet. Resultaterne evalueres for at se, om de når målsætningen og de ønskede funktionaliteter, der var opstillet i starten af projektet.

I kapitel 2 blev det defineret, at der skulle udarbejdes en database, der skal indeholde vigtig information om robot-systemet. En mariaDB database blev udarbejdet og indeholder vigtig data, som login oplysninger, recepter, nuværende program og hastighed, og roboternes tool-frame og work-frame. Da robot-systemet ikke virker uden recept data og program data, antages det, at disse data er de mest essentielle for robot-systemet og derfor er funktionaliteten opfyldt. Ydermere var login oplysninger nødvendig for at udarbejde en brugerflade med bruger-niveauer og det er derfor ligeledes vigtigt, at disse er gemt i databasen.

Da et ønske var, at der kan udføres trendlog, gemmes denne information også i databasen. Til trendlog gemmes eksekveringstid, antal billeder taget, stickout klippet, antal ændringer af billedepositioner, og klippe succes. Disse emner er brugbare til dataindsamling for at optimere drivhuset, dog kan det diskuteres at yderligere emner kan være mere brugbare.

En ønsket funktionalitet var, at der skal kunne udføres alarm håndtering. Til dette er alle robotalarmer gemt i databasen. Alarmer, der opstår på roboterne, gemmes i en database tabel "alarmLog". Enhver alarm gemmes, som tidspunkt hvor alarmen opstår, alarm nummer, beskrivelse af alarmen, årsagen til alarmens forekomst, og hvad man skal gøre for at modvirke alarmen. Når en opstået alarm er defineret, som en "error", sendes alarmnummeret til brugerfladen. Måden hvorpå alarmen fjernes er ved, at brugerfladen sender samme alarmnummer tilbage til robot-systemet, hvorefter alarmen fjernes og roboterne sættes i "Drive on".

Måden hvorpå alarmer logges fungerer hensigtsmæssigt, dog ønskes fejl håndtering udføres anderledes. Det er ikke hensigtsmæssigt, at alle fejl håndteres ved at sende en godkendelse til roboterne, hvorefter de sættes i "Drive on". Fejlene bør håndteres, som beskrevet i "Alarm remedy" men eftersom, der findes 8829 alarmer¹ til roboterne, er dette ikke implementeret.

Til recepthåndtering, blev recepter defineret med følgende information: Robot hastighed, drifttype, pakkemønster, vognhøjde, ventil forsinkelse, antal

¹ antal alarmer modtages fra Comau

billedkonfigurationer og kassestørrelse. En recept hentes ved hjælp af dens receptnummer. Nye recepter kan defineres ved at angive et ubrugt receptnummer. Håndtering af recepterne sker efter hensigten, dog bør yderligere receptdata defineres, når TPR robot-systemet bliver mere automatisk.

Det er forsøgt at designe brugerfladen, så den er simpel at navigere. Siderne i brugerfladen indeholder få store elementer, der gør brugerfladen overskuelig og nem at finde information på. Der kan dog forekomme problemer med sekventielle styringer fra brugerfladen.

Ved start af en sekventiel styring, åbnes en Bootstrap modal. Det er muligt at lukke en modal uden at stoppe den startede sekvens.

Da brugerfladen kun er testet af udviklere af TPR robot-systemet fra Egatec, kan det ikke kvantificeres, hvorvidt brugerfladen har et simpelt design, der er nem at navigere.

Ved kalibrering af vision kamerasæt, fortælles brugeren løbende, hvordan kalibrering af kamerasæt skal udføres. Ved denne sekventielle styring, åbnes en ny modal, efter hver kommando. Dette resulterer i, at der er risiko for, at brugeren for hurtigt kan trykke igennem kalibreringen, og robot-systemet når ikke at blive færdig med kalibrering. Der er af denne grund risiko for, at kalibreringen forværres.

Målsætningen, opsat i starten af projektet, definerer at en webbaseret brugerflade, der kan tilgås fra alle enheder i samme netværk, skal udarbejdes. Brugerfladen skal have forskellige bruger-niveauer til begrænset adgang og funktionaliteter. Målet med produktet er, at det ikke er nødvendigt at benytte en programmør med adgang til PLC og robotter for at styre robot-systemet.

En brugerflade med to bruger-niveauer er udarbejdet. Måden hvorpå restriktioner til funktionaliteter forekommer er ved, at der sørges for, at brugeren ikke har adgang til bestemte kontrollere. Ved at skabe restriktioner for hele kontrollere, sørges der for, at brugere ikke har adgang til de views, der vises af kontrollere.

I stedet for at lave restriktioner til hele kontrollere, kunne man med fordel lave restriktioner til bestemte metoder i stedet. På denne måde kan, der oprettes forbindelse til alle views og lave restriktioner på funktionaliteter i stedet. Ved at huske, hvilke funktionaliteter, der ikke kan benyttes, kan man ændre udseendet på de forskellige views for at vise, at bestemte funktionaliteter ikke kan benyttes.

Det antages, hvis robot-systemet opererer fejlfrit og brugerfladen benyttes korrekt, at robot-systemet vil kunne kontrolleres gennem brugerfladen. Da disse omstændigheder ikke er muligt, kan brugerfladen endnu ikke erstatte en programmør. Eftersom alarm håndtering ikke fungerer som ønsket, kan alarmer på robotterne ikke altid fjernes uden adgang til robotterne gennem et teach panel.

6.1 PROJEKT UDFORDRINGER

Grundet omstændighederne med COVID-19, har der ikke været optimale muligheder for, at få vejledning. Da omstændighederne resulterede i mangel på fysisk kontakt, har det været problematisk at få kontakt til vejlederen. Ydermere, grundet manglen på den fysiske kontakt, har der forekommet problematikker med at få adgang til robot-systemet, der befinner sig hos Alfred Pedersen & søn.

Eftersom TPR projektet er et udviklingsprojekt, blev der ikke udviklet en brugerflade ikke til et færdiggjort system. Af denne grund, har antagelser om virkemåden af robot-systemet været nødvendig.

KONKLUSION

Projektets mål var at udarbejde en brugerflade, der kan styre TPR robot-systemet. Ved at lave en server på en industriel Raspberry Pi, har det været muligt at lave en hjemmeside til styring af robot-systemet. Da RPi er tilsluttet samme netværk som robot-systemet, er det muligt, ved hjælp af MQTT kommunikationsprotokollen at sende kommandoer fra hjemmesiden til robot-systemet.

Udover at benytte MQTT protokollen, er databasen defineret således, at både brugerflade og robot-system kan skabe adgang til databasen. Dette har været til stor værdi, da kommunikation mellem brugerflade og robot-system er minimalt. Hvis brugerfladen skulle hente data fra databasen, for at sende disse til robot-systemet, ville kommunikation være langsommere. Ydermere vil det skabe problemer for drift af robot-systemet, hvis en brugerflade ikke er tilsluttet robot-systemet.

Der er udarbejdet en brugerflade, hvorpå et login-systemet findes. Hver bruger har et ”access-level” nummer. Alt efter hvilket nummer der benyttes vil, der være restriktioner på forskellige kontrollere i MVC strukturen. Brugerfladen er designet således, at elementer er store, så de er nemme at se, og brugerfladen bliver mere overskueligt. Da overskueligheden ikke er testet på respondenter, der ikke kender til robot-systemet, kan det ikke konkluderes hvorvidt designet er overskueligt.

For at robot-systemet kan styres gennem brugerfladen, er den eksisterende PLC kode ændret. Sekvenser i robot-systemet bliver nu startet, hvis bestemte kommandoer sendes fra brugerfladen. Vigtige sekvenser er defineret, som funktionsblokke eller programsekvenser, så disse både kan kaldes fra brugerfladen, men også benyttes direkte i klippe-sekvenser. Eksempelvis er værktøjsstyring lavet som en funktionsblok, så al styring af værktøj sker i samme funktionsblok.

Robotkonfigurationsændringer kan ske gennem brugerfladen vha. definerede sekvenser i robot-systemet. Da robot frames kan ændres på brugerfladen, vil det være muligt at ændre, hvordan robotterne bevæger sig. Ydermere, hvis robotterne befinner sig i en konfiguration, hvor en operatør mener, at robotterne vil kolidere, kan robotterne flyttes til en venteposition.

Ved tryk på knappen ”idle”, flyttes robotterne til venteposition uden at kolidere. Dette sker ved at finde roboternes position i forhold til hinanden, hvor de flyttes efter denne.

Håndtering af input fra brugerfladen sker ved hjælp af MQTT protokollen og gennem database forbindelsen. Ved at sende en kommando gennem MQTT forbindelsen, får robot-systemet besked om at hente yderligere data fra databasen.

Data logges i databasen fra robot-systemet. Driftsdata findes i løbet af en klippesekvens. Når en sekvens er slut, gemmes al data, som derefter sendes til databasen. Så snart en alarm på en robot opstår, sendes alarmen til PLC'en, hvorefter den evalueres, inden den gemmes i databasen.

Håndtering af alarmer sker ved, at information om den givne alarm hentes fra databasen. Hvis alarmen er over 4, er alarmen en "error". Hvis dette sker, stoppes robotterne. Uanset om alarmen er en fejl eller advarsel, gemmes fejlen i alarmLog i databasen. En Fejl bliver godkendt ved at sende alarmnummeret fra brugerfladen til robot-systemet.

Det kan konkluderes, at et produkt til styring af TPR robot-systemet er udarbejdet. Produktet kan kontrollere robot-systemet dog med begrænsninger.

Håndtering af alarmer fungerer ikke optimalt, og fysisk sikkerhed ved brug af brugerfladen kan forbedres.

I målsætningen var der beskrevet, at brugerfladen skal kunne sørge for, at en programmør ikke behøves for at styre robot-systemet. Det kan konkluderes, at dette ikke er muligt. Dog vil brugerfladen kunne være et værktøj som programmøren kan benytte til yderligere udvikling af robot-systemet.

8

FREMTIDIGE IMPLEMENTERINGER

I dette kapitel, vil fremtidige revisioner beskrives. Disse revisioner består både at ændringer og tilføjelser, der vil give værdi til hele TPR projektet.

8.1 RISIKOVURDERING

Hvis fremtidige revisioner skulle forekomme, er dette emne det første det skal udføres.

Risikovurderinger er en struktureret fremgangsmåde, hvor vurderingen af risici bliver så objektivt som muligt. Risici er populært sagt fremtidige problemer, der ikke har vist sig endnu. Da man ikke kan se ind i fremtiden, må risikovurderinger derfor altid være en kombination af erfaringer, forestillingsevne og følelser og fornemmelser. Risikovurderingen identificerer potentielle risici ved et produkt og definerer handlinger for at modvirke disse risici[15].

For at brugerfladen kan benyttes, skal en risikovurdering udarbejdes for at vise, at opgaveløsningen foregår på forsvarlig vis i forhold til sikkerhed.

8.2 ASYNKRON ABONNERING

En anden implementering, der har stor betydning for robot-systemet er, at brugerfladen kan abonnere asyntkront på information fra PLC. Dette giver stor værdi, da alarmer kan forekomme på brugerfladen, så snart alarmen forekommer. På denne måde findes alarmen med det samme i stedet for, først at observere, at robot-systemet er gået i fejl.

For at dette er muligt, er det vigtigt, at brugerfladen læser efter kommandoer i baggrunden således, at brugerfladen kan opdateres løbende.

8.3 VISNING AF VISION BILLEDER

Det kan forekomme, at en fejl på robot-systemet opstår. Derfor giver det mening at vise det sidste billede, taget inden fejlen opstod.

En fremtidig implementation kan være at, hvis en robot går i fejl, gemmes det sidste billede i alarmLog tabellen i databasen. Fejlsøgning vil være kortere da,

der er mulighed for at undersøge om fejlen forekom grundet et forkert punkt fra vision systemet.

8.4 UPLOAD AF DATA TIL STATIONÆR DATABASE

Da RevPi, med server til brugerfladen, har en bestemt kapacitet (4GB), er det vigtigt, at enheden kan upload data til en større database, når enheden er ved at være fuld. Denne database med højere kapacitet, kan være en stationær database, der kan opdateres når er databasen på robot-systemet er fuld. Denne implementation vil ikke blot give værdi, da mere data kan gemmes. Hvis flere robot-systemer benyttes i samme drivhus, da data fra alle robot-systemer gemmes samme sted, kan data samles til sammenlignes og optimering af driften i drivhuset.

8.5 VEDLIGEHOLDELSE AF TPR

Alle maskiner og robot-systemer skal vedligeholdes periodisk. Af denne grund vil en mulig implementering være, at brugerfladen sørger for systematisk vedligeholdelse af robot-systemet.

Da robot-systemet logger al driftdata, vil det være muligt at definere, at klinger på Robot Cut skal udskiftes efter eksempelvis 10000 klip. Det vil endvidere være muligt at fortælle, hvornår vision-data skal hentes eller hvornår klippeværktøj skal serviceres.

9

LITTERATUR

[1] MQTT

Udgivet af Wikipedia

<https://en.wikipedia.org/wiki/MQTT>

Besøgt d. 25. september 2020

[2] CANopen Explained

Udgivet af CSS Electronics

<https://www.csselectronics.com/screen/page/canopen-tutorial-simple-intro/language/en>

Besøgt d. 23. oktober 2020

[3] ADS introduktion

Udgivet af Beckhoff

https://infosys.beckhoff.com/english.php?content=.. /content/1033/tcadcommon/html/tcadcommon_intro.htm& id=

Besøgt d. 23. oktober 2020

[4] MVC Architecture & Its Benefits in Web Application Development

Udgivet af techaffinity

<https://techaffinity.com/blog/mvc-architecture-benefits-of-mvc/>

Besøgt d. 23. oktober 2020

[5] Our Technology

Udgivet af Root AI

<https://root-ai.com/#tech>

Besøgt d. 6. November 2020

[6] The Project

Skrevet af PZB Egattec A/S

Bilag #6

[7] Qii-Lift H-350 pipe rail trolley

Udgivet af Bogaerts Greenhouse Logistics

<https://www.bogaertsgl.com/index.php/qii-lift/qii-lift-h-series/qii-lift-h-350>

Besøgt d. 13. November 2020

[8] What Is the Current State of Labor in the Greenhouse Industry?

Udgivet af Greenhouse Grower

- [https://www.greenhousegrower.com/management/
what-is-the-current-state-of-labor-in-the-greenhouse-industry/](https://www.greenhousegrower.com/management/what-is-the-current-state-of-labor-in-the-greenhouse-industry/)
Besøgt d. 27. November 2020
- [9] Technical specifications
Udgivet af Comau
[https://www.comau.com/en/our-competences/robotics/
robot-team/racer-5-0-80](https://www.comau.com/en/our-competences/robotics/robot-team/racer-5-0-80)
Besøgt d. 30. November 2020
- [10] RevPi Connect base module
Udgivet af Kunbus
<https://revolution.kunbus.com/revpi-connect/>
Besøgt d. 1. December 2020
- [11] php-mqtt/client
Udgivet af Github bruger "Namoshek"
<https://github.com/php-mqtt/client>
Sidste besøg d. 1. December 2020
- [12] PDO vs. MySQLi: The Battle of PHP Database APIs
Udgivet af websitebeaver
<https://websitebeaver.com/php-pdo-vs-mysqli>
Besøgt d. 3. December 2020
- [13] Quality of Service 0,1 & 2 - MQTT Essentials: Part 6
Udgivet af HIVEMQ
<https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels>
Besøgt d. 3. December 2020
- [14] OPC Unified Architecture
Udgivet af Wikipedia
https://en.wikipedia.org/wiki/OPC_Unified_Architecture
Besøgt d. 14. December 2020
- [15] Risikovurdering
Udgivet af Wikipedia
<https://da.wikipedia.org/wiki/Risikovurdering>
Besøgt d. 17. December 2020
- [16] MariaDB vs MySQL: Key Performance Differences
Udgiver af Guru99
<https://www.guru99.com/mariadb-vs-mysql.html>
Sidst besøgt d. 30. December 2020