HØGSKOLEN I ÅLESUND

Aalesund University College

Bacheloroppgave

IE303612 Automatiseringsteknikk

Kleven Robot Controller

1815, 1843 og 1845

Totalt antall sider inkludert forsiden: 160 sider inkludert 7 vedlegg

Innlevert Ålesund,

HØGSKOLEN I ÅLESUND

Aalesund University College

Obligatorisk egenerklæring/gruppeerklæring

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

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§30 og 31.**	☒
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver**	☒
5.	**Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §30**	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☒

Publiseringsavtale

Studiepoeng: 20

Veileder: Ottar L. Osen og Hans Støle

Fullmakt til elektronisk publisering av oppgaven

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

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

Dato:

Forord

Prosjektet er en bacheloroppgave utført av 3 avgangstudenter ved automasjonsingeniør studiet ved høgskolen i Ålesund. Oppgaven er utført på vegne av og i samarbeid med Kleven Maritime AS som avslutting på 3årig ingeniørutdanning.

Takk til: Simon Sundal v/ Kleven Verft for tildeling av oppgaven, og godt samarbeid.
Ottar L. Osen v/ Høgskolen i Ålesund for god veiledning.
Hans Støle v/ Høgskolen i Ålesund for god veiledning.
Helge Tor Kristiansen v/ Høgskolen i Ålesund.

Kleven Robot Controller

Kandidatnummer: 1815, 1843 og 1845

May 28, 2015

Contents

SAMMENDRAG

Dette er en prosjektrapport som tar for seg utviklingen av en Android applikasjon som i en viss grad skal erstatte bruken av en robot-kontroller av typen Motoman DX100. Prosjektet er gjennomført i samarbeid med Kleven Maritime AS, og er en bacheloroppgave utført av tre automasjonsingeniør studenter ved Høgskolen i Ålesund.

Oppgaven ble utformet for å forhindre unødvendige stopp i produksjonen og for å kunne gjøre det enklere for operatører å utføre oppgaver ved hjelp av et nettbrett. For å få til dette ble det satt opp trådløs kommunikasjon mellom nettbrettet og Motoman robot, som igjen ble koblet opp mot en ekstern database for lagring av dokumentasjon og henting av tilgjengelige jobber. For å kunne manipulere databasen, ble det laget en "SQL-database Manager", som gjør det enkelt å utføre endringer i databasen.

I denne rapporten er det fremlagt metoder for hvordan de forskjellige problemstillingene ble løst, med flere mulige løsninger som ble vurdert med hensyn på funksjonalitet og effektivitet innad i systemet. Det ble til slutt konkludert at resultatet oppfyller de spesifiserte kravene fra oppdragsgiver, som også ble godt fornøyd med det endelige produktet.

TERMINOLOGI

BEGREPER

PHP - Står egentlig for Personal Home Page, men betyr nå Hypertext Preprocessor.
Extends/Arver - Klasse som bygger videre på en eksisterende klasse.
Implements - Implementerer en eksisterende klasse.
Handshake - Stegene som verifiserer tilkobling eller autorisering av at noen vil ha tilgang.
Protokollheader - Eit bestemt oppsett av data i begynnelsen av en blokk med data som sendes.
Socket - Endepunktet ved overføring av data over nettverkslaget.
Int - Integer verdi, heltall.
Real - Real verdi, kommatall.
Double - presisjonsinteger, kommatall.
String - Data type, inneholder opptil flere bokstaver eller en setning.
ArrayList - Liste med mulighet til å lagre alle mulige elementer i.
Loose connection - Klassene er lite knyttet til hverandre, som fører til at klasser kan byttes ut uten å medføre problemer.
Carriage return - En mekanisme i data tekst som betyr begynnelsen av en linje.
Line feed - En mekanisme i data tekst som betyr ny linje.
World Coordinates - Lineære bevegelser hvor roboten beveger flere ledd samtidig.
Joint Coordinates - Roboten beveges ledd for ledd.
Teachbox - Robot skap + pendant.
Pendant - DX100 robot kontroller.
Open source - fritt tilgjengelig kode eller lignende.
ENUM - En spesiell data type som gjør det mulig for en variabel å bli satt til konstant

Sanntid - blir løst med tråder, kode som kjøres tilsynelatende samtidig.
Little-endian - Forklarer i hvilken rekkefølge bytes blir tolket, i dette tilfellet lagres den minst signifikante byten i den laveste adressen.

SYMBOLER

`<CR><LF>` - Carriage return og Line feed

FORKORTELSER

ASA - Anders Simon Audun
ASA RC - ASA Robot controller
ASA DM - ASA Database manager
IDE - Integrated development environment
SQL - Structured query language
OOP - object orientated programming
HSEC - High Speed Ethernet Client
HTTP - Hypertext transfer protocol
SMTP - **URL** - Uniform resource locator
SDK – Software development kit
ADT - Android development kit
XML - Extensible Markup Language
UI - User interface
SOAP - Simple Object Access Protocol
API - Application programming interface
IO - Input/Output

1 INNLEDNING

Havet blir sett på som verdens største motorvei, og blir brukt til å transportere varer og mennesker over alt i verden. For å kunne bruke denne veien, trengs det båter. Det blir laget flere hundre båter i året på forskjellige verft over hele verden. Produksjonen av skip blir gjort i flerfoldige land, men produksjonen av avanserte offshore fartøy blir produsert i land som Norge. Langs den langstrekte kysten i Norge ligger det mange forskjellige verft. Et av de fremste verfta i Norge er Kleven Verft. Per i dag er det det største norskeide skipsbyggingskonsernet i landet[1]. Produksjonen av skrog blir gjort i lavkost land som Polen, men ved å automatisere produksjonen på verftet i Ulsteinvik, prøver Kleven å få en del av produksjonen tilbake til Norge. Med håndterings- og sveiseroboter har de en helautomatisert produksjonslinje for skrogbygging. De har også roboter for å sveise sammen seksjonane fra små deler til hele stålkonstruksjoner som til slutt blir en del av skroget. Disse robotene har prosjektgruppen tatt for seg, og utviklet et styringssystem i Android. Denne applikasjonen skal i mindre grad erstatte den nåværende pendanten.
Prosjektet er en bacheloroppgave utført av 3 avgangstudenter ved automasjonsingeniør studiet

ved høgskolen i Ålesund. Oppgaven er utført på vegne av og i samarbeid med Kleven Maritime AS som avslutting på 3årig ingeniørutdanning.

Bakgrunn for oppgave:
Kleven Maritime AS (heretter Kleven) bruker i dag automatiserte sveiseroboter i store deler av sin produksjon i Norge. Disse er av merket Motoman og leveres med en DX100 kontroller. Gjennom flittig bruk har det vist seg å være et stort problem at kabelen til kontrolleren til stadighet blir skadet. Dette fører til dyre utgifter og uønsket stans i produksjon. Ettersom stans i produksjon medfører høye kostnader i form av reperasjoner og nedetid, er dette et problem Kleven er meget interessert i å få løst. Applikasjonen har også som hensikt å gjøre det enklere og raskere for operatører å utføre sveisejobber. Dette har dannet grunnlaget for oppgaven som gruppen ønsker å løse gjennom dette prosjektet.
I samarbeid med Kleven har gruppen kommet frem til at en mulig løsning vil være å delvis erstatte Motoman DX100 med et nettbrett. Dette vil minske bruken av DX100 kontrolleren og dermed minske sannsynlighet for skade på kabel. I tillegg vil dette kunne forenkle opplæring av nye operatører, forenkle dokumentering og forhåpentligvis føre til økt produksjon.
Sveiserobotene som applikasjonen har som hensikt å styre, flyttes rundt på hele tiden. Dette medfører at miljøet, og sveisejobben endres fra gang til gang. De største fordelene med denne applikasjonen kommer først frem når en lager program på roboten (jobber) som kan tilpasses til det miljøet roboten er i og den oppgaven den skal utføre. Når slike jobber er laget kan applikasjonen brukes til å endre parameter som posisjoner, avstander, hastigheter, eller lignende på en enkel måte. Parameterne kan da brukes til å definere det nye miljøet den befinner seg i, slik at jobben kan utføres uten å måtte omprogrammere roboten. Slike type jobber kan da etter å ha blitt programmert på roboten, lett implementeres i applikasjonen ved hjelp av en database, der er det mulig å legge til jobber og knytte de opp i mot de parameterne jobben trenger for å kunne tilpasses forskjellige oppgaver.
Basert på dette er problemstillingen gruppen har tatt for seg definert på følgende vis:
Hoved problemstilling:

- Utvikling av en trådløs applikasjon som delvis kan erstatte og minske bruk av Motoman DX100 kontroller.

Videre har vi valgt å dele opp prosjektet i 4 mindre deler. Disse har vi valgt å definere på følgende vis:
Del problem:

- Utvikle en Android applikasjon med enkelt brukergrensesnitt.
- Sette opp trådløs kommunikasjon mellom nettbrett og Motoman robot.
- Sette opp kommunikasjon mot ekstern database for lagring av dokumentasjon og henting av tilgjengelige jobber.
- Utvikling av enkel "Backend-solution" for enkel behandling av den eksterne databasen.

I denne rapporten vil vi ta for oss:
Utvikling av Android applikasjonen i Java og oppsett for kommunikasjon med SQL-databaser. Her vil vi forklare hvordan vi satt opp kommunikasjon oppimot MySQL og MSSQL ved bruk av et PHP-skript. Dette innebærer sending og mottak av informasjon til og fra SQL-database via Java/Android, hensikten med PHP-skriptet, oppsett av database med tabeller samt problemer vi møtte på. Vi vil også ta for oss alternative metoder for kommunikasjon.

Vi vil også ta for oss oppsett for kommunikasjon med robot. Her vil vi se på endel forskjellige løsninger for kommunikasjon opp i mot Motoman, dette inkluderer:
• Ethernet server function
• MotoNIS
• High-Speed ethernet server function
• Modbus og PLS

Vi vil ta for oss fordeler og ulemper ved de forskjellige alternativene, samt implementering og oppsett av "High-Speed Ethernet Client". Det vil også bli forklart hvordan vi implementerte kommunikasjonen i applikasjonen og satte opp roboten for kommunikasjonen.

2 TEORETISK GRUNNLAG

2.1 Robot og Kommunikasjon

Robotene som blir brukt er av typen Motoman MH5 som er produsert av YASKAWA. YASKAWA er en av de ledende fabrikantene innen industrielle roboter. Lysbue sveising er kjerna i firmaet, og har ført industrien de siste 25 åra[2]. Det finnes flere kontrollere, men her blir det bare gått inn på en av de, DX100 robot kontrolleren. Det er en teachbox som har muligheter til å håndtere sammensatte oppgaver, blant annet å kontrollere opp til 8 roboter, samt I/O enheter og kommunikasjonsprotokoller.
Fordelene med å bruke denne typen kontroller er[3]:
• Robust PC arkitektur
• Raskere prosessering, høy ytelse
• 25% energi sparing

2.1.1 Kommunikasjons protokoller

I nettutgaven av "store norske leksikon" har Ragnar Johnsen beskrevet kommunikasjonsprotokoller på følgende måte:
"Kommunikasjonsprotokoll, fastlagt sett av regler for informasjonsutveksling mellom kommuniserende digitale enheter. I prinsippet skal en kommunikasjonsprotokoll sikre at enheter som «snakker sammen» benytter samme «språk» slik at de forstår hverandre. Det benyttes en form for kommunikasjonsprotokoll i nesten alle sammenhenger der to eller flere enheter utveksler digital informasjon, for eksempel mellom en PC og en skriver, mellom datamaskiner i et nettverk, mellom en mobiltelefon og telenettet og mellom en fjernkontroll og fjernsynsapparatet[4]."

2.1.2 Ethernet

Ethernet er den teknologien som blir brukt i lokale datanettverk. Det ble introdusert i 1980 og ble standardisert i 1983 som IEEE 802.3. Siden den gang har standarden utviklet seg fra 2.94 Mb/s til 100 Gb/s, og brukes blant annet på fiber, trådløs og tvunnet-par kabler. Data som blir sendt i lokale nettverk brytes ned i rammer. Hver ramme inneholder informasjon om hvor den skal og hvor den kommer fra. Denne teknologien er en del av ethernet standarden, og ligger på det fysiske- og datalinklaget i OSI modellen[13, 14].

2.1.3 UDP og TCP

"Transmission Control Protocol" (TCP) og "User Datagram Protocol" (UDP) er to protokoller som ligger på transportlaget i OSI-modellen. TCP gir en pålitelig, sortert og feilfri leveranse av datapakker fra ende til ende. Siden denne protokollen har håndtrykk, opphopingskontroll og feilhåndtering, er den relativt treg og uegnet for sanntidsapplikasjoner[15]. Der UDP er mer egnet til sanntid fordi den har minimalt med protokoll mekanismer, og har dermed ikke handshaking, sortering og lignende funksjoner. Dette gjør at protokollen er raskere og er godt egnet der rask overføring er viktigere enn å vente på en forsinket datapakke[16].

2.1.4 Klient-server kommunikasjon

I en klient-server arkitektur, er serveren sin jobb å levere en funksjon eller tjeneste til en eller flere klienter[5]. Klienter og servere utveksler informasjon i et "request-response" mønster: Klienten sender en forespørsel og serveren returnerer et svar. For at de skal forstå hverandre må de prate samme språk, og ha regler om hvordan kommunikasjonen skal foregå. Dette er definert av en kommunikasjonsprotokoll (2.1.1 Kommunikasjons protokoller).

2.1.5 Simple Object Access Protocol

"Simple Object Access Protocol" (SOAP) er en protokoll som står for utveksling av informasjon med en webtjeneste. Den bruker XML som meldingsformat, og HTTP eller SMTP for å sende informasjon[6].

2.1.6 PLS

PLS står for "Programmerbar Logisk Styring", dette er en liten datamaskin som blir brukt i alle typer industri for å automatisere prosesser[9]. Disse datamaskinene har en rekke digitale og analoge inn- og utganger som kan lett programmeres for ønsket oppførsel, og blir brukt til å styre elektromekanisk utstyr. Noen PLSer støtter forskjellige kommunikasjons protokoller, som gjør at de også kan brukes som et mellomledd mellom utstyr som ikke bruker samme protokoll. Det finnes mange typer PLSer, noen er små og har et begrenset antall IO og kommunikasjons muligheter, andre kan være modulære enheter som bygges opp etter hvilke behov brukeren har. Disse kan ha flere tusen inn- og utganger, og støtte de fleste feltbusser.

2.1.7 Modbus

Modbus er en åpen kommunikasjonsprotokoll orginalt utgitt av Modicon, den har blitt en av de mest populære kommunikasjonsmetodene mellom industrielle elektroniske komponenter og styringer[11]. Fordeler med modbus er at den er lett å ta i bruk, den er laget for industriell bruk og den er gratis. Protokollen kan brukes både over seriell kommunikasjon og over ethernet, og støtter både TCP og UDP. Modbus header[12]:

Name	Length (bytes)	Function
Transaction identifier	2	For synchronization between messages of server & client
Protocol identifier	2	Zero for Modbus/TCP
Length field	2	Number of remaining bytes in this frame
Unit identifier	1	Slave address (255 if not used)
Function code	1	Function codes as in other variants
Data bytes	n	Data as response or commands

Figure 1: Modbus header[12]

Siden modbus ble designet rundt 1970 for kommunikasjon med PLSer er det ganske begrenset hvilke datatyper den støtter. Den kan kun brukes til å lese og skrive bit for bit eller 16-bits register, men der finnes også implementasjoner av modbus som støtter floating point, 32-bit integer og 8-bit data.

2.1.8 Motonis

MotoNIS SDK er et "software development kit" for å bygge applikasjoner som kan kommunisere med robot-kontrollere[8]. Det gir rask og pålitelig kommunikasjon over alle typer ethernet-nettverk. MotoNIS SDK inneholder en webtjeneste, som heter CoreService, og MotoNIS Web som er en webserver som installeres på robotkontrolleren. Disse gir tilgang til å overvåke roboten, hente jobbliste, skrive/lese variabler og IO, motion control, og fil overføringer. Den inneholder også et bibliotek for å lage .NET klienter, kildekode for å lage en C klient, og en wsdl-fil som beskriver CoreService webtjenseten. Denne beskrivelsen av webtjenesten gjør at tredjeparts verktøy kan implementere denne tjenesten ved å bruke SOAP-protokollen(2.1.5 Simple Object Access Protocol).

2.1.9 Ethernet server

"Ethernet server function" er en kommunikasjonsprotokoll som brukes for datautveksling mellom DX100-robotkontroller og eksterne enheter som for eksempel pc eller nettbrett[10]. Den ligger opp på TCP-protokollen og kommuniserer gjennom port 80. Ved å implementere denne protokollen kan vi blant annet:

- Lese og nullstille alarmer.
- Sjekke status parameterne til roboten.

- Lese/skrive posisjonsvariabler.
- Lese/skrive IO, og primitive variabler(int,real,double).
- Hente Jobbliste.
- Velge og starte/stoppe jobber.
- Sende bevegelse kommandoer.

Alle kommandoer blir sendt som Strings og det utføres en handshake før kommandoen sendes. Så lenge socketen ikke lukkes kreves det ikke en ny handshake for hver kommando.

Kommunikasjonsflyt:

1. Klienten kobler til TCP port 80 på DX100.
2. Klienten sender en START-forespørsel.
3. DX100 sender et svar om den godtar forespørselen.
4. Klienten sender ønsket kommando.
5. DX100 svarer om den godtar kommandoen.
6. Klienten sender data tilhørende kommandoen hvis nødvendig.
7. DX100 sender svar til klienten.
8. Klienten lukker tilkoblingen.

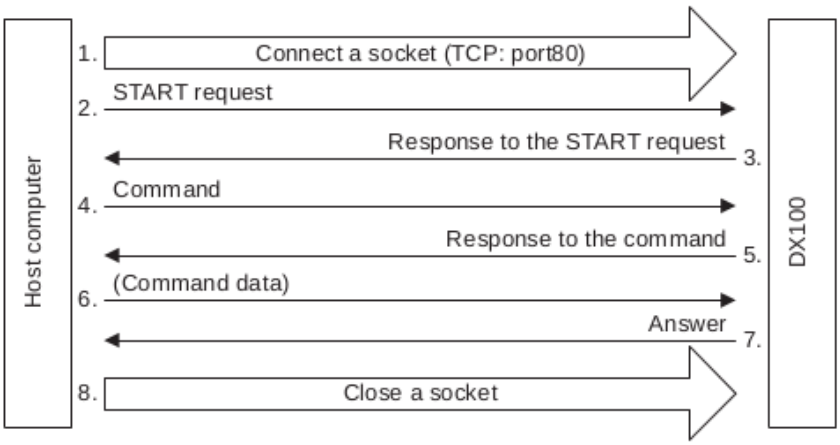


Figure 2: Kommunikasjon eksempel[10]

START-forespørsel: Klienten må sende en String for å si i fra at det kommer en kommando. Den er definert slik: `"CONNECT Robot\_access <CR><LF>"`. DX100 kontrolleren sender

da tilbake: "OK: DX Information Server(ver)<CR><LF>". Antall kommandoer som skal sendes kan defineres ved å legge til: "Keep-Alive:N." før <CR><LF>. Responsen vil da også inneholde "Keep-Alive:N" der N er antall kommandoer.

Kommando-forespørsel: Etter godtatt START-forespørsel kan kommando sendes til robot, den må starte med "HOSTCTRL_REQUEST X Y <CR><LF>" der X er kommando-navnet og Y er antall bytes data som følger med kommandoen (0 hvis kommando ikke inneholder mer data). Responsen fra DX100 blir da: "OK: X <CR><LF>", hvis Y ikke er 0 sendes kommando-data nå. Deretter svarer DX100 på kommandoen som ble sendt[10].

2.1.10 High-speed ethernet server

"High Speed Ethernet server function" er en kommunikasjonsprotokoll som brukes for datautveksling mellom DX100 robot-kontroller og eksterne enheter som for eksempel pc eller nettbrett[7]. Med denne funksjonen aktivert, er det mulig å kommunisere over to ganger raskere enn med "Ethernet server function". Dette fordi det ligger på UDP-protokollen(2.1.3 UDP og TCP), og at den ikke gjennomfører noe form for handshake, noe som gjør den veldig rask. Ved å implementere denne protokollen kan en blant annet:

- Lese og nullstille alarmer.
- Sjekke status-parameterene til roboten.
- Lese/skrive posisjonsvariabler.
- Lese/skrive IO, og primitive variabler(int, real, double).
- Hente jobbliste.
- Velge og starte/stoppe jobber.
- Sende bevegelse kommandoer.

Datapakkene protokollen sender er bygd opp av maks 512 bytes, der de 32 første er headeren og de resterende 480 er datadelen av pakken. Datadelen blir kun brukt av noen kommandoer der data utover det som er i headeren er nødvendig.

Item		Data size	Settings
Identifier		4byte	Fixed to "YERC"
Header part size		2byte	Size of header part (fixed to 0x20)
Data part size		2byte	Size of data part (variable)
Reserve 1		1byte	Fixed to "3"
Processing division		1byte	1: robot control 2: file control
ACK		1byte	Request: 0 Other than request: 1
Request ID		1byte	Identifying ID for command session (increment this ID every time the client side outputs a command. In reply to this, server side answers the received value.)
Block No.		4byte	Request: 0 Answer: add 0x8000_0000 to the last packet. Data transmission other than above: add 1 (max: 0x7FFF_FFFF)
Reserve 2		8byte	Fixed to "99999999"
Sub-header (request)	Command No.	2byte	Execute processing by this command. (conforms to "Class" of CIP communication protocol)
	Instance	2byte	Define SECTION to execute a command. (conforms to "Padding" of CIP communication protocol)
	Attribute	1byte	Define SUB SECTION for executing a command. Attribute: (conforms to "Attribute" of CIP communication protocol)
	Service (request)	1byte	Define data accessing method.
Sub-header (answer)	Service (answer)	1byte	Add 0s80 to service (request).
	Status	1byte	0x00: normal reply Other than 0x00: abnormal reply
	Added status size	1byte	Size of added status (0: not specified / 1: 1 WORD data / 2: 2 WORD data)
	Added status	2byte	Error code specified by added status size For details, refer to *chapter 5 "Added Status Code"*.
Padding		Variable	Reserve area

Figure 3: Oppbygging av forespørsel- og svarheader [7]

Kommunikasjonsflyt:

1. Klienten kobler til UDP port 10040 på DX100.
2. Klienten sender et UDP-datagram som samsvarer med den definerte headeren.
3. DX100 prosesserer kommandoen og sender en respons til klienten.
4. Klienten lukker tilkoblingen.

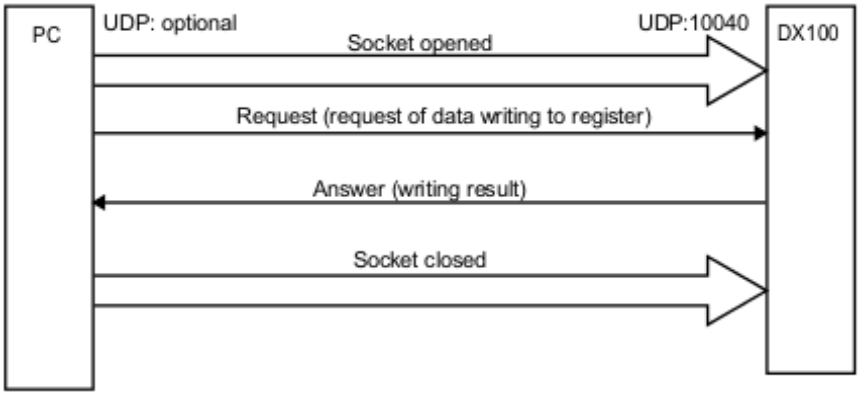


Figure 4: Kommunikasjon eksempel [7]

2.2 Java og Android

2.2.1 Java

Java er et objekt-orientert programmeringsspråk laget og oppdateres av Oracle. Java ble først demonstrert i 1995, av The Green Team.

I dag er Java brukt over alt på internett, det er også selve oppbyggingen bak mange applikasjoner og enheter som blir brukt hver dag. Programmeringsspråket blir brukt i alt fra mobiltelefoner til spill og navigasjonssystem[17].

2.2.2 Android

Android er et mobilt operativsystem som blir kjørt på de fleste mobile plattformer verden over. Det er en modifisert versjon av Linux og er utviklet av Google. Operativsystemet bruker touch kommandoer som er i likhet med reelle bevegelser, som swipe funksjoner, trykk funksjoner og andre naturlige bevegelser[18]. Google har offentliggjort kildekoden under en åpen kilde lisens, det er derfor gratis for alle å laste ned utviklingsprogram og utvikle egne applikasjoner etter eget behov.
Det finst flere mulige utviklingsplattformer for Android, det nyeste er Android Studio. Det er en IDE som er laget for å utvikle kun Android applikasjoner. Men det finst også andre plattformer som Eclipse med Android SDK, IntelliJ og AIDE-IDE.

2.2.3 Adapter

Et brukergrensesnitt i Android består av flere komponenter, såkalte Views(2.2.9 View). Et AdapterView er en komponent som bruker et adapter for å behandle bakgrunnsdataene[19]. Adapteret virker som et mellomledd mellom hva bruker ser og det som ligger i bakgrunnen. Det kan for eksempel være å hente ut verdier fra en liste eller finne ut hvor bruker trykket. På samme måte som et adapter brukes for å hente ut denne informasjonen, blir et adapter også brukt for å legge til data i bakgrunnsdataene[20].

Et adapter objekt oppfører seg som en bru mellom et "View"(Se 2.2.9 View) som for eksempel en knapp, en liste eller et tekstfelt, og dataen som ligger under dette synspunktet. Et adapter kan bli brukt for å legge til verdier i en liste eller for å vise flere elementer på samme linje i en liste[20].

2.2.4 Android Manifest

Manifest filen beskriver viktig informasjon om applikasjonen, og må ligge med i alle applikasjoner. Denne filen blir automatisk opprettet av Eclipse utifra hva som blir spesifisert når prosjektet opprettes.
Oppgaven til manifestet er å beskrive komponenter i applikasjonen, som for eksempel aktiviteter. Deklarering av tillatelser skjer under Android manifestet. Tillatelser må deklareres for å få tilgang til beskyttet deler av APIen. Deklarering av det laveste API versjonen er også gjort i Android manifestet[21].

2.2.5 Utviklingsverktøy

- **Eclipse m/Android SDK** er en gratis IDE som i hovedsak gjør det mulig å programmere i Java. Eclipse prosjektet ble opprinnelig laget av IBM i 2001. Det ble i 2004 laget et selvstendig selskap som skulle styre Eclipse samfunnet[22].
  Ved å legge til en Android SDK blir det mulig å programmere applikasjoner for Android. Det som blir lagt til er en måte å designe et grensesnitt i en .xml fil som gir en forhåndsvisning i hvordan det vil bli seende ut på nettbrettet eller telefonen. Det vil deretter være mulig å koble en Android enhet i pcen og få opp denne applikasjonen på nettbrettet, eller starte en emulator på pcen som viser det samme som et nettbrett.

  - Android SDK: Ved bruk av en utviklingsplattform for Java, må en laste ned en ekstern programvare for Android; Android SDK. Dette programmet laster ned de bibliotekene som er nødvendig for å programmere en Android applikasjon.

- **NetBeans** er også en gratis IDE som støttes av flere programmeringsspråk som Java, PHP, C/C++, osv. NetBeans ble åpnet for åpen kildekode av Sun Microsystems i 2000. Den er gratis for kommersiell og ikke-kommersiell bruk. Android SDK er ikke offisielt støttet av NetBeans og fører dermed til ukjente problemer[23].

- **Android Studio** er den offisielle IDE'en for android applikasjonsutvikling. Android Studio tilbyr:

  - Fleksibelt oppbyggingssystem
  - Kode maler for å hjelpe å bygge opp applikasjoner
  - Layout klippebord med støtte for "drag and drop"

  Android Studio ble publisert i 2013 av Google. IDE'en er gratis å bruke, og er bare brukt for utvikling av Android applikasjoner. Den skal i hovedsak erstatte Eclipse ADT som

Google's primær IDE for Android applikasjon utvikling[24, 25].

2.2.6 Fragmenter og Aktiviteter

Aktivitetsklassen er en viktig del av applikasjonens livssykluse. Den tar seg av å lage et vindu, hvor en kan plassere brukergrensesnittet sitt inn i. Aktiviteter er styrt av en ≪activity stack≫ som gjør at ved oppstart av en ny aktivitet vil den nye aktiviteten bli plassert øverst i stacken og bli den aktiviteten som blir kjørt. Den forrige aktiviteten vil alltid ende opp under den nye aktiviteten, og vil ikke bli synlig før den nye aktiviteten avsluttes[26].
En aktivitet har 4 forskjellige tilstander[26]:

- Viss aktiviteten vises på skjermen er den i ≪Running≫
- Om den har mistet fokuset, men fortsatt er synlig er den i ≪Paused≫
- Om aktiviteten er tatt over av en annen aktivitet, er den satt i ≪Stopped≫
- Viss en aktivitet er pauset eller stoppet, kan systemet droppe aktiviteten fra minnet ved å gjøre den ferdig eller avslutte prosessen.

Et fragment representerer oppførselen av grensesnittet i en aktivitet. En kan se på et fragment som en modular seksjon av en aktivitet som har sin egen livssykluse, og som en kan legges til eller fjernes viss aktiviteten kjører. Et fragment må alltid være innesluttet i en aktivitet. En kan manipulere hvert fragment, som å legge de til eller fjerne dem. Når en utfører en slik transaksjon, kan en legge fragmentet i en ≪back-stack≫ som er styrt av aktiviteten. ≪Back-stacken≫ gjør det mulig for brukeren å gå tilbake til forrige fragment, ved å trykke på tilbakeknappen[27].

Et fragment har sin egen livssykluse hvor metoder blir kjørt etter hvilket stadie det er i. For eksempel blir `Oncreate()` kjørt når aktiviteten starter fragmentet, og `onCreateView()` blir kjørt når brukergrensesnittet til fragmentet skal bli satt opp[28]. Under vises hele livssyklusen til et fragment, med de metodene som blir kjørt.

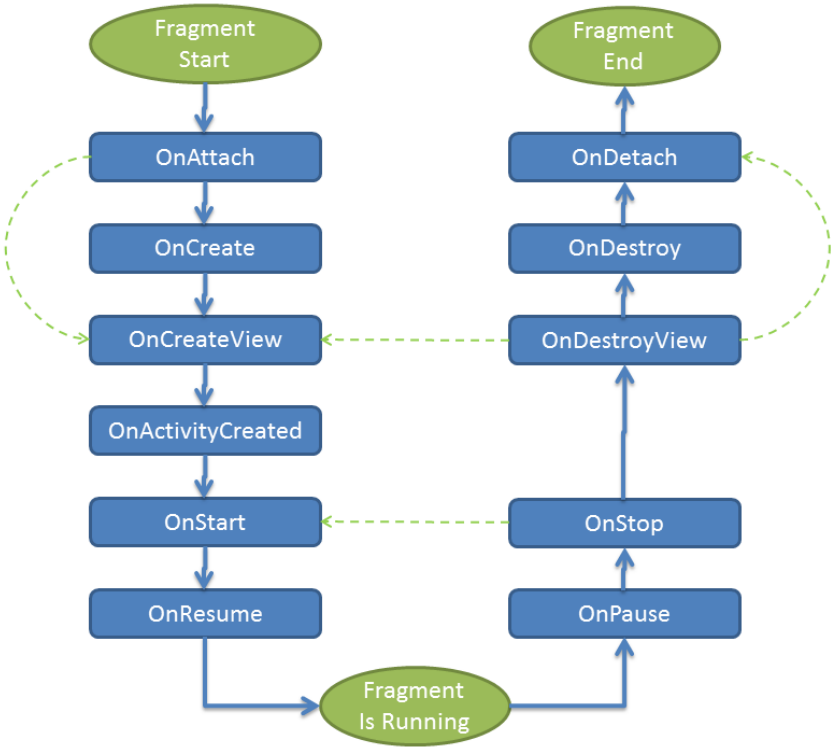


Figure 5: Fragment lifecycle [28]

2.2.7 Singleton

En singleton er en måte å utforme en klasse på. Ved å bygge opp klassen som en singleton, sørger man for at det bare kan bli laget et objekt av denne klassen. Dette objektet blir gjort tilgjengelig for alle klasser i applikasjonen ved å bruke en statisk metode som returnere dette objektet[29].

2.2.8 Bundle

En bundle holder på referanser til verdier ved å knytte disse til nøkler. Hver verdi blir knyttet til en nøkkel, og verdiene blir hentet ut ved å bruke den riktige "nøkkelen".

Fra Android Developers:
A mapping from String values to various Parcelable types[30].

På samme side får man også listet opp de forskjellige metodene en bundle tilbyr, og dermed hvilke verdier som kan bli lagret i en bundle.

2.2.9 View

Et "View" er en klasse i Android som beskriver en komponent i et brukergrensesnitt. Dette kan for eksempel være en knapp, en liste, et helt layout eller et tekstfelt. På nettsiden til Android.Developer er View beskrevet på følgende måte:

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.)[31].

2.2.10 Tråder

Tråder brukes for å kunne kjøre flere deler av et program tilsynelatende samtidig[32]. I Java kan en tråd lages ved å arve fra Thread klassen og overskrive run()-metoden.

Alle Android applikasjoner består av minst en tråd. Denne tråden kalles gjerne for Hovedtråd / UI-tråden og har blant annet ansvaret for å behandle brukergrensesnittene. Til forskjell fra brukerskapte tråder, er det ikke tilatt å stoppe hovedtråden[33].

Under er det listet opp de forskjellige tillstandene en tråd kan ha i Java. Listen er hentet fra Java sin API [34]:

- NEW - A thread that has not yet started is in this state.
- RUNNABLE - A thread executing in the Java virtual machine is in this state.
- BLOCKED - A thread that is blocked waiting for a monitor lock is in this state.
- WAITING - A thread that is waiting indefinitely for another thread to perform a particular action is in this state.
- TIMED_WAITING - A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
- TERMINATED - A thread that has exited is in this state.

2.2.11 AsyncTask

AsyncTask er en klasse for Android som tillater utførelse av bakgrunnsoperasjoner. Til forskjell fra tråder, er AsyncTask ofte brukt til korte bakgrunnsoppgaver som utføres asynkront. Alle AsyncTask har 3 felles typer, “Params”, “Progress” og “Result”, de håndterer inngangsparameterne, fremgangen og resultatet av den utførte operasjonen. Klassen blir kjørt i 4 steg, “onPreExecute”, “doInBackground”, “onProgressUpdate” og “onPostExecute”, og startes ved bruk av execute()-metoden[35].

2.2.12 Timer og Timertask

En timer brukes for å utføre oppgaver til en angitt tid eller gjentatte ganger. Timer klassen benytter en tråd som utfører oppgavene som angitt av bruker. Timer inneholder flere forskjellige valg bruker kan velge mellom, for eksempel kan en oppgave bli utført hvert minutt, etter en viss tid har gått eller på et bestemt klokkeslett eller dato. Siden timer kun har en tråd vil oppgavene bli utført sekvensielt, noe som kan føre til potensielle forsinkelser i utførelsen[36].
Oppgavene som skal kjøres må være en instanse av TimerTask[37].

2.2.13 SharedPreferences

Sharedpreferences er en metode i Android for å lagre informasjon i data knyttet til applikasjonen. Denne informasjonen blir lagret i minne på enheten og vil bli værende så lenge applikasjonen ikke avinstalleres eller at minnet slettes av bruker. Sharedpreferences bruker nøkkel-verdi par, det vil si at når man legger til verdien, knytter man denne til en nøkkel, som igjen brukes for å hente frem den lagret verdien. Nøkkel-verdi paret blir lagret i den valgte ≪preference filen≫ som angir når man oppretter et sharedpreferences objekt. Her har man også mulighet til å spesifisere hvem som skal kunne hente ut den lagrede informasjon. Normalt vil dette være "MODE_PRIVATE", som betyr at kun applikasjon som opprettet preference filen kan finne den[38]

2.3 Database

En database blir brukt til å lagre informasjon på en enkel måte, hvor flere enheter kan koble seg på databasen og få tak i informasjonen som blir lagret. En database består av en til flere tabeller med rader og kolonner. I tabellene kan man enkelt legge til og hente ut informasjon ved å benytte database applikasjoner. Databaser kan enten være lokalt på en enhet, eller ligge på en server for å gi tilgang til informasjonen via Internett[39].

2.3.1 Raspberry Pi

En Raspberry Pi er en billig datamaskin som egentlig ble laget for barn i 2006. Den er laget for at folk skal lære seg å programmere. På åtti-tallet måtte folk lære seg å kode for å kunne ta i bruk pcen, noe som ikke trengs for å bruke dagens datamaskiner, derfor ble Raspberry Pi til. Det finst 3 versjoner av Raspberry Pi;Raspberry Pi 1 Model A+,Raspberry Pi 1 Model B+ og Raspberry Pi 2 Model B+. Forskjellen ligger i prosessoren og antall ekstra utstyr som USB porter, audio jack, osv[40].

2.3.2 SQL

SQL ble utviklet av IBM på 1970-tallet. Det ble laget for å manipulere og gjenopprette data fra IBMs originale database administrasjonssystem. Det er et programmeringsspråk for databaser, som lar brukeren manipulere data inn i en database. Det SQL gjør er å sende et "query", en forespørsel, som håndterer informasjon i databasen. Dette gjør det for eksempel mulig å hente data, legge til data, slette data og oppdatere data i databasen. Databaser blir brukt til å lagre info på en server, hvor flere klienter har mulighet å hente ut det som blir lagret der[41].

2.3.3 Microsoft SQL Server

Er Microsofts database løsning og finnes i mange forskjellige versjoner. Ved å besøke Microsofts offisielle nettside, kan en finne den serveren som passer best for systemet en lager[42]. Det positive med å bruke Microsofts SQL server er at serveren virker bra i Windows, men det er også det eneste operativsystemet det fungerer i[43].

2.3.4 MySQL

Er et databaseadministrasjonssystem som brukes for å opprette og administrere databaser. Det påstås å være verdens mest populære «open source» database og blir blant annet brukt av flere høy profilerte nettsteder som facebook og youtube[44] .

2.3.5 SQLite

SQLite er et bibliotek som implementerer en fullstendig, serverløs, transaksjonell SQL database motor. Ulikt andre SQL databaser, har ikke SQLite en seperat server prosess. Den leser og skriver direkte til disken[45].

2.3.6 JDBC

JDBC står for «Java Database Connectivity» og er en API utgitt av Oracle for å administrere databaser rett fra Java. API'en er standard for selvstendige tilkoblinger mellom Java språket og flere forskjellige databaser. Det er mulig å bruke JDBC også fra Android, men de har ikke inkludert noen JDBC drivere, disse må brukeren anskaffe selv[46].

2.3.7 PHP

Muligheten for at program skal kunne sende og motta objekt over verdensveven, har ført til utviklingen av forskjellige skriptspråk som Python, Ruby og PHP[47]. PHP er et universalt skriptingspråk som er spesielt egnet for webutvikling. Det å sende noe over verdensveven ved hjelp av et PHP-skript er en rask, fleksibel og pragmatisk løsning, som blir brukt i alt fra blogger til de mest populære nettsidene i verden[48].

3 MATERIALER OG METODE

3.1 MATERIALER

3.1.1 Samsung Galaxy Tab S

Samsung Galaxy Tab S nettbrettet er toppmodellen til Samsung og har en størrelse på 10.5", og kjører Android versjon 4.4.2[53]. Nettbrettet er enheten Android applikasjonen blir kjørt på. Enheten er i kontakt med både SQL databasen og Motoman roboten via WiFi. Nettbrettet skal delvis erstatte DX100 kontrolleren.

3.1.2 Motoman robot

Robotene som ble brukt i prosjektet, tilhører Kleven Verft og er robotene hvor applikasjonen vil bli brukt ved ferdigstilling av prosjektet. Gruppen fikk tildelt en Yaskawa Motoman MH5 robot hvor de kunne teste applikasjonen og kommunikasjonen når de ville. Roboten er koblet til en kontroller av typen DX100.

3.1.3 DX100 kontroller

Yaskawa Motoman DX100 er robotstyringen som brukes for å programmere sveiserobotene. Den har mulighet til å kontrollere opp til 8 roboter, og programmeres gjennom en "programmeringspendant/teach box". Kontrolleren støtter Motoman sine kommunikasjonsprotokoller, i tillegg har den utvidingsmuligheter som gjør at den støtter feltbusser som for eksempel: Ethernet/IP, Profibus-DP, DeviceNet og CC-link[54].

3.1.4 Router

For å kunne koble robot til nettbrett og databasen til nettbrett, ble det brukt en lokal router med trådløst nettverk. Robot ble koblet til med ethernet kabel, mens databasen og nettbrettet ble koblet til det trådløse nettverket.

3.1.5 Raspberry Pi

En Raspberry Pi ble brukt som lokal database liggende i robotskap. Dette for å unngå problemer ved oppkobling på nettverket hos Kleven, men også for å kunne bruke databasen andre plasser enn bare på Kleven.

3.2 Bibliotek

I applikasjonen ble det brukt biblioteker som gjør det mulig å utvide funksjonaliteten til programmet utover det som går med kun standard Java kode. For å lage Android applikasjoner må Java inneholde bibliotekene til Android API som er et sett med bibliotek brukt for å lage slike applikasjoner. I dette kapittelet beskrives de bibliotekene som ble brukt for å legge til ekstra funksjonalitet i applikasjonen.

3.2.1 Joystickview

Joystickview er et bibliotek laget av Zerokol fra 2014[56]. Den er laget spesielt for Android applikasjoner som simulerer en joystick for interaktive applikasjoner. Den viser også verdier for vinkel, kraft og retning, etter hvor en plasserer fingeren på joysticken. Denne funksjonen ble ikke implementert i den nye applikasjonen, men ble brukt i den gamle versjonen. Men mulighetene for å implementere denne i den nye versjonen ligger der fortsatt.

3.2.2 org.apache.commons

Dette biblioteket har funksjoner for behandlig av array. I dette prosjektet brukes det til å slå sammen flere array til et array[65].

3.2.3 Jamod

Dette biblioteket er en implementasjon av modbus(2.1.7 Modbus) i Java(2.2.1 Java), og kan brukes til å kommunisere med alle modbus enheter[66].

3.2.4 json-simple-1.1.1

Dette biblioteket gjør det mulig å konvertere til og fra JSON format[64].

3.3 ASA Robot controller

3.3.1 Popup-vinduer

AlertDialog.Builder er en klasse i Android som brukes for å produsere popup-vinduer (dialoger)[57]. Et objekt av AlertDialog.Builder klassen blir først opprettet og brukes deretter til å bygge opp en AlertDialog. Dette inkluderer blant annet å sette en tittel på vinduet og knytte det til en spesifikk design-fil(3.3.10.3 Android XML files). Når man er ferdig å bygge opp AlertDialogen ved hjelp av builder-klassen, brukes AlertDialog.Builder.Create() for å lagre objektet. I eksempelet under vises et utsnitt fra hvordan AlertDialogBuilder ble brukt for å lage LOGG-vinduet i applikasjonen.

```
logBuilder = new AlertDialog.Builder(activity);
logBuilder.setTitle("Communication logger");
LayoutInflater inflater = activity.getLayoutInflater();
View logview = inflater.inflate(R.layout.comlog, null);
logBuilder.setView(logview);
logDialog = logBuilder.create();
```

Her brukes LayoutInflater for å hente ut XML-filen som beskriver designet av vinduet.

Når dialogen er klar, benyttes `logDialog.show()` for å gjøre den synlig. Når man ikke lenger ønsker å vise dialogen, kan man for eksempel bruke `logDialog.hide()`. Dette skjuler vinduet for brukeren men det blir ikke lukket. Ønsker man å lukke vinduet kan man i stedet bruke `logDialog.dismiss()`

3.3.2 Singleton

På nettet finnes det mange løsninger for å sette opp en Singleton. Under vises måten det ble utført på i applikasjonen.

I klasser som skal være singleton, lages først et INSTANCE felt. Denne er deklarert som ”Private” og ”Static”, og holder en referanse til klasse objektet. Fordi det ikke skal kunne opprettes flere

objekter av denne klassen, settes konstruktøren til "privat". For å la andre klasser få tilgang til denne klasses brukes en statisk metode som kalles "getInstance". "Statisk" metode brukes for å gjøre metoden tilgjengelig uten å først måtte lage et objekt av klassen. "getInstance" returnerer INSTANCE-variabelen som inneholder en referanse til dette klasse objektet. På denne måten får alle klasser tilgang til det samme objektet og variablene og metodene som har blitt laget her.

3.3.3 Statiske variabler

Det er ofte nødvendig å dele variabler mellom flere klasser. En metode er å lage en Singleton klasse som nevnt over. En annen løsning er å bruke statiske variabler. I singleton-metoden sørger man for at alle klasser benytter samme klasse objekt. Med statiske felt sørger man for at alle objekt av en klasse inneholder de samme verdiene. Dette blir gjort ved å la alle felt være statiske og tilgjengenlige via set- og get metoder.

Når en klasse skal lese eller skrive til disse variablene, oppretter den et objekt av klassen og benytter seg av set- og get-metodene. Alle andre klasser som også skal bruke disse variablene, oppretter sitt eget objekt av klassen. Når verdiene blir oppdatert i det ene objektet blir de automatisk oppdatert i de andre objektene som en følge av statisk-deklarasjon.

3.3.4 Timer og TimerTask

En timer brukes for å utføre oppgaver til spesifiserte tider. Disse oppgavene er kalt TimerTasks.

Et objekt av timer klassen blir først laget. Denne har ansvaret for hvilken oppgave som skal kjøres, og når den skal kjøres. Timer klassen inneholder blant annet scheduleAtFixedRate og schedule. Disse tar igjen inn forskjellige parameter som gjøre det mulig å utføre oppgaven på en gitt tid eller dato, etter en gitt stund eller etter hvert x sekund/minutt/time.

TimerTask er tråden som skal kjøres og koden som skal bli utført blir lagt i run()-metoden. I en vanlig tråd blir run()-metoden kjørt ved å bruke ".start()", mens en timertask blir startet ved hjelp av en timer. run()-metoden kan bli kjørt alt fra 1 til uendelig antall ganger, alt etter hvordan man setter opp timeren.
For å stoppe timeren brukes ".cancel()" som avslutter alle oppgavene som blir utført.

3.3.5 RunOnUiThread

I applikasjonen blir det brukt flere forskjellige tråder. Når en tråd har laget et brukergrensesnitt (UI), har denne rettighet på dette grensesnittet. Dette gjør at andre tråder ikke kan oppdatere tekstfelt og knapper i dette grensesnittet. Hvis det allikevel er nødvendig å oppdatere et brukergrensesnitt fra en annen tråd, kan man benytte seg av runOnUiThread().

I den tråden man ønsker å oppdatere fra, lager man et objekt av Runnable-klassen. I run()-metoden til denne klassen legger man koden som gjør endringene på brukergrensesnittet. Ved så

å kalle på Activity klassen og bruke runOnUiThread(Runnable objekt) blir denne koden utført av tråden som har eierskap til dette brukergrensesnittet.

3.3.6 ListView med flere elementer på hver rad

"ListView" er et layout som lister opp innholdet fra topp til bunn, og kan brukes for å vise lister på en enkel måte. For å legge til verdier og håndtere trykk i listen, kan man bruke mange forskjellige "Adapter" som er tilgjengelig i Android(2.2.3 Adapter). For å klare å vise flere elementer på hver linje, ble det laget et eget adapter basert på baseAdapter-klassen. Koden er laget utifra en veiledning skrevet av Nithya Vasudevan i 2012 og kun mindre endringer er gjort for å tilpasse applikasjonen[58].

I vårt tilfelle får Adapteret inn en Arrayliste bestående av mindre Arraylister i konstruktøren.
`public TodayAdapter(Activity activity, ArrayList<ArrayList<String>> list)`
De mindre listene inneholder hver sin jobb med tilhørende parameter, og skal vises på samme rad. Hver rad i listen kan kun holde på en View (2.2.9 View), og er derfor nødvendig å definere et eget View som kan bestå av flere mindre Views. I applikasjonen lager vi en egen XML-Layout fil som består av et view(layout) med 6 tekstfelt(textview) etter hverandre. XML filen gjør det mulig å spesifisere størrelser, farger og hvilke komponenter vi ønsker i listen.

Figure 6: View bestående av flere views

I adapter klassen brukes en ViewHolder klasse. Dette er klassen XML-filen blir knyttet opp mot, og gjør det mulig å legge til og hente ut data fra feltene.

For hvert lille array i listen blir "getView" metoden arvet fra Baseadapter kjørt. Denne metoden knytter først komponentene i XML-filen til feltene i ViewHolder klassen. Deretter setter den feltene i Holder-klassen til å være lik verdiene i arrayet, før den returnerer View objektet som blir lagt til i listen.

Når brukeren holder inne en rad i listen, brukes en `onItemClickListner()` som henter ut verdien fra listen.

3.3.7 Spesifisering av enhets-versjoner

Ved å spesifisere laveste kompatible enhets-versjon, gjør man det umulig å begrense hvilke enheter som kan kjøre applikasjonen. Enheter som ikke kjører den valgte versjonen eller nyere, vil få informasjon om at applikasjonen ikke er støttet.

Valg av laveste kompatible versjon har mye å si for utformingen av applikasjonen, da funksjoner som ikke er støttet av denne, ikke vil kunne brukes[59].
I manifestet spesifiseres laveste kompatible enhets-versjon under "Uses-sdk" ved å bruke:

android:minSdkVersion="17"

I eksempelet over er 17 brukt som versjon nummer, tilsvarende Android v4.2 (JELLY_BEAN_MR1).
I tillegg kan man velge å spesifisere en mål-versjon (targetVersion) som sier hvilken versjon applikasjonen er testet mot, og høyeste tillatte versjon. Dette blir gjort ved å legge til:

android:targetSdkVersion="verdi"
android:maxSdkVersion="verdi"

3.3.8 Tillatelser

I Android må det i noen tilfeller spesifiserer egne tillatelser i manifestet. Dette inkluderer blant annet for å lagre og lese fra minnet, og sjekke tilkobling til internett.

I eksempelet under vises hvordan man kan legge til tillatelser i manifestet for å lese og skrive fra minnet.

```
<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>
<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>
```

3.3.9 Skjerm orientering

I Android er det mulig å spesifisere orienteringen til skjermen. Enkelt forklart, om applikasjonen skal vises liggende (landskap) eller stående (portrett). Hvis ikke noe er spesifisert vil applikasjonen kunne brukes i begge moduser.

For å spesifisere orienteringen brukes:

```
android:screenOrientation="landscape"
```

Denne legges under `<Activity>` og vil i eksempelet over, sørge for at applikasjonen kun blir kjørt i landskaps-modus. Alternativt kan man bruke "portrait" for portrett-modus

3.3.10 Grafisk brukergrensesnitt

3.3.10.1 Planlegging

Når man skal lage en applikasjon, er det viktig å ha en plan for hvordan designet skal se ut, men også funksjonalitet bør taes i betrakting. Det første som ble gjort var å lage flere forskjellige

modeller av designet i et web-basert modelleringsprogram kalt "Draw.io"[55]. Dette programmet gir brukeren mulighet til å lage grove modeller av hvordan designet vil se ut. Ved hjelp av dette verktøyet var det mulig å se for seg hvordan designet kunne utformes, og hva som ikke ville virke.

3.3.10.2 Grafisk programmering

Ved å bruke en ADT som følger med i Android SDK'en er det mulig å lage et visuelt bilde av applikasjonen. Når en lager en ny side må en først velge det ønskede nettbrettet/telefonen som skal bli brukt. Dette for å få plassert alt på riktig plass om en ønsker en statisk applikasjon. Ved å bruke forskjellige metoder i oppbyggingen er det mulig å lage en dynamisk applikasjon som tilpasser seg enheten.
Før en begynner å fylle inn med knapper og de ønskede innretningene, må en først velge et layout. For å få en dynamisk applikasjon ble det valgt at de fleste sidene skulle være av typen lineært layout hvor innretningene som ble lagt til ble tilpasset etter hvilke enhet som er tilkoblet. Men desse layoutene ble valgt etter behovet ved plassering av innretning. Under oppbygging av et fragment er det mulig å bruke "drag & drop" funksjonen hvor en plasserer de ønskede knappene og tekstfeltene sånn en vil. Der er alltid begrensninger etter hvilke layout en bruker, men ved å bruke "relative layout" eller "grid layout" vil det være lett å plassere innretningene der en vil på skjermen.
Når en har plassert de ønskede innretningene er det lurt å endre id navnene slik at det er lett å finne frem til de riktige innretningene i den tilhørende aktiviteten som skal utføres på dette fragmentet. Det er også mulig å endre farge, størrelser, tekst og alt en måtte øsnke å endre under "properties" feltet.

3.3.10.3 Android XML files

Oppbyggingen av det visuelle grensesnittet blir lagret i en XML-fil. Som forklart i 3.3.10.2-Grafisk programmering, er det mulig å bygge opp applikasjonen visuelt. Det vil da automatisk bli generert en XML-fil. XML-filen beskriver alle komponenter i et brukergrensesnitt i egne blokker. Alle kommandoer og innretninger inneholder en start-tag `<`,og en slutt-tag `>`.Det som er inn i taggen er innholdet av innretningen. Eksempel på dette vises i figuren under, hvor en har definert en knapp.

```
14      <Button
15          android:id="@+id/tab1_btnTodays"
16          android:layout_width="wrap_content"
17          android:layout_height="wrap_content"
18          android:text="Todays Tasks" />
19
```

Figure 7: Eksempel på kode i XML-fil

Som vist i figuren over er det blitt lagt til en knapp i grensesnittet. Den har fått et eget id navn, og teksten som står på knappen er "Todays Tasks". Størrelsen på knappen er satt til å automatisk tilpasse seg etter behov (lang text `-->` lang knapp osv.). For hver endring en gjør i "Properties" i windowbuilderen blir disse lagt til i denne boksen. I eksempelet er ikke tekst

størrelse eller farge definert, dette vil da føre til at de blir satt til sine standard verdier (svart, 20px).

3.3.11 Fragment håndtering

3.3.11.1 Bytte mellom fragmenter

I applikasjonen benyttes det flere fragmenter med tilhørende brukergrensesnitt(2.2.6 Fragmenter og aktiviteter). For å kunne bytte mellom disse ble det laget en metode kalt
`public void changeFrag(String FragName, Bundle arguments)`
FragName er navnet på fragmentet som skal vises og arguments er pakken (bundelen) med verdier som skal følge med.

I metoden lages først et objekt av FragmentManager og FragmentTransaction.

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction ft = fragmentManager.beginTransaction();
```

Via FragmentTransaction får man mulighet til å blant annet legge til, fjerne, bytte ut eller skjule fragmenter. I metoden blir det deretter sjekket hvilket fragment som skal bli vist. Dette gjøres med å sammenligne navnet på eksisterende fragmenter med FragName som ble benyttet i konstruktøren.
Hvis FragName stemmer overens med et eksisterende fragment, blir FragmentTransaction brukt for å erstatte nåværende fragment med det valgte.

```
else if (FragName.equalsIgnoreCase("FragmentJOGG")) {
FragmentJOGG.setMyArguments(arguments);
ft.replace(R.id.container, FragmentJOGG, "jogg");
ft.addToBackStack("jogg");
```

I eksempelet over blir det byttet til JOGG-fragmentet. I tillegg brukes `setMyArguments()`(3.3.12.2 MyArguments) for å sende med informasjonen og `addToBackStack()`(3.3.11.2 Pop to backstack) for å loggføre byttet. Selve byttet blir utført i slutten av metoden ved å bruke `.commit()`.

3.3.11.2 Pop to backstack

PopToBackstack() er en metode som henter tilbake gamle transaksjoner og reverserer dem. Metoden blir kjørt når bruker trykker på tilbake knappen på enheten, men kan også bli lagt til andre steder i koden.

I 3.3.11.1-Bytte mellom fragmenter, blir det vist et eksempel hvor man bytter fra et fragment til et annet. Her blir FragmentTransaction brukt til å erstatte fragmentet, samtidig som ft.addToBackStack("jogg") blir brukt for å lagre transaksjonen. Denne metoden legger transaksjonen til en liste som inneholder alle transaksjoner som har blitt utført.
Ved å bruke popToBackstack() blir denne listen hentet opp og siste transaksjon blir reversert. I eksempelet har det blitt brukt en "tag" kalt "jogg". Dette gir et navn til transaksjonen, så

man kan identifisere hver transaksjon. Det fører til at man for eksempel kan fjerne det angitte transaksjonen fra listen.

Når bruker er kommet gjennom programmet, er det ikke lenger ønsket at bruker skal kunne gå tilbake. Derfor blir hele listen over tidligere transaksjoner fjernet når brukeren kommer tilbake til første siden.
Dette blir gjort ved å bruke en loop som sletter hver oppføring i listen helt til den er tømt.

```
   while(fragmentManager.getBackStackEntryCount()>0){
fragmentManager.popBackStackImmediate();}
```

3.3.12 Behandling av data

3.3.12.1 SharedPreferences

SharedPreferences brukes for å lagre verdier i en preference-fil. Verdier som lagres her vil bli husket, selv om applikasjonen lukkes.

I applikasjonen blir først preference-filen hentet ved å bruke

```
SharedPreferences memory = this.getSharedPreferences("memory",MODE\_PRIVATE);
```

Denne oppretter et objekt av SharedPreference klassen og henter opp preferencefilen "memory". Hvis filen ikke finnes eller er laget av en annen applikasjon (privat mode), blir filen opprettet. Det er mange filtyper som er støttet, men i denne applikasjonen er det kun String som blir benyttet.

```
memory.getString("NØKKEL", "Verdi om nøkkel ikke finnes")
```

For å legge til ny/endre eksisterende data opprettes et objekt av SharedPreference-Editor. Denne blir knyttet opp mot Sharedpreference objektet beskrevet over. Under vises hvordan editoren settes opp mot SharedPreferences objektet og legger til en ny String i preference filen.

```
SharedPreferences.Editor memory_editor = memory.edit();

{memory_editor.putString("ROBOT-PORT1", portHSEC.getText());
```

3.3.12.2 MyArguments

MyArguments er en metode som ble laget for å sende data til de forskjellige fragmentene. Når et fragment blir opprettet for første gang(se Figure. 5 lifecycle) kan man velge å sende med en bundle med verdier kalt "Arguments". Bundelen inneholder data lagret i key-value verdier og kan hentes ut i fragmentet ved å bruke getArguments. På samme måte legges data til i Arguments ved å bruke (Fragment-navn).setArguments(Bundle).

Fordi dette kun kan gjøres ved første instansiering av fragmentet var ikke dette tilstrekkelig

for vårt bruk. Derfor ble det laget en metode som heter setMyArguments(Bundle). Når et fragment startes for første gang, blir Arguments først satt til å være en tom bundle. Når en bundle skal sendes til dette fragmentet, brukes setMyArguments(Bundle) hvor fragmentet selv oppdaterer sin Arguments-bundle. Dette er godkjent fordi en fragment selv kan oppdatere sin egen Arguments-bundle når som helst.

I setMyArguments blir først all data lagret i Arguments slettet. Deretter blir Arguments fylt med bundelen som kommer inn. Fragmentet holder etter dette ikke noen referanse til den originale bundelen, men har alle verdiene kopiert inn i sin Argument. Fragmentet kan nå fylle på med sin informasjonen i argumentet og sende denne videre til neste fragment med samme metode.

Det hadde også vært mulig å brukt bundelen direkte uten å først lagre den som Argument. Men Argument har den fordelen at de ikke forsvinner når en bytter til neste fragment. Dette fører igjen til at det er mulig for brukeren og gå tilbake til forrige fragment uten at informasjonen som var skrevet inn har forsvunnet.

3.3.12.3 Konvertering fra JSON

Når listene kommer tilbake fra server er disse lagret i en String og skrevet som et JsonArray. For å konvertere denne til et vanlig array lages først et JsonArray av responsen ved å bruke `jsonarray = new JSONArray(responseString);`. Deretter lages det en ArrayListe kalt resultat som skal inneholde det ferdige resultatet av konverteringen. Etter at vi har sjekket at jsonarrayet ikke er tomt, brukes en for-løkke for å hente ut en og en verdi av jsonarrayet og putte de inn i Arraylisten.

3.3.12.4 PropertyChangeListener

PropertyChangeListeners gjør det mulig for en klasse å få informasjon når en variabel endres. I klassen som skal informere, opprettes det et objekt av "PropertyChangeSupport". Denne klassen brukes for å sende ut meldinger når andre klasser skal informeres om en endring.

For at andre klasser skal få beskjeden må de opprette et objekt av PropertyChangeListener klassen. Denne knyttes til PropertyChangeSupport objektet gjennom metoden nevnt under.

```
pcs.addPropertyChangeListener(type.getType(), listener);
```

I eksempelet over er "pcs" PropertyChangeSupport-objektet. `type.getType()` brukes for å hente ut en hendelse fra ENUM klassen, og listener er PropertyChangeListener objektet.
Enumen ble laget for å unngå duplisering av kode. ENUM(type) holder på alle de tilgjengelige hendelsene(event) som kan oppstå. Dette fører til at samme metode kan brukes for å knytte "listenere" til forskjellige hendelser. En hendelse kan for eksempel være at jobblisten er oppdatert. Når en variabel i klassen blir oppdatert, brukes metoden under for å informere om hendelsen.

```
pcs.firePropertyChange(PropertyChangeItems.allUpdated.getType(), false, "success");
```

I dette eksempelet informeres det om en hendelese på "alle tilgjengelige jobber". Siste parameter

er den nye verdien på hendelsen, og informerer om at oppdatering av listen var vellykket. ”false” parameteret representerer den forrige verdien på hendelsen og blir ikke benyttet.

I tillegg er det laget en metode i global-variabel klassen som fjerner listeneren igjen. Denne brukes når fragmentene ikke lenger er synlige og trenger denne informasjonen.

3.3.12.5 Sammenligning av jobber i database og på robot

I applikasjonen trengte vi en metode for å identifisere de jobbene som ligger i databasen, men som ikke har blitt programmert på roboten. Metoden ble kalt `compare()` og lagt i ”storagebox” klassen. I metoden sjekkes først om listene fra roboten og databasen har blitt oppdatert. Dette blir gjort ved å sjekke 2 booleanske verdier som blir satt når listene oppdateres. Hvis ikke begge disse verdiene er ”true”, gjør ikke metoden noe mer. Hvis begge er ”true”, blir listen med tilgjengelige jobber tømt. Deretter brukes en ”for” funksjon for å gå gjennom alle verdiene i listen fra databasen. Hvis listen fra roboten inneholder den samme verdien, blir denne lagt til i listen over alle tilgjengelige jobber. For de som ikke finnes i begge listene, blir det lagt til ”(not found on robot)” bak verdien før den blir lagt til i listen. Når hele listen er gjennomgått, brukes propertychangeListener for å informere andre klasser om at listen er oppdatert. Til slutt settes de booleanske verdiene til ”false” i vente på at ny informasjon kommer fra databasen og roboten.

```
private void compare() {

    if(allsql && allrob){
    allAvailableTasks = new ArrayList<String>();
    for (String taskName : avaialableSQL) {
       if (avaialableRobot.contains(taskName+".JBI")) {
           allAvailableTasks.add(taskName+".JBI");
         }
        else{
            allAvailableTasks.add(taskName + "(not found on robot)");
        }
    }
    pcs.firePropertyChange(PropertyChangeItems.allUpdated.getType(), false, "success");

    allsql = false;
    allrob = false;
}

}
```

Figure 8: Sammenligning av database og robot

3.3.13 Feilhåndtering

3.3.13.1 Try and catch

Det er viktig å beskytte seg mot at applikasjonen kan krasje. Det er mange ting som kan føre til

krasj. For eksempel at man prøver å behandle en verdi som ikke eksisterer, som en konsekvens av at kommunikasjonen har blitt brutt. En metode for å beskytte seg mot dette er try-catch. Ved å legge risikable metoder i en try-metode, kan man spesifisere hva som skal skje ved en feil. En try-metode er alltid etterfulgt av minst en catch. I catch-metoden skriver man hva som skal gjøres ved en feil og ved hvilke typer feil denne koden skal bli kjørt. Ofte er det tilfelle at en metode kan resultere i flere forskjellige type feil, og det kan derfor være greit å ha flere catch-metoder som dekker forskjellige type feil. Under vises et eksempel ved bruk av try-catch.

```
try {
    response = httpclient.execute(httpPost);
    result = EntityUtils.toString(response.getEntity());

} catch (ClientProtocolException e) {
    // TODO Auto-generated catch block
    result = "clientProtocolExeption";
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    result = "IOexception";
    e.printStackTrace();
}
return result;
```

Figure 9: Eksempel på try and catch

3.4 ASA Database manager

3.4.1 WindowBuilder

For å lage grensesnittet til Database manageren, ble det brukt et tilleggsprogram for Eclipse kalt "WindowBuilder"[60]. Det er et program laget for å visuelt programmere et brukergrensesnitt ved hjelp av "drag and drop" funksjoner. I bakgrunnen blir det dannet en klasse i Java programmet som definerer plassering, størrelse og stil. Denne klassen ble kalt "GUI" og består av en konstruktør hvor metoden "initalize()" blir kjørt. "initialize()"-metoden initialiserer hele grensesnittet og bygger opp det visuelle akkurat slik det ble programmert. I "GUI" klassen ble det laget to andre metoder; "selectedLine()" og "update()". "selectedLine()"metodene blir brukt til å legge til en lytter til den aktuelle tabellen som er åpen. Mens "update()" oppdaterer den aktuelle tabellen ved å trykke på tabellen i valgmenyen igjen.

3.4.2 Popup vinduer

For å lage popup-vinduer i Java arver klassene fra JDialog. JDialog er en klasse i Java som tilbyr muligheten for å åpne et eget dialog vindu over det eksisterende vinduet.
I konstruktøren til klassen defineres først tittel, størrelse på vindu og hva som skal skje om krysset i hjørnet blir trykket(exit). Dette blir gjort ved å bruke:

```
setTitle("tittel");
setBounds(X, Y, bredde, høyde);
this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
```

For å sørge for at andre klasser skal vente til popup-vinduet er lukket før de fortsetter, brukes `this.setModal(true);`. Modal vil si popup-vindu ikke returnerer en verdi før bruker har lukket/skjult vinduet. Dermed blir andre klasser sittende og vente på denne verdien før de fortsetter.

Videre blir contentPane (hoved panelet) hentet frem og layout satt ved å bruke
`getContentPane().setLayout(new BorderLayout());`
Selve designet og oppbyggingen av brukergrensesnittet blir gjort i windowbuilder(3.4.1 windowBuilder).

For å vise vinduet har vi laget en metode som heter `.execute()`. Denne bruker `.setVisible(true)` for å gjøre vinduet synlig og returnerer en verdi. Som en konsekvens av "modal" vil ikke denne bli returnert før `.dispose()` blir kjørt.

3.4.3 Tilkobling

Tilkoblingen blir gjort i en metode kalt `connect()` i SQL-klassen.

I `connect()` sjekkes det først om bruker allerede har hatt en vellykket tilkobling til serveren. Hvis dette er tilfelle vil det ikke være nødvendig å be bruker om å fylle inn brukernavn/passord/adresse. Dette blir gjort ved at "accessgranted"-variabelen blir satt til true etter en vellykket forespørsel, mens ved feil blir denne satt til false.
Hvis "accessgranted" er false åpnes login-popup og det blir sjekket om en tidligere respons eksisterer. Hvis en tidligere respons eksisterer betyr det at login-data eller adressen er feil, og denne informasjonen blir presentert bruker via login-vinduet. Ved feil brukernavn/passord inneholder responsen «access denied» og dette formidles til bruker. Ved feil på server eller ved feil adresse er det ikke noen standard respons. Derimot vil dette føre til at programmet ikke klarer håndtere informasjonen som planlagt, og vi kan derfor basert på denne feilen, be bruker sjekke adressen eller server.

For selve tilkoblingen brukes HttpURLConnection som kommer fra java.net sitt bibliotek(3.2 Bibliotek). Under vises hvordan kommunikasjonen blir satt opp.

```
public void connect() {

    try {
        if (!accessGranted) {
            loginPop = new login();
            if (response.contains("Access denied")) {
                loginPop.showError("username/password");
            }
            else if(response.contains("unknown error")){
                loginPop.showError("unknown");
            };
            loginPop.execute();
            setPassword(loginPop.getPassword());
            setUsername(loginPop.getUsername());
            setURL(loginPop.getAddress());
        }
        adress = new URL(URL);
        connection = (HttpURLConnection) adress.openConnection();
        connection.setDoOutput(true);
        connection.setInstanceFollowRedirects(false);
        connection.setRequestMethod("POST");

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

Figure 10: Tilkobling til database

3.4.4 Henting av tabeller

For å holde det enkelt og unngå duplisering av kode valgte vi å ha en metode kalt `loadtable()` som kan laste inn alle tabellene. For å ha kontroll på hvilken tabell som skal hentes, tas det inn et parameter i form av navn på ønsket tabell. Først lages det en liste av kolonne navn tilhørende den ønskede tabellen. Denne listen blir ikke sendt, men brukt senere når tabellen (Jtable) skal lages.

```
String[] tempcolumn = { "Order", "ship", "section", "task", "done", "operator" };
```

(eks fra DagensJobb)

Selve forespørselen blir bygget som en lang String. For at PHP-skriptet skal klare å forstå hvilken verdi som er hva, er denne bygget opp på key-value prinsippet. I Stringen skrives først hva verdien er, etter fulgt av et «=» tegn. Deretter kommer verdien og avsluttes med et «&» tegn. «=» brukes for å si hva som hører sammen, mens «&» tegnet brukes for å si at det kommer et nytt «key-value» par.

Hele Stringer blir sendt i form av bytes ved å bruke:

```
connection.getOutputStream().write(parameters.getBytes());
```

Her er "connection" HttpURLConnection objektet som ble opprettet i `connect()` metoden.

Når responsen kommer fra serveren, blir denne først lagret i en String. Denne Stringen er skrevet i JSON, et format som ikke kan direkte fylles inn i et Jtable objekt, og må derfor konverteres til brukbart format. Måten dette ble løst på var å først konvertere den til et objekt via `JSONvalue.parse()`. Objektet blir så konvertert til et JSONArray igjen, fordi det er lettere å håndtere.

Når responsen er i form av JSONArray, brukes det en for-løkke for å fylle opp en matrise med innholdet. Denne matrisen og listen med alle kolonne navnene brukes da for å lage et nytt JTable objekt. For å gjøre det enklere for brukeren, valgte gruppen å utruste Jtable objektet med en defaultRowSorter. Dette fører til at oppføringene i tabellen kan sorteres alfabetisk. Den ferdige tabellen blir returnert til GUI-klassen som viser frem tabellen.

3.4.5 Legge til ny jobb

"Add new Task" popup-vindu ble laget for å gjøre det enkelt å legge til ny jobb i databasen. For å legge til en ny jobb må bruker fylle inn navn, workcall og opptil 10 parameter. For hvert parameter må det også spesifiseres adresse og type. Her valgte vi å lage en ENUM klasse over de mulige typene som roboten forstår.

Når brukeren har fylt inn all informasjonen om jobben blir det lagret i en vektor med data i form av String. Den første stringen som blir lagt til vektoren er Jobb-navnet. På samme måte som i henting av tabeller blir stringen bygd opp etter key-value prinsippet. Her skriver vi først inn at det er et jobbnavn etterfulgt av et "=" tegn. Deretter legger vi til det valgte navnet og avslutter med et"&" tegn. Stringen med jobbnavn blir lagt til i første posisjon i vektoren, og etterfulgt av den resterende informasjonen.

TASKNAME=TEST&WORKCALL=TEST.JBI&PARAM1=SPEED& ...

Når vektoren er komplett, brukes SQL-klassen til å sende informasjonen videre.

```
MYSQL.getInstance().addNewTask("ADDNAME", data);
```

I eksempelet over brukes "ADDNAME" for å fortelle at det er en ny jobb som skal legges til.

I `addNewTask()` blir først connect() brukt for å opprette en kommunikasjon(3.4.3 Tilkobling). Ettersom vi allerede har klart å hente inn data fra databasen, vet vi at passord og brukernavn stemmer. Grunnen til at vi vet dette er at når brukernavn/passord endres, vil det bli sendt en forespørsel om å få første tabell. Siden denne funksjonen har feilhåndtering, vil det ikke være nødvendig i innloggingsdata. Informasjon om innloggingsdata og data fra vektoren blir så lagret i en lang String og sendt.

```
connect();
response = "";
String parameters = "Username" + "=" + Username + "&" + "Password"
        + "=" + Password + "&" + "Method" + "=" + method + "&";

for (int index = 0; index < data.size(); index++) {

    temp = data.get(index);
    parameters += temp;
}
connection.getOutputStream().write(parameters.getBytes());
BufferedReader buff = new BufferedReader(new InputStreamReader(
        connection.getInputStream()));
```

Figure 11: Legg til jobb

Meldingen som kommer tilbake fra serveren indikerer om det var en vellykket operasjon. Siden popup-vindu er modalt, vil GUI vente til popup-vindu lukkes, hvor det oppdaterer den synlige tabellen.

3.4.6 Legge til dagens jobb

På samme måte som i "5.4.5 Legge til ny jobb", har vi valgt å bruke et popup-vindu for å legge til dagens jobber. Planen med dagens jobber var å gjøre det mulig for ledelsen å spesifisere jobber som skal utføres den dagen. Metoden som brukes for å legge til dagens jobb er den samme som i legg til ny jobb, men bruker "addTODAY" som et parameter for å fortelle hva som skal gjøres. For å gjøre det enkelt og oversiktelig ble det valgt å bruke en modal popup som brukergrensesnitt. Popup-vindu består av tekstfelt for ordrenummer, navn på skip, om jobben er gjort og hvem som gjorde den. De 2 siste er satt til å ikke kunne endres fra ASA DBM da dette er en del av sluttdokumentasjonen. I tillegg må det være et sted å skrive inn hvilken jobb det er snakk om. Her kunne vi også brukt et tekstfelt, men dette ville ført til at bruker kunne skrive en jobb som ikke eksisterer. For å løse dette valgte vi å vise en kombo-boks (dropdown-liste) med alle jobbene som eksisterer i databasen. Disse blir tatt inn som parameter når brukeren trykker på knappen for å legge til dagens jobber.

Når bruker har fylt ut alt, vil det på samme måte som i "legg til jobber" bla laget en vektor som inneholder alle verdiene. Også her benytter vi oss av key-value prinsippet for å gjøre det lesbart for skriptet. Når vektoren er komplett, brukes SQL-klassen til å sende informasjonen videre.

MYSQL.getInstance().addNewTask("ADDTODAY", data);

I eksempelet over bruker vi "ADDTODAY" for å fortelle at det er en ny jobb som skal legges til.

I `addNewTask()` blir først connect() brukt for å opprette en kommunikasjon(3.4.3 Tilkobling). Ettersom vi allerede har klart å hente inn data fra databasen, vet vi at passord og brukernavn stemmer. Grunnen til at vi vet dette er at når brukernavn/passord endres, vil det bli sendt en

forespørsel om å få første tabell. Siden denne funksjonen har feilhåndtering, vil det ikke være nødvendig i innloggingsdata. Informasjon om innloggingsdata og data fra vektoren blir så lagret i en lang String og sendt.

Meldingen som kommer tilbake fra serveren indikerer om det var en vellykket operasjon. Siden popup-vindu er modalt, vil GUI vente til popup-vindu lukkes, hvor på det oppdaterer den synlige tabellen.

3.4.7 Sletting av eksisterende data

For å kunne slette data ble det valgt å bruke en ListSelectionListener[61]. Denne brukes for å lage en referanse til det som ble valgt i tabellen. Det er flere modeller som kan velges, alt etter om det er enkelt verdier som skal velges, eller hele rader/kolonner. Det ble valgt å bruke SINGLE_SELECTION da denne velger en hel rad.

Når bruker trykker på knappen, sjekker GUI om en rad har blitt valgt og i hvilken tabell. Basert på type tabell blir popup-vinduet "confirmDelete" åpnet og valgt ordrenummer eller navn blir vist. I dagens jobber var det mer fornuftig å skille på ordrenummer ettersom det kan være flere oppføringen av samme jobb.

```
confirmDelete cd = new confirmDelete(mainTable.getValueAt(selectedRow,column)
.toString());
```

Når bruker har bekreftet valget blir MYSQL-klassen brukt for å sende forespørselen.

```
MYSQL.getInstance().deleteTask(method, mainTable.getValueAt(selectedRow, column)
.toString());
```

Her blir "method" brukt for å spesifisere tabellen, mens `maintable.getValueAt()` indikerer hvilken rad som skal fjernes.

I **deleteTask** blir tilkoblingen først satt opp(3.4.3 Tilkobling) før en key-value String med de nødvendige verdiene blir sendt.

```
connect();
response = "";
String parameters = "Username" + "=" + Username + "&" + "Password"
        + "=" + Password + "&" + "Method" + "=" + method + "&";
if (method.equals("DELETENAME")) {
    parameters = parameters + "TASKNAME=" + taskNameOrOrder;
} else {
    parameters = parameters + "ORDER=" + taskNameOrOrder;
}
    connection.getOutputStream().write(parameters.getBytes());
BufferedReader buff = new BufferedReader(new InputStreamReader(
        connection.getInputStream()));
```

Figure 12: Slette jobb

3.5 Databaser

3.5.1 Oppsett av Raspberry Pi

For å sette opp en Raspberry Pi med SQL-database, trenger en å installere et operativsystem. Det ble brukt en installasjonsfil kalt "NOOBS", dette er en enkel installasjon av et linux operativsystem kalt Raspbian. Denne må overføres til et SD-kort og vil bli installert så fort en monterer SD-kortet i enheten.
Når operativsystemet er installert, er enheten klar til å installere tilleggsprogramvaren for å sette opp SQL-databasen[50].
En starter med å åpne et terminal-vindu og skriv:

```
sudo apt-get install mysql-server
sudo apt-get install php5-mysql
```

SQL-serveren og bibliotekene for databasen vil bli installert etter disse er utført. Ved å utføre kommandoen:

```
mysql -p -u root
```

Vil en starte serveren, og må så lage en bruker med passord. Logg inn ved etterspørsel av innlogging og passord. For at den skal ha noen virkning trengs det å lages en ny database og en bruker:

```
CREATE DATABASE MY_DATABASE_NAME;
CREATE USER 'MY_USERNAME'@'localhost' IDENTIFIED BY 'MY_PASSWORD';
```

Det er nå mulig å komme seg inn på den mye databasen lokalt. SQL-databasen er nå oppe å kjører og er mulig å nå tak i fra maskiner på samme lokale nettverk.

Men for at en skal kunne nå tak i denne via PHP, trengs det en webserver[49]. Ved å installere en pakke kalt apache2 var det fort gjort å sette opp en webserver for enheten.

```
sudo apt-get install apache2 -y
```

I denne pakken er det lagt ved en test webserver, som vises ved å skrive "http://localhost/" i en nettleser på enheten, eller ip-adressen til enheten på en annen datamaskin. Det vil med dette hoppe opp en melding om at webserveren virker.

3.5.2 Laging av PHP-skript for MySQL

Kommunikasjon med databasen går via et PHP-Skript som ligger på en server. Fra ASA RC og ASA DBM blir det sendt en HTTP-POST melding med verdier, og det er skriptet sin jobb å videre formidle disse på rett måte.

Øverst i skriptet defineres de verdiene som er nødvendig for å sette opp tilkoblingen. I dette tilfellet vil det være brukernavn, passord, database-navn og database-adresse.
Fordi meldingene som kommer inn er av typen HTTP-POST brukes `$_POST['KEY']` for å hente ut verdier fra meldingen. Denne metoden henter ut verdien som er definert under den angitte nøkkelen (KEY).

For å sette opp kommunikasjon med databasen brukes:
```
$con=mysqli_connect($servername,$username,$password,$dbname);
if (mysqli_connect_errno($con)){
echo "Failed to connect to MySQL: " . mysqli_connect_error();}
```

Her blir det prøvd å sette opp en kommunikasjon med de angitte parameterne. Hvis det ikke er mulig å opprette en tilkobling, returnerer skriptet en melding med den oppståtte feilen. Den oppståtte feilen vil i dette tilfelle være definert i `mysqli_connect_error()`

Fordi skriptet brukes for å generere flere forskjellige forespørsler, blir en variabel brukt for å skille mellom disse. Denne variabelen er kalt "method" og kommer i post meldingen. Ved å sjekke verdien av "method"(key) oppimot de tilgjengelige metodene, blir rett del av koden kjørt.

Etter at metoden har blitt identifisert, hentes de resterende nødvendige verdiene fra posten. De forskjellige metodene har hver sin type forespørsel(query) utifra hva den skal gjøre. Forespørslene er definert som en lang String som sier hva som skal gjøres og hvor det skal gjøres. Under vises oppsett av tre forskjellige type forespørsler.

For å legge til verdier i databasen brukes
```
$query = "INSERT INTO [navn på tabell] (Verdi1,verdi2...verdiX)"
```

For å fjerne oppføringer i databasen brukes
```
   $query = "DELETE FROM [navn på tabell] WHERE [verdi] LIKE [Verdi som skal slettes]"
```

For å hente ut informasjon fra databases brukes
```
$query = " SELECT [RAD/KOLONNE] FROM [NAVN på tabell]
```

For å sende forespørselen brukes `$result = mysqli_query($con,$query)`, hvor `$con` er tilkoblin-

gen til databasen og `$query` er forespørselen. `$result` vil da inneholde resultatet av forespørselen.

I de tilfellene hvor vi ikke ber om en respons fra MySQL databasen, trenger vi bare å returnere en String som sier om operasjonen var vellykket. Dette oppnår vi ved å sjekke verdien på `mysqli_query($con,$query)`. Om denne er "true" returneres en melding hvor det står at det var en vellykket operasjon. Hvis denne derimot er "false", betyr det at det har skjedd noe feil. Det returneres da en beskrivelse av feilen ved hjelp av `mysqli_error()`.

I de tilfellene hvor vi skal hente ut informasjon fra databasen må vi først trekke ut de verdiene vi trenger fra responsen. Dette gjøres ved å sette opp en loop som henter ut alle verdiene og legger den i et Array.

```
else if($TableName == 'TODAY'){
    $sql = "SELECT * FROM workorders";
    $result = mysqli_query($con,$sql);

    while($r = mysqli_fetch_assoc($result)) {

        $rows[] = $r['WorkOrder'];
        $rows[] = $r['Ship'];
        $rows[] = $r['Section'];
        $rows[] = $r['Task'];
        $rows[] = $r['Done'];
        $rows[] = $r['Operator'];
    }
}
```

Figure 13: php- trekke ut verdier

Her brukes `mysqli_fetch_assoc($result)` for å gå gjennom responsen, hvor verdien av hver angitte nøkkel blir lagret i et Array kalt `$rows`

Til slutt blir resultatet returnert ved å bruke `echo json_encode($rows)`. Denne returnerer Arrayet i form at et JsonObjekt.

3.5.3 Skrive til database

(Denne seksjonen beskriver skriving til database fra Android Applikasjon. For ASA DBM, se 3.4.4 Henting av tabeller, 3.4.5 Legge til ny jobb, 3.4.6 Legge til dagens jobb og 3.4.7 Sletting av eksisterende data.)

Klassen som tar seg av sending av data til SQL-databasen heter SQLwriter og arver fra Async-

Task(2.2.11 AsyncTask).
I SQLwriter blir Apaches http-bibliotek som er inkludert i Android, brukt for kommunikasjon med PHP-skriptet. For å sende meldinger til PHP-serveren er det nødvendig å spesifisere URL adressen som tilhører denne. Denne URL vil i utgangspunktet være fast og er derfor lagret i minnet til applikasjonen. Dette inkluderer også brukernavn og passord for tilkobling til databasen. Disse verdiene blir hentet inn fra sharedpreferences (2.2.13 SharedPreferences) hver gang et nytt objekt av klassen opprettes.
I applikasjonen blir det laget et nytt objekt av klassen hver gang det skal sendes en melding, og dette fører igjen til at passord, brukernavn og URL alltid blir oppdatert før en melding sendes.

Fra Asynctask klassen får vi tilgang til en metode som heter `doInBackground()`. Dette er en metode som kjører på en egen tråd og kjører i bakgrunn av hovedtråden(2.2.10 Tråder). På grunn av restriksjoner i Android er det ikke tillatt å gjøre nettverks-forespørsler på hovedtråden og det er derfor nødvendig å gjøre dette på en separat tråd[62].
I doInbackgroud blir alle verdier som skal skrives til databasen lagret i hvert sitt NameValuePair-objekt, for deretter bli lagt til i en liste. NameValuePair baserer seg på Nøkkel-verdi lagring av data. Når man legger til en verdi(for eksempel SVEIS_1.JBI) må man også definere en "nøkkel" som brukes for å hente ut denne verdien igjen. I dette tilfelle er "SVEIS_1.JBI" navnet på en jobb og nøkkelen vil derfor være "TASKNAME".

```
parameters.add(new BasicNameValuePair("TASKNAME", "SVEIS_1.JBI));
```

Når all informasjonen er lagret i NameValuePairs og plassert i listen, blir listen lagt til i meldingen som skal sendes til PHP-skriptet.

```
httpPost.setEntity(new UrlEncodedFormEntity(parameters));
```

Meldingen blir deretter sendt og tråden blir stående og vente på en respons fra PHP-skriptet.

```
response = httpclient.execute(httpPost);
```

Når responsen kommer fra PHP-skriptet, blir denne konvertert til String og deretter sendt videre til `OnPostExecute()` metoden.

```
result = EntityUtils.toString(response.getEntity());
```

`OnPostExecute()` er en metode som også stammer i fra Asynctask og blir først kjørt når `doInbackground()` er ferdig. I denne klassen blir stringen sendt til LOGG-klassen hvor den blir lagt til loggen over hendelser.

```
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);

    System.out.println(result);
    LOGG.getINSTANCE().add(result);
```

Figure 14: OnPostExecute, SQLwriter

3.5.4 Lesing fra database

(Denne seksjonen beskriver lesing fra database via Android Applikasjon. For ASA DBM, se 3.4.4 Henting av tabeller, 3.4.5 Legge til ny jobb, 3.4.6 Legge til dagens jobb og 3.4.7 Sletting av eksisterende data.)

Klassen som tar seg av innhenting av data fra SQL-databasen heter SQLReader og arver fra AsyncTask (2.2.11 AsyncTask).

På samme måte som i SQLwriter(3.5.3 Skrive til database) blir URL, brukernavn og passord hentet fra sharedpreferences(3.3.12.1 SharedPreferences). Siden SQLreader brukes oppimot flere tabeller i SQL, blir det brukt en variabel kalt "method" som brukes for velge hvilken informasjon vi er interessert i.

- getNames - returnerer alle tilgjengelige jobber.
- getRows - returnerer alle parameterne som er spesifisert for den valgte jobben
- getTodays - returnerer alle jobbene som ligger i dagens liste, samt tilhørende informasjon

I `doInbackground()` metoden blir det først sjekket at første parameter er en av de stringene som er definert over. Hvis ikke det er tilfelle vil ikke PHP-skriptet forstå hva som skal gjøres. Parameterne kommer inn via `execute(String...params)` som blir brukt for å starte tråden. To parameter brukes i de tilfellene hvor man ønsker å lese av parameterne til en gitt jobb. Her vil første parameter være "getRow" og andre parameter navnet på jobben.

Hvis metoden er gjenkjent, blir brukernavn, passord og metoden lagret i hvert sitt NameValuePair-objekt. De blir så lagt til i en liste over parameter som skal sendes med forespørselen. NameValuePair objekt baserer seg på nøkkel-verdi lagring av data. Når man legger til en verdi(for eksempel getRow) må man også definere en "nøkkel" som brukes for å hente ut denne verdien igjen. I dette tilfelle er "getRow" navnet på metoden og nøkkelen vil derfor være "Method".

Deretter brukes Apache.Http biblioteket til å lage en ny «httppost». I "httpposten" blir parameterne lagt til, før den blir sendt.

```
httpPost.setEntity(new UrlEncodedFormEntity(parameters));
response = httpclient.execute(httpPost);
```

Responsen kommer i form av et JsonArray som inneholder alle de forespurte verdiene. Verdiene fra JsonArrayet trekkes deretter ut og lagres i et standard Array kalt "result".

```
jsonarray = new JSONArray(responseString);
result = new ArrayList<String>();
if (jsonarray != null) {
    for (int i = 0; i < jsonarray.length(); i++) {

        result.add(jsonarray.getString(i));

    }
```

Figure 15: Konvertering fra Json

Ettersom samme klassen blir brukt til å sende 3 typer forespørsel, er det nødvendig å skille mellom hvilken type forespørsel som blir brukt. For eksempel vil "result" inneholde alle jobbnavn hvis metoden var getNames, men hvis metoden var getRow vil det inneholde parameter. Dette blir gjort i `OnPostExecute()` metoden arvet fra Asynctask. `OnPostExecute()` blir brukt til etterbehandling av resultatet når `doInbackground()` er fullført.

Selv om posten blir generert og sendt på korrekt måte, er det alltid en underliggende fare for at noe kan gå galt. Dette kan for eksempel være at serveren eller databasen er nede, eller at brukernavn og passord er feil. For å kunne behandle dette ble det brukt "try-catch"(3.3.13.1 Try and catch) på de stedene vi forventer at det kan skje feil. I deklarasjonen av klassen er det definert en String som i utgangspunktet er satt til "success". Hvis det skulle oppstå en feil i løpet av koden blir denne stringen satt til å inneholde feilmeldingen.

Behandling av resultatet blir gjort utifra verdien på "method".

For getNames vil det tilsi at Arrayet med tilgjengelige jobber i storagebox blir oppdatert. Dette blir gjort ved å bruke `storagebox.getInstance().updateAvailableSQL(result,msg);`. For getRow vil koden være så å si det samme som for getNames. Resultatet inneholder parameterne til en gitt jobb og blir oppdatert i storagebox ved metoden `storagebox.getInstance().updateParameters(result,msg).`

For getToday vil etterbehandlingen bli noe annerledes. Resultat Arrayet fra `doInbackground()` vil i dette tilfelle være en lang liste med mange jobber. Hver jobb vil i tillegg være etterfulgt av tilhørende informasjon som ordrenummer, seksjon og skip. For å gjøre denne mer håndterbar har vi her valgt å splitte arrayet i mindre array. Med andre ord separere ut hver av jobbene med tilhørende informasjon og lagre de i en egen Arrayliste.

Dette blir gjort ved å bruke en for-løkke. I for-løkken blir en etter en verdi i arrayet trukket ut og lagret i et nytt Array. Når det nye Arrayet inneholder 7 verdier (tilsvarende 1 jobb), vil arrayet bli lagt til i et annet array og prosessen vil gjenta seg til det ikke er flere verdier igjen.

Resultatet blir et array som inneholder et array for hver jobb.

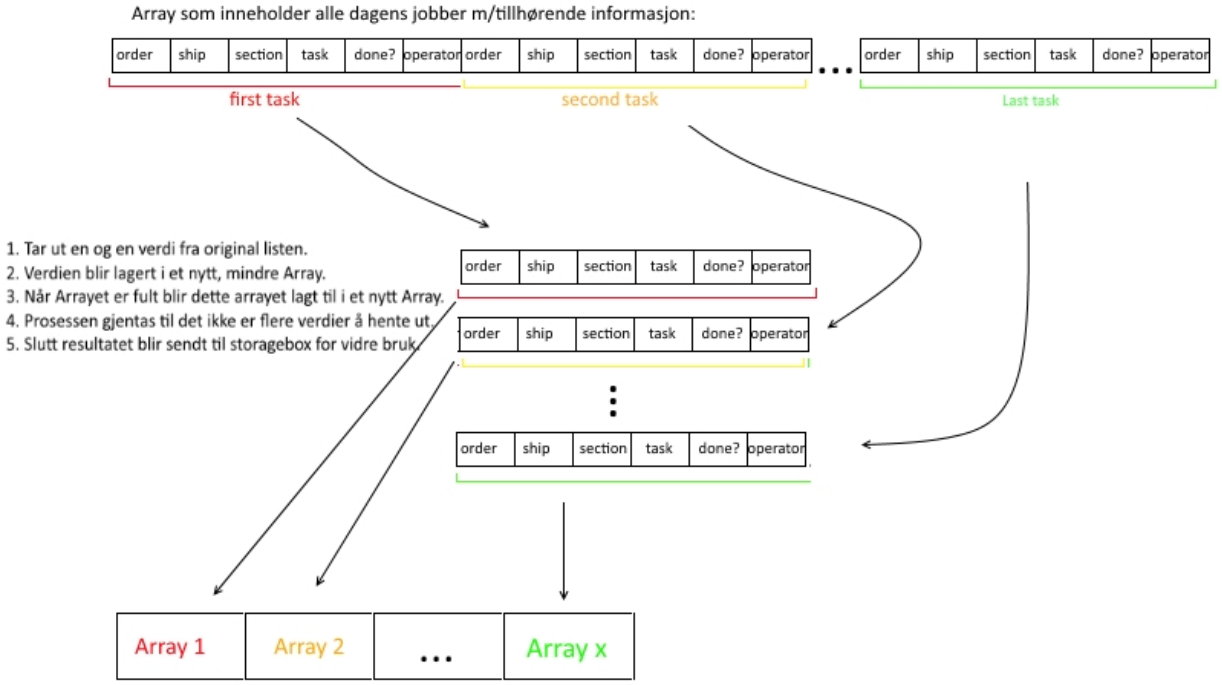


Figure 16: Splitting av Arrays

Til slutt blir arrayet lagret i storagebox ved å bruke

```
storagebox.getInstance().updateTodays(result,msg).
```

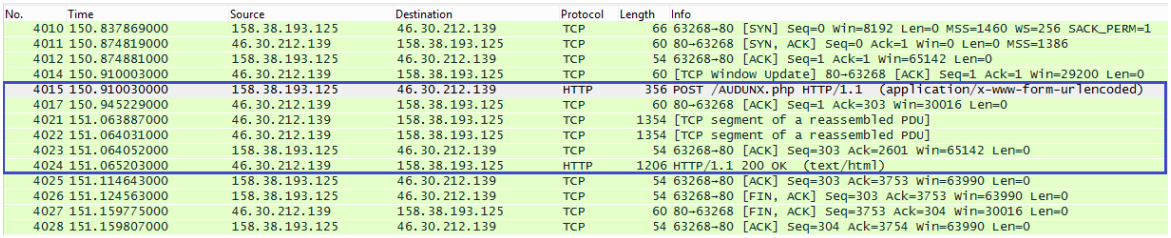
3.5.5 Pakke sniffing

For å overvåke trafikk opp mot server og robot benyttes Wireshark. For å overvåke trafikken som går utifra applikasjonen, blir denne kjørt i en emulator på datamaskinen. I Wireshark velges først hvilket nettverkskort som skal overvåkes. Alle pakker som blir sendt og mottatt av dette kortet blir dermed vist i et eget vindu. I vinduet kan man angi forskjellige filter som kan sortere ut kun de ønskede pakkene. Ved sjekking av trafikk til og fra robot brukes `udp.port eq 10040 and udp.port eq 10041`. Port 10041 brukes av HSEC for å hente listen med jobber på roboten, Mens 10040 dekker resten av kommunikasjonen med roboten. For å overvåke trafikken opp mot server brukes `ip.addr eg 46.30.212.139` som filtrerer ut alle meldinger som går til eller fra den spesifiserte IP-adressen.

Under vises et eksempel på trafikken mellom ASA DBM og server. I eksempelet ser man at

TCP kommunikasjonen først blir opprettet. Deretter blir forespørselen sendt som en HTTP POST og besvart av server. Tilkoblingen blir så lukket.
Ved å trykke på meldingene i vinduet, kan man se all informasjon som ligger i den valgte meldingen. Dette har vært en nyttig måte for gruppen å teste kommunikasjonen og se hva som faktisk kommer frem.

I bildet under er det brukt en blå firkant for å vise hvor forespørsel blir sendt og når dataene er mottatt.



No.	Time	Source	Destination	Protocol	Length	Info
4010	150.837869000	158.38.193.125	46.30.212.139	TCP	66	63268→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
4011	150.874819000	46.30.212.139	158.38.193.125	TCP	60	80→63268 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1386
4012	150.874881000	158.38.193.125	46.30.212.139	TCP	54	63268→80 [ACK] Seq=1 Ack=1 Win=65142 Len=0
4014	150.910003000	46.30.212.139	158.38.193.125	TCP	60	[TCP Window Update] 80→63268 [ACK] Seq=1 Ack=1 Win=29200 Len=0
4015	150.910030000	158.38.193.125	46.30.212.139	HTTP	356	POST /AUDUNX.php HTTP/1.1 (application/x-www-form-urlencoded)
4017	150.945229000	46.30.212.139	158.38.193.125	TCP	60	80→63268 [ACK] Seq=1 Ack=303 Win=30016 Len=0
4021	151.063887000	46.30.212.139	158.38.193.125	TCP	1354	[TCP segment of a reassembled PDU]
4022	151.064031000	46.30.212.139	158.38.193.125	TCP	1354	[TCP segment of a reassembled PDU]
4023	151.064052000	158.38.193.125	46.30.212.139	TCP	54	63268→80 [ACK] Seq=303 Ack=2601 Win=65142 Len=0
4024	151.065203000	46.30.212.139	158.38.193.125	HTTP	1206	HTTP/1.1 200 OK (text/html)
4025	151.114643000	158.38.193.125	46.30.212.139	TCP	54	63268→80 [ACK] Seq=303 Ack=3753 Win=63990 Len=0
4026	151.124563000	158.38.193.125	46.30.212.139	TCP	54	63268→80 [FIN, ACK] Seq=303 Ack=3753 Win=63990 Len=0
4027	151.159775000	46.30.212.139	158.38.193.125	TCP	60	80→63268 [FIN, ACK] Seq=3753 Ack=304 Win=30016 Len=0
4028	151.159807000	158.38.193.125	46.30.212.139	TCP	54	63268→80 [ACK] Seq=304 Ack=3754 Win=63990 Len=0

Figure 17: Eksempel på bruk av Wireshark

3.6 Robot Kommunikasjon

3.6.1 Kommunikasjons-klienter

3.6.1.1 MotoNIS MotoNIS SDK(2.1.8 Motonis) inneholder bibliotek og kildekode for å lage klienter i .NET og C, det følger også med en wsdl-fil. Denne type filer brukes for å beskrive funksjoner som en webtjeneste tilbyr, og kan brukes til å implementere disse funksjonene i programvare. Siden applikasjonen er laget i Java(2.2.1 Java), må gruppens klient programmeres i samme språk. Dette ble gjort ved å bruke wsdl-filen til å autogenere Java kode som implementerer funksjonene til webtjenesten. Den genererte koden lager en "stub" som er et objekt som representerer webtjenseten, når en metode blir utført på denne, sender det metodekallet videre til webtjenesten på roboten.

Slik brukes de genererte klassene til å utføre en robot-kommando, i dette tilfellet blir strømmen til servomotorene skrudd på:

```
CoreServiceStub stub = new CoreServiceStub(new java.net.URL("http://192.168.4.10:8080/"),
        new CoreService_ServiceLocator());
stub.setUsername("Motoman");
stub.setPassword("DX100");
IntHolder i = new IntHolder();
SERVOPOWERSENDDATA sp = new SERVOPOWERSENDDATA();
sp.setSServoPower((short) 1);
STDRSPDATAHolder resp = new STDRSPDATAHolder();
stub.nxSetServoPower(sp, i, resp);
```

Figure 18: MotoNIS eksempel

"Stub" bruker SOAP-protokollen(2.1.5 Simple Object Access Protocol) for å utveksle informasjonen, dette gjør at koden er avhengig av en del bibliotek. Detter er et problem fordi Android

ikke støtter alle bibliotekene som trengs for at denne klienten skal virke på nettbrettet. Denne metoden ble kun vurdert og testet, men ikke implementert i sluttresultatet(4.3.3 Robot).

3.6.1.2 Modbus og PLS

Som en metode til å kommunisere med roboten, ble det vurdert og påbegynt en klient som bruker Modbus(2.1.7 Modbus) opp i mot en PLS(2.1.6 PLS). En støttet feltbuss ville da blitt brukt for å sende data videre til roboten. Med denne metoden er det et veldig begrenset antall funksjoner som kan utføres, men siden metoden baserer seg på standardiserte protokoller, er den robust og rask å implementere. For kommunikasjon mellom nettbrett og PLS, ble det brukt et bibliotek som heter Jamod, som gjør det enkelt å implementere Modbus i Java(2.2.1 Java). Kommunikasjonsklassen er laget slik at den er lett å endre eller utvide i ettertid. Dette ble gjort ved å lage en enum klasse som linker alle funksjonene som skal taes i bruk til en Modbus adresse. Her er det satt av mer enn nok adresser til forskjellige formål, slik at det lett kan legges til ekstra funksjonalitet.

```
public enum Variable {
    //Robot inputs 12288-12499
    //Robot outputs 12500-12699
    //Robot Calls 12700-12899

    // Inputs 12288-12499
    joggXpluss(12288),
    joggXminus(12289),
    joggYpluss(12290),
    joggYminus(12291),
```

Figure 19: Variabel eksempel

Når en knapp blir trykt kjøres denne kommandoen, som da oppretter kommunikasjon med PLSen og sender verdi 322 til den adressen som joggXpluss er linket til som i dette tilfellet er 12288:

```
    ModbusMaster m = new ModbusMaster(0, Variable.joggXpluss, 322,
            "158.38.140.139");
}
```

Figure 20: Modbus master eksempel

Hver variabel vil da sendes videre av PLS til robot, der den vil bli behandlet av et master-program. Dette programmet må da stå for kalling av jobber, og oppbygging av posisjons variabler og andre variabler. Denne metoden ble kun vurdert og testet, men ikke implementert i sluttresultatet(5.4 Robot kommunikasjon).

3.6.1.3 Ethernet client

Klienten for "Ethernet Server function"(2.1.9 Ethernet Server) ble implementert ved hjelp av socket programmering utført i Java(2.2.1 Java). En enum-klasse ble laget for å representere alle funksjonene, hvor hvert funksjonsnavn er knyttet sammen med den representative robot

kommandoen, og tilhørende data. Dette gjør det oversiktlig og lett å legge til/fjerne/endre kommandoer i ettertid. Robot kommandoene og kommunikasjonen er definert i manualen(Vedlegg 10 MotomanEthernetServerFunction.pdf).

```
public enum Variables {

    getPosWorld("RPOSC", "1,1"),
    getPosJoint("RPOSJ", ""),
    getCurrentJob("RJSEQ" , ""),
    getStatus("RSTATS", ""),
    getJobs("RJDIR", "*"),
    holdOFF("HOLD", "0"),
    holdON("HOLD", "1"),
```

Figure 21: Enum eksempel

Transfer Klassen: Klassen som står for kommunikasjonen har en metode som heter transfer(), denne utfører all datatrafikk. Den oppretter en tilkobling med roboten gjennom TCP port 80(2.1.3 TCP og UDP) og utfører en handshake med roboten, slik at roboten er klar for å ta imot kommandoer. Den tar inn en funksjon fra enum-klassen som parameter og sender kommando og data til roboten og tar i mot responsen. Hver funksjon har sin egen metode der den kaller på transfer(), og tolker det svaret i fra roboten. Denne metoden ble kun vurdert og testet, men ikke implementert i sluttresultatet(5.4 Diskusjon, Robot kommunikasjon).

```
/** Gets the world coordinates of the robots position in a double array ...
public double[] getPosWorld() {
    String[] s = transfer(Variables.getPosWorld).split(",");
    double[] posW = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    for (int i = 0; i < s.length; i++) {
        posW[i] = Double.parseDouble(s[i])/1000.0;
    }
    return posW;
}
```

Figure 22: Transfer eksempel

3.6.1.4 High-Speed Ethernet client

For å kunne ta i bruk «High-Speed Ethernet server function»(2.1.10 High-Speed ethernet server) til DX100(3.1.3 DX100 kontroller) måtte en klient bli programmert til nettbrettet(3.1.1 Samsung Galaxy Tab S). Dette ble gjort ved å lage superklasser for forespørsel- og svarheader, og en forespørsel- og svarklasse for hver robotkommando. Superklassene er bygd opp etter det formatet som er spesifisert i protokollmanualen(Vedlegg 9 HSEC.pdf), og inneholder alt som er felles for alle meldinger som sendes til og fra roboten. Kommandoklassene arver fra superklassene, og inneholder det som er unikt for hver kommandomelding, som blant annet «commandNo»(Figur 3) og meldingsdata. Utifra disse klassene kan en da hente ut en melding som kan sendes ved hjelp av UDP(2.1.3 TCP og UDP) til roboten, og når roboten svarer blir dataen lagret og prosessert i en svarklasse.

```
public byte[] outData() {

    byte[] b = new byte[32];
    b[0] = identifier[0];
    b[1] = identifier[1];
    b[2] = identifier[2];
    b[3] = identifier[3];
    b[4] = headerPartSize[0];
    b[5] = headerPartSize[1];
    b[6] = dataPartSize[0];
    b[7] = dataPartSize[1];
    b[8] = reserve1;
    b[9] = processingDivison;
    b[10] = ACK;
    b[11] = requestID;
    b[12] = blockNo[0];
    b[13] = blockNo[1];
    b[14] = blockNo[2];
    b[15] = blockNo[3];
    b[16] = reserve2[0];
    b[17] = reserve2[1];
    b[18] = reserve2[2];
    b[19] = reserve2[3];
    b[20] = reserve2[4];
    b[21] = reserve2[5];
    b[22] = reserve2[6];
    b[23] = reserve2[7];
    b[24] = commandNo[0];
    b[25] = commandNo[1];
    b[26] = instance[0];
    b[27] = instance[1];
    b[28] = attribute;
    b[29] = service;
    b[30] = padding[0];
    b[31] = padding[1];

    return b;
}
```

Figure 23: This is the message header data.

Viktige klasser:

Messageheader: Dette er klassen som representerer forspørselheaderen til protokollen. Headeren er representert i form av felt med type byte og byte-array, og alle standardverdier blir satt her. Klassen har set- og get-metoder for hvert felt, i tillegg til metoden `outData()` som returnerer hele headeren som et byte-array med rett byte-rekkefølge (Little-endian).

Responseheader: Responseheader er klassen som representerer svarheaderen til protokollen. Den har metoden `fillHeader()` som fyller alle felt med data som kommer fra roboten, og den har set- og get-metoder for alle felt. Her skrives ut eventuelle feilmeldinger som kommer fra roboten. Alle feilkoder ligger i en enum-klasse som linker koden til tilhørende feilmeldinger som er definert i manualen(Vedlegg 9 HSEC.pdf).

HSEC: HSEC klassen står for å sende og motta meldinger fra roboten. Den er designet som en singleton-klasse(2.2.7 Singleton) og er derfor bare instansiert en gang. Den har en metode for hver kommando som er tilgjengelig. I hver metode blir en forespørselklasse instansiert, hvis en kommando har parameter (adresse, verdi, skrive/lese, etc...) blir det satt her. Deretter blir meldingen hentet ut som et byte-array, og sendt som et UDP-datagram til roboten på port 10040 (10041 hvis det er en jobbliste-kommando). Når roboten mottar meldingen utfører den kommandoen og sender en respons, denne responsen blir mottatt og lagret i en svarklasse som blir

returnert av kommandometoden. Alle kommandoer har en svarklasse som lagrer den forventede responsen fra roboten, og prosesserer den slik at svaret lett kan brukes videre.

```
public StatusResponse dxGetStatus() {
    try {
        InetAddress host = InetAddress.getByName(ip);
        socket = new DatagramSocket();
        StatusRequest req = new StatusRequest();
        req.setRequestID(requestID);
        byte[] request = req.getRequest();
        DatagramPacket packet = new DatagramPacket(request, request.length,
                host, port);
        socket.send(packet);
        requestID++;
        socket.setSoTimeout(2000);
        packet.setData(new byte[512]);
        socket.receive(packet);
        StatusResponse response = new StatusResponse(packet.getData());
        return response;
```

Figure 24: Eksempel: StatusResponse klassen

Eksempel på en svarklasse: StatusResponse er klassen som lagrer og prosesserer responsen fra roboten etter den har mottatt en StatusRequest. Konstruktøren til klassen tar inn byte-arrayet som kommer fra roboten, de 32 første fyller responseheaderen og de resterende er da datadelen av meldingen. Datadelen til en statusrespons består av to Integer verdier der hver bit representerer et statusparameter i roboten, dette er definert i manualen. Klassen bruker en metode som gjør to Integere om til et boolean-array som representerer verdien i bits, og setter verdien til hvert statusparameter til "true" eller "false". Dette gjør at hvert parameter kan sjekkes individuelt ved hjelp av get-metodene til klassen.

```
public StatusResponse(byte[] response){
    super();
    this.response = response;
    System.arraycopy(this.response, 0, respHead, 0, 32);
    fillHeader(respHead);
    System.arraycopy(this.response, 32, statusData1, 0, 4);
    System.arraycopy(this.response, 36, statusData2, 0, 4);
    int[] statusData = {bytesToInt(statusData1), bytesToInt(statusData2)};
    boolean[] status = getStatusArray(statusData);
    step = status[0];
    cycle1 = status[1];
    auto = status[2];
    running = status[3];
    safeOperation = status[4];
    teach = status[5];
    play = status[6];
    remote = status[7];
    holdByPP = status[9];
    holdExternal = status[10];
    holdCommand = status[11];
    alarming = status[12];
    errorOccurring = status[13];
    servoOn = status[14];

}

 /**
 * Checks if the robot is running
 * @return the running status
 */
public boolean isRunning() {
    return running;
}
```

Figure 25: StatusResponse klassen

PositionVariable: Når robotposisjonen leses eller en posisjonsvariabel leses/skrives er strukturen til datadelen av meldingene lik. PositionVariable er en klasse som inneholder denne datadelen, som er en samling av all data som blir brukt for å representere en robotposisjon. Dette gjør at en kompleks datatype som dette, på 52 bytes kan bli lagret i en klasse som gjør den lett å håndtere eller endre. Alle klasser som brukes for å sende/motta posisjoner har da set- og get-metoder for posisjonsvariabeler. Klassen har set- og get-metoder for alle koordinater slik at de kan vises eller endres individuelt.

```
public void setPositionVariable(byte[] data) {
    System.arraycopy(data, 0, dataType, 0, 4);
    System.arraycopy(data, 4, form, 0, 4);
    System.arraycopy(data, 8, toolNumber, 0, 4);
    System.arraycopy(data, 12, userCoordinateNumber, 0, 4);
    System.arraycopy(data, 16, extendedForm, 0, 4);
    System.arraycopy(data, 20, firstAxisData, 0, 4);
    System.arraycopy(data, 24, secondAxisData, 0, 4);
    System.arraycopy(data, 28, thirdAxisData, 0, 4);
    System.arraycopy(data, 32, fourthAxisData, 0, 4);
    System.arraycopy(data, 36, fifthAxisData, 0, 4);
    System.arraycopy(data, 40, sixthAxisData, 0, 4);
    System.arraycopy(data, 44, seventhAxisData, 0, 4);
    System.arraycopy(data, 48, eighthAxisData, 0, 4);
}
```

32bit integer	Byte 0	Byte 1	Byte 2	Byte3
1	Data type			
2	Form			
3	Tool number			
4	User coordinate number			
5	Extended form			
6	First axis data			
7	Second axis data			
8	Third axis data			
9	Fourth axis data			
10	Fifth axis data			
11	Sixth axis data			
12	Seventh axis data			
13	Eighth axis data			

Figure 26: Posisjons variabel klassen

Eksempel på forespørselklasse: ReadWritePulsePositionVariableRequest er den klassen som generer meldingen som sendes til roboten for å skrive/lese posisjonsvariabler. Den arver fra Messageheader-klassen som står for standardoppsettet av header-delen av forspørselen, men noen små endringer må gjøres. For at roboten skal vite hvilken kommando vi vil utføre settes "CommandNo" til 0x7f(hex), som da er koden for å skrive/lese posisjonsvariabler. Metoden "setPosition(PositionVariable pos)" blir brukt ved skriving, den tar inn en posisjonsvariabel og setter "DataPartSize" til størrelsen på posisjonsvariabelen. Alle forespørselklasser har en metode som heter "getRequest()", den returnerer meldingen som skal sendes til roboten i form av et byte-array.

```java
public class ReadWritePulsePositionVariableRequest extends MessageHeader {

    private byte[] dataPart;
    private boolean write;

    public ReadWritePulsePositionVariableRequest() {
        super();
        setCommandNo((short) 0x7f);
        setDataPartSize((short) 0x00);
    }

    /**
     * Set position parameter
     *
     * @param pos
     */
    public void setPosition(PositionVariable pos) {
        dataPart = pos.getPositionArray();
        setDataPartSize((short) dataPart.length);
    }

    /**
     * Generates request
     * @return request
     */
    public byte[] getRequest() {
        if (write) {
            return ArrayUtils.addAll(outData(), dataPart);
        } else {
            return outData();
        }
    }
```

Figure 27: Eksempel på en forespørselklasse

3.6.2 Implementering av kommunikasjonsklient

3.6.2.1 AsyncTask klasser Kommunikasjonen mot robot er noe som må foregå i bakgrunnen når en bruker applikasjonen. Dette gjøres ved å lage en klasse som arver fra AsyncTask(2.2.11 AsyncTask) for hver av de kommunikasjonsoperasjonene applikasjonen skal utføre:

- Hente Jobbliste
- Servo på/av
- Hold på/av
- Velge jobb
- Starte jobb
- Jogge-kommandoer og jogge fart

- Sende jobbparameter
- Reset alarm

Siden alle klassene arver i fra AsyncTask har de metoden doInBackground(), som blir brukt til å utføre bakgrunnsoppgaver. I denne metoden hentes det frem en instanse av HSEC-klassen(3.6.1.4 HSEC), der den videre blir brukt til å utføre ønsket kommando og eventuelle feilmeldinger blir rapportert. Etter dette blir resultatet publisert i onPostExecute() metoden, dette kan variere i type alt etter hva kommando som blir utført.

Eksempel på AsyncTask: Klassen som var laget for å hente jobblisten fra roboten heter GetJobList og arver i fra AsyncTask, slik at den kan utføres i bakgrunnen. Siden denne returnerer en liste med jobbnavn, må metoden doInBackground() returnere en ArrayList av Strings. For å hente jobblisten må instansen av HSEC-klassen hentes og metodekallet `.dxGetFilesList()` kjøres. Et String-array blir returnert og en for-løkke sjekker at hver String inneholder ".JBI" som er jobb-filtypen før den legges inn i listen. Etter denne prosessen er ferdig kjøres metoden `onPostExecute()`. Den tar da resultatet fra doInBackgroud(), som i dette tilfellet er en ArrayList å lagrer det i storagebox(4.1.4.1 Storagebox).

```
public GetJobList(Activity activity) {
        this.activity = activity;
}

protected ArrayList<String> doInBackground(Void... arg0) {
        // TODO Auto-generated method stub
        try {
                String[] joblist = HSEC.getInstance(activity).dxGetFileList();
                for (String s : joblist) {
                        if (s.contains(".JBI")) {
                                System.out.println(s);
                                result.add(s);
                        }
                }
        } catch (Exception e) {
                msg = "Failed to get Joblist from robot... ";
        }
        return result;
}

protected void onPostExecute(ArrayList<String> result) {
        super.onPostExecute(result);
        storagebox.getInstance().updateAvailableROB(result,msg);
}
```

Figure 28: Jobliste AsyncTask

Lignende klasser blir da laget for alle de aktuelle robotkommandoene, der hver klasse er tilpasset til den funksjonen den skal utføre. Et godt eksempel på dette er klassen som sender jobbparameter, den sjekker typen til alle parameterne som tilhører en jobb som skal utføres. Der etter velger den rett metode for å sende data av valgt type og sender da alle utfylte parameter til roboten. Alle AsyncTask-klassene som er til robotkommunikasjon, bruker HSEC som er kommunikasjonsprotokollen gruppen har valgt å bruke (5.4 Drøfting, Robot Kommunikasjon). Hvis en annen protokoll skal taes i bruk må klassene lages på nytt eller tilpasses for denne.

3.6.2.2 TimerTask klasser

De kommandoene som sendes en gang har AsyncTask-klasser, for eksempel når en knapp har blitt trykket ned. I applikasjonen er det også kommunikasjon som utføres flere ganger med et gitt tidsintervall, og til dette brukes det TimerTask(2.1.12 Timer and Timertask). ASA RC har to TimerTasks som kommuniserer med roboten, PositionThread og StatusThread. PositionThread kjøres når roboten jogges, den samler inn robot posisjoner i sanntid og sjekker om en alarm har oppstått. StatuscheckThread kjøres etter robot jobben har blitt startet, den sjekker status og rapporterer om roboten kjører, har stoppet, eller har blitt pauset. Disse klassene arver i fra TimerTask og har metoden run(), i denne metoden har man de metodene som skal utføres i bakgrunnen. En Timer brukes da til å starte TimerTasken med gitte tidsintervall.

```
@Override
public void run() {

        try {
                response = HSEC.getInstance(activity).dxGetStatus();
                pcs.firePropertyChange(running, isRunning, response.isRunning());
                isRunning = response.isRunning();
                pcs.firePropertyChange(paused, isPaused, response.isHoldByPP()
                                || response.isHoldCommand() || response.isHoldExternal());

                isPaused = response.isHoldByPP() || response.isHoldCommand()
                                || response.isHoldExternal();

        } catch (Exception e) {
                System.err.println("somthing went wrong at statusthread");
        }
}
```

Figure 29: Metode i statusThread

Her ser vi run()-metoden til StatusThread, her blir statusen hentet ved å bruke instansen av HSEC-klassen(3.6.1.4 HSEC, viktige klasser) og metoden dxGetStatus(). Fra responsen ser man om roboten kjører, eller er satt i pause. En propertyChangeListner sjekker da om status har endret seg, og rapporterer videre om dette er tilfellet. For å starte denne TimerTasken brukes en Timer med metodenkallet scheduleAtFixedRate(). I denne metoden velges TimerTask, starttid og tidsintervallet i millisekund.

```
Timer timerT1;
timerT1 = new Timer();
timerT1.scheduleAtFixedRate(statusThread, startDate, 2000);
```

3.6.3 Jogging av robot

Jogging av robot kan gjøres på to forskjellige måter;

- Jogging i world-coordinates:
  Det betyr at roboten beveger flere ledd samtidig for å kunne flytte verktøyet til roboten i en retning om gangen. F.eks: X-aksen i pluss retning, eller Z-aksen i minus retning.
- Jogging i Joint-coordinates:
  Når roboten jogges flyttes kun et ledd om gangen. F.eks: Basen roterer, eller skulder-ledd flyttes opp eller ned.

Hver retning for hvert koordinatsystem ble da tildelt en definert verdi, så når knappen for gitt retning blir trykket ned skrives en variabel i roboten til den verdien. På roboten programmerte gruppen et program(Vedlegg 13 JOGG.JBI) som leser variabelen, og etter hva verdi den har beveger roboten i mange små inkrement i definert retning. Når knappen slippes endres verdien tilbake til 0 og roboten stopper bevegelsen. Posisjons-inkrementet er lagret i posisjons-variabler på roboten, og kan defineres som Joint eller World posisjoner. Dette gjør at vi kan lage et inkrement for hver eneste retning i begge typer koordinatsystem, og bevege roboten ved hjelp av IMOV robot-instruksjonen. Det er en bevegelse instruksjon som flytter roboten en angitt økning fra den aktuelle posisjonen(Vedlegg 12 DX100 Manual). Dette er løst i programmet ved å bruke Jump-instruksjonen, som hopper til en annen linj*e i programkoden. Der blir bevegelsen utført og programmet starter fra toppen igjen.

```
*A
Jump *Spluss if I001 = 6
Jump *Sminus if I001 = 7
...
Jump *Xpluss if I001 = 18
Jump *Xminus if I001 = 19
...
Jump *A

*Spluss
IMOV P100 V=I002
Jump *A

*Sminus
IMOV P101 V=I002
Jump *A
...

*Xpluss
IMOV P112 V=I003
Jump *A

*Xminus
IMOV P113 V=I003
Jump *A
...
```

Figure 30: Utdrag av robot program

4 RESULTATER

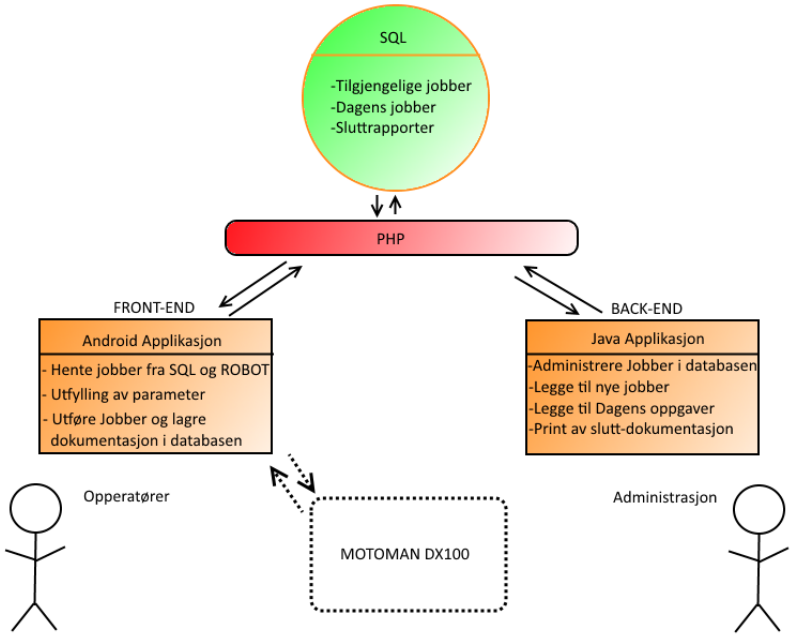


Figure 31: Enkel oversikt over planlagt kommunikasjon

Dette kapittelet vil omhandle sluttproduktet og resultatene gruppen kom frem til.

4.1 ASA Robot controller

Ved utvikling av denne applikasjon har gruppen brukt kravene fra Kleven som et utgangspunkt. I tillegg til kravene fra Kleven ønsket gruppen å tilby funksjoner som ikke ble spesifisert i kravene. Det innebærer funksjoner som gruppen så for seg kan forenkle operatørenes hverdag ytterligere og som kan være nyttige i fremtiden.
Ved programmering av applikasjonen ble det lagt til rette for at behov forandres, og at utstyr kan bli erstattet av annet utstyr. Dette ble blant annet gjort ved å bruke Java som programmeringsspråk, da dette er et av de mest brukte programmeringsspråk per dags dato[67].

4.1.1 Prosjekt oppbygging

For å tilrettelegge for videreutvikling av applikasjon, ble Java programmet delt i klasser med hvert sitt ansvarsområdet. «Java-doc» har blitt brukt for å dokumentere koden, og ligger vedlagt på minnepenn(Vedlegg 16 Java DOC). Klassene er i den grad det har latt seg gjøre, ikke direkte knyttet til hverandre, noe som gjør utskifting av klasser lettere og minsker sannsynligheten for følgefeil.

Den ferdige applikasjonen består av 1 activity og 5 fragments. I tillegg er det laget diverse klasser for sending og behandling av data. For å gjøre det lettere å holde oversikten, er disse fordelt i 8 prosjektmapper etter type eller bruksområde. En komplett oversikt over klassene er vedlagt i rapporten (Vedlegg 2 - KlasseDiagram ASA RC).

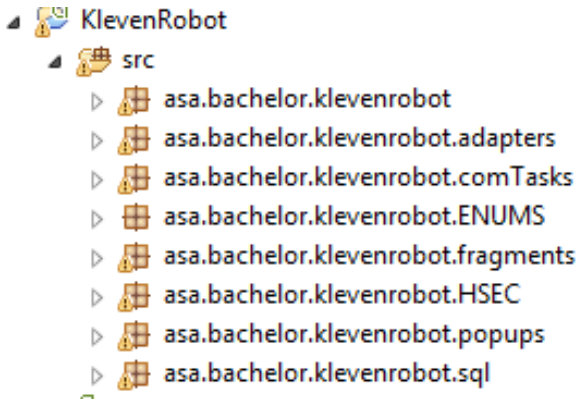


Figure 32: Prosjekt mapper, Android Applikasjon

- KlevenRobot - Inneholder kun MainActivity.
- KlevenRobot.adapters - Adaptere som brukes for å håndtere lister.
- KlevenRobot.comTasks - Klasser som håndterer kommunikasjon med robot.
- KlevenRobot.ENUMS - ENUM klasser
- KlevenRobot.fragment - Fragmentene som brukes i applikasjonen
- KlevenRobot.HSEC - High-Speed ethernet client klasser
- KlevenRobot.popups - eksterne popup-vinduer
- KlevenRobot.sql - Kommunikasjon mot database

I manifestet til Android applikasjonen ble det lagt til en tillatelse for å bruke internett (3.3.8 Tillatelser) for å utveksle informasjon med server og robot. Det ble også spesifisert en minste godkjent versjon (3.3.7 Spesifisering av enhets-versjoner) for å bruke applikasjonen (Android v4.2) og en mål-versjon (Android v4.4). Dette ble gjort for å sørge for at enheten kan bruke funksjonene som er implementert og for å sørge for at den er best mulig tilpasset enheten som skal brukes(Android v4.4).

Gruppen valgte også å spesifisere at applikasjonen kun skal støtte landskaps-modus (3.3.9 Skjerm orientering).

4.1.2 Brukergrensesnitt

I applikasjonen er det 6 fragmenter med hvert sitt individuelle brukergrensesnitt. Gjennom prosjektet ble det fokusert på å lage et så enkelt og forståelig grensesnitt som mulig. For å oppnå dette ble det i løpet av prosjektet testet flere forskjellige måter å presentere informasjon for brukeren. Dette ble gjort ved å lage forskjellige skisser i Draw.io før arbeidet ble startet, og ved å bruke grafisk programmering(se 3.3.10 Grafisk brukergrensesnitt)
Med informasjon menes den informasjonen brukeren får presentert, samt den informasjonen bruker må fylle inn for å utføre en jobb.

I det endelige resultatet er brukergrensesnittene rensket for farger, "fancy" løsninger og unødig informasjon. Det er også lagt opp til at bruker følger en trinnvis prosess. Med dette menes at bruker starter på første side, for så å følge en spesifisert rute frem til jobben er utført. Utifra hvilken jobb som har blitt valgt, blir dataene hentet inn fra databasen (3.5.4 Lesing fra database) og presentert for bruker. Det er også lagt til mulighet for å gå tilbake og gjøre endringer ved å bruke tilbakeknappen på enheten.(3.3.11.2 Pop to backstack)

For å sørge for at grensesnittet holdes enkelt, er all informasjon som ikke er en del av jobben skjult. Denne informasjonen er i stedet gjort tilgjengelig via knapper på meny-linjen, hvor et popup-vindu åpnes(3.3.1 Popup-vinduer).

I "Figure 33" vises det første designet gruppen kom opp med. I denne løsningen ble det brukt hvit tekst på mørk bakgrunn i tillegg til fargerike ikoner. I hoved-fragmentet ble bruker presentert for masse informasjon og de tilgjengelige jobbene ble presentert i en dropdown-liste. Etter gruppen hadde prøvd ut designet, kom de frem til at dette brukergrensesnittet var langt fra så enkelt som det kunne være.

I den endelige løsningen er informasjonen begrenset til et minimum for å vise kun det som absolutt er nødvendig. I tillegg er det blitt brukt flere fragmenter for å fordele informasjonen og for å legge opp til en logisk trinnvis prosess. Det endelige designet er vist i "Figure 34".

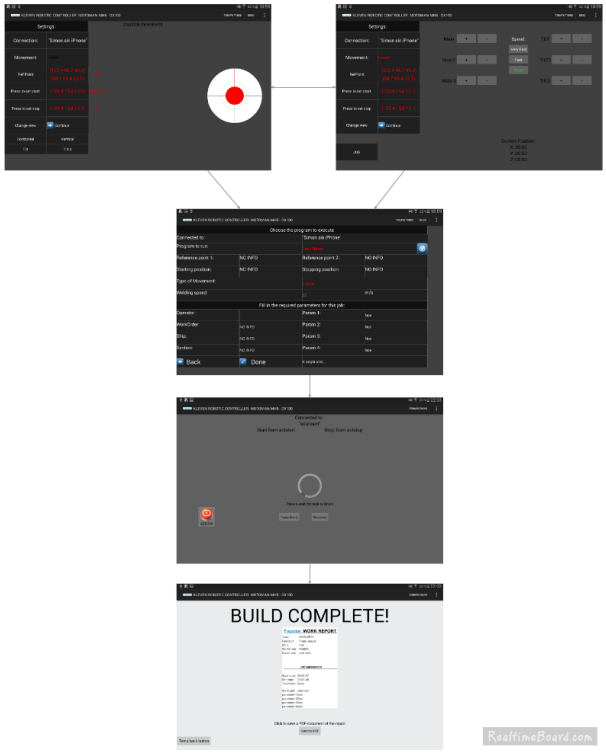


Figure 33: Design nummer 1

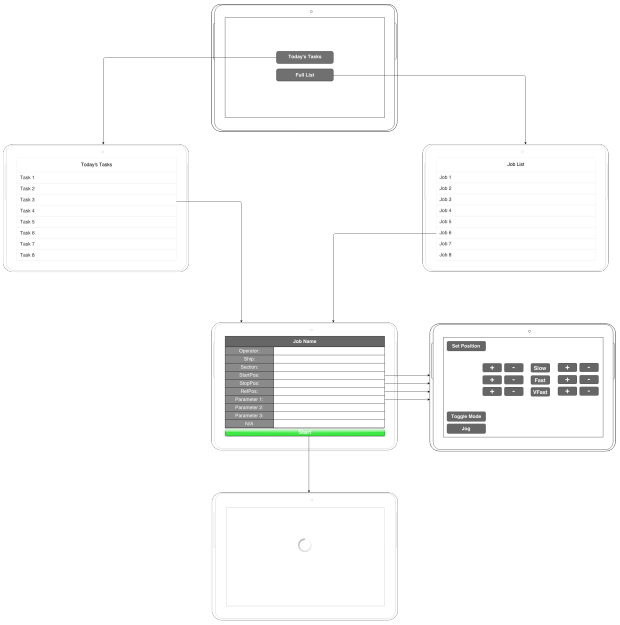


Figure 34: Design nummer 2

4.1.3 Innsamling av jobb-data

I applikasjonen blir dataene som trengs for å utføre en jobb, produsert i forskjellige fragments og samlet i en bundle. Bundle er en pakke som gjennom applikasjonen blir fylt opp med alle parameter roboten trenger. Fra fragmentene blir bundelen sendt til Activity, som bytter til neste fragment(3.3.11.1 Bytte mellom fragmenter) og sender bundelen videre via en metode kalt MyArguments (3.3.12.2 MyArguments).

I applikasjonen starter man med å velge mellom "alle tilgjengelige jobber" og "dagens jobber". Dette fører til at man kommer til fragmentet som viser den valgte listen over jobber. Her velger bruker en jobb fra listen, og det opprettes en "bundle" som sendes videre til neste fragment. I bundelen blir verdiene fra listen lagt til i form av key-value par. Hvert fragment har ansvar for å produsere en del av dataene og legge dette til den samme bundelen. Ved å hente inn informasjon om den valgte jobben fra databasen, vet fragmentene hva som er nødvendig for å utføre jobben. En jobb fra databasen er bygget opp på følgende måte:

- TASKNAME - Navn på jobb
- WORKCALL - Navn på program i robot
- PARAM1..10_LAB - Beskrivelse av parameter(Hastighet, temperatur, startpunkt osv...)

- PARAM1..10_TYPE - Type parameter(POS, INT, SHORT, LONG)
- PARAM1..10_ADR - Adresse i roboten

Disse verdiene danner grunnlaget for hva som må fylles inn før jobben kan utføres.
Hvis bruker valgte dagens jobber, vil databasen også inneholde følgende parameter:

- WORKORDER - ordrenummer
- SHIP - navn på skip
- SECTION - seksjon
- DONE - om jobben allerede er utført
- OPERATOR - hvem som var operatør

Listen under viser hvilke fragments som har ansvar for å fylle inn de forskjellige verdiene. Bruker kan velge mellom punkt 2 og 3, og de har derfor begge ansvar for å opprette en bundle og legge til navn på jobb. Parameter-verdi for posisjoner som legges til i punkt 5, brukes utelukkende for å vise bruker valgt posisjon. I utførelse av jobb er det et PositionVariable objektet som blir brukt. Denne klasses er beskrevet under "viktige klasser" i 3.6.1.4 High-speed ethernet client.

1. FragmentMAIN: Bruker velger mellom "dagens jobber" og "alle tilgjengelige jobber".
2. FragmentAllAvailalbleTasks: Bundle opprettes, legger til navn på valgt jobb.
3. FragmentTodaysTasks: Bundle opprettes, legger til navn, ordrenummer, skip, seksjon, operatør
4. FragmentParameters: Legger til Parameter navn, parameter verdi (ikke for posisjoner), parameter type, parameter adresse. Basert på om bruker valgte dagens - eller alle tilgjengelige jobber, legges ordre, skip og seksjon til bundelen.
5. FragmentJOGG - Legger til parameter-verdi for posisjoner (tekst representasjon av posisjon) samt et posisjons-objekt for endeledd og base.
6. FragmentLastPage - mottar bundelen som inneholder all data om en jobb.

Når brukeren kommer til siste fragment, har bundelen tatt runden gjennom alle fragmentene. Dermed har bundelen all informasjonen som trengs for å utføre den valgte jobben. I bildet under har vi prøvd å illustrere hvordan bundelen bygges opp.

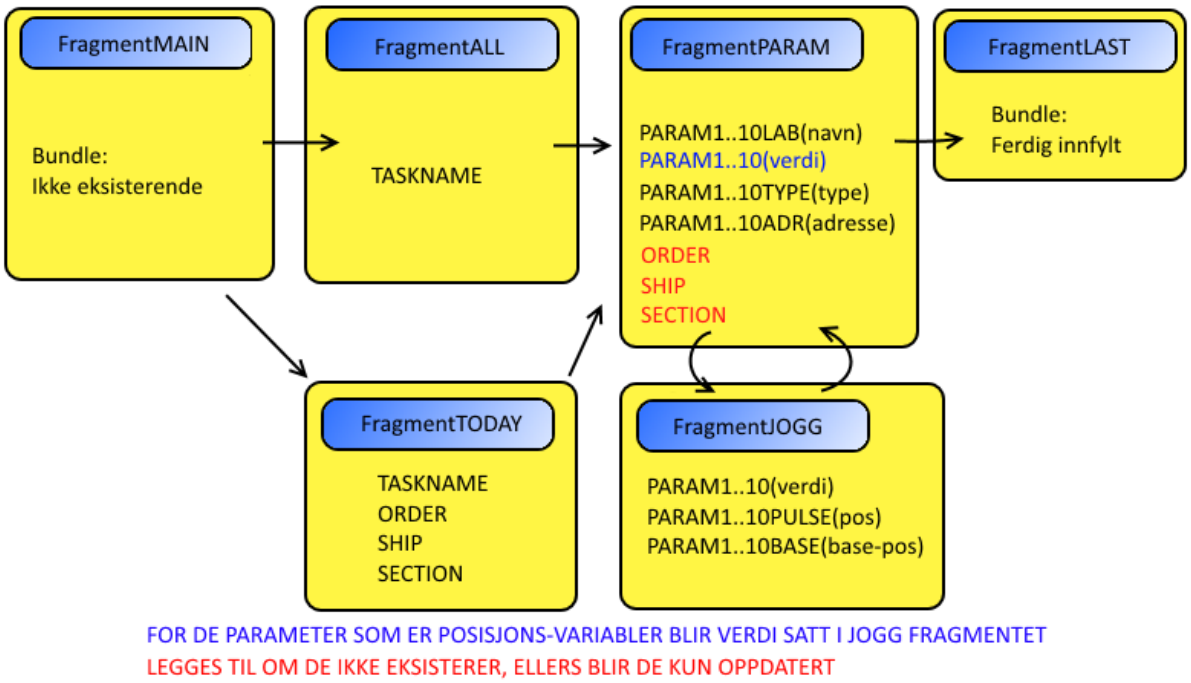


Figure 35: Grafisk fremstilling av pakke oppbyggning

4.1.4 Virkemåte

4.1.4.1 MainActivity

MainActivity er den eneste aktiviteten som blir brukt i applikasjonen. MainActivity kjører i bakgrunnen og har ansvar for å bytte mellom fragmenter og behandle trykk på meny-linjen. Ved trykk på meny-linjen blir MainActivity informert og oppretter et popup-vindu. For bytte mellom fragmenter er det laget en metode som heter `changeFrag()` som er forklart i 3.3.11.1-Bytte mellom fragmenter.

4.1.4.2 Storagebox

Storagebox er en klasse som blir brukt for å holde styr på dataene som kommer fra databasen. Når en forespørsel blir sendt til databasen, blir resultatet lagret i storagebox. For at alle klasser/fragmenter skal ha tilgang til dataene lagret her, er klassen utformet som en singleton(3.3.2 Singleton).

I storagebox er det implementert metoder for å hente og legge til data. Når ny data har blitt lagt til brukes PropertyChangeSupport klassen for å informere andre klasser om at data er klart til å bli hentet. Klassene som venter får denne informasjonen ved å implementere en PropertyChangeListener. Oppsett og virkemåte for PropertyChangeSupport og PropertyChangeListener er beskrevet i 3.3.12.4 - PropertyChangeListeners.

Siden "tilgjengelige jobber" må ligge på både robot og i database, blir disse to først sjekket oppimot hverandre i storagebox før andre klasser blir informert(3.3.12.5 Sammenligning av jobber i database og på robot).

4.1.4.3 Første side (Dagens jobber/Alle tilgjengelige jobber) Etter å ha prøvd oss frem på ulike metoder på hvordan applikasjonen burde være funksjonelt, ble det bestemt at det ville være mest fornuftig å kunne velge type jobb først. Resultatet er at den første siden kun består av 2 knapper. Den første tar deg til siden med de planlagte jobbene, mens den andre tar deg til siden med alle tilgjengelige jobber. Ved å trykke på en knapp, blir en beskjed sendt til activity om at dette fragmentet skal lukkes og at den valgte skal vises (3.3.11.1 Bytte mellom fragmenter).

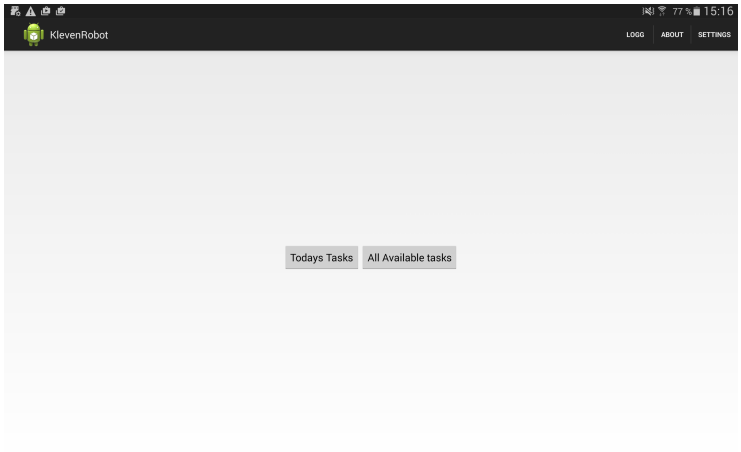


Figure 36: Brukergrensesnitt, side 1

4.1.4.4 Andre side (Dagens jobber/Alle tilgjengelige jobber)
Avhengig av om bruker valgte dagens- eller alle jobber blir det neste fragmentet i prosessen vist. Disse inneholder en dynamisk generert liste av jobbene. Når activity får beskjed om å vise en av disse fragmentene, blir det samtidig sendt en forespørsel til server(3.5.4 Lesing fra database) og et popup-vindu kommer til syne. Popup-vinduet brukes for å indikere at data holder på å bli lastet ned og for å vise eventuelle feilmeldinger om noe har gått galt. Popup-vindu blir brukt hver gang brukeren må vente på at dataene skal bli ferdig behandlet, og er laget ved å benytte AlertDialog klassen i Android(3.3.1 Popup-vinduer). For å vite når popup-vinduet skal lukkes, eventuelt vise en feilmelding, ble det benyttet PropertyChangeListener (3.3.12.4 PropertyChangeListener).

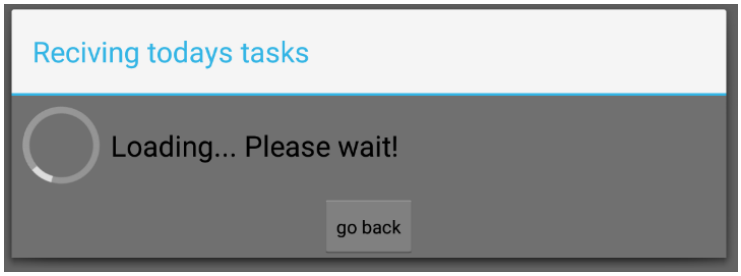


Figure 37: Loading popup

SQL klassen som tar seg av kommunikasjonen, sender resultatet av forespørselen (listen med jobber) til StorageBox-klassen. I tillegg sender den med en melding som indikerer om operasjonen var vellykket. Denne meldingen blir basert på resultatet av try-catch metoden(3.3.13.1 Try-catch) i SQL-klassen. Hvis noe går galt med innhenting av data, blir dette fanget opp av catch - metoden og en melding blir skrevet ut i fra typen feil. Hvis ikke catch fanger opp noen feil blir meldingen satt til å være ”success”. Når storagebox får listen og meldingen fra SQL-klassen, blir propertyChangeSupport brukt for å sende meldingen videre til alle klasser som venter på denne listen.
Hvis det er alle tilgjengelige jobber som skal vises, blir det først sendt en melding fra storagebox når listen fra databasen og roboten har blitt sammenlignet(3.3.12.5 Sammenligning av jobber i database og på robot). Dette gjør at bruker får informasjon om hvilke jobber som er mulig å utføre. De som ikke er mulig å utføre vil være skrevet i rød skrift og med teksten ≪not found on robot≫.
I vårt tilfelle er både popup-vinduet og fragmentet utstyrt med PropertyChangeListenere. Hvis meldingen som følger med er ”Success” blir popup-vinduet lukket, mens fragmentet vet at det er klart for å hente listen fra storagebox. Hvis forespørselen er noe annet enn ”success” indikerer dette at noe gikk galt. Popup-vindu viser dermed meldingen og lar bruker gå tilbake til forrige fragment via PopToBackStack metoden(se 3.3.11.2 PopToBackstack).

Hvis feilen er relatert til innloggingsdata, vil bruker kunne endre denne informasjonen via et popup-vindu kalt ”settings”. Settings oppdaterer verdiene som er lagret i minnet til applikasjonen ved å bruke sharedPreferences (3.3.12.1 SharedPreferences). Disse verdiene blir deretter hentet ut og brukt av SQL-klassen når en forespørsel skal sendes.

Listene er presentert i et Listview layout med mulighet for å bla ned-/oppover. Dette gjør at listen aldri blir for lang til å vises. For å vise dagens jobber med tilhørende informasjon på en rad ble det brukt et tilpasset adapter(3.3.6 Listview med flere elementer på hver rad). Det tilpassede adapteret gjør at listen kan vise flere verdier i på samme rad. Dette gjør at bruker kan se ordrenummer, skip og seksjon på jobben som skal utføres.

Når bruker velger en av jobbene i dagens-listen, blir ordrenummer, skip, seksjon og jobbnavn lagt til i en ny bundle. Ved valg i alle tilgjengelige jobber-listen vil det bli lagt til kun jobbnavn, og resten må fylles ut i neste fragment. Den nåværende synlige fragmenten blir deretter lukket og bundelen sendt videre til neste fragment. Det blir gjort ved å bruke changeFrag metoden i

activity(3.3.11.1 Bytte mellom fragmenter), som benytter seg av MyArguments metoden(3.3.12.2 MyArguments) for å sende bundelen videre.

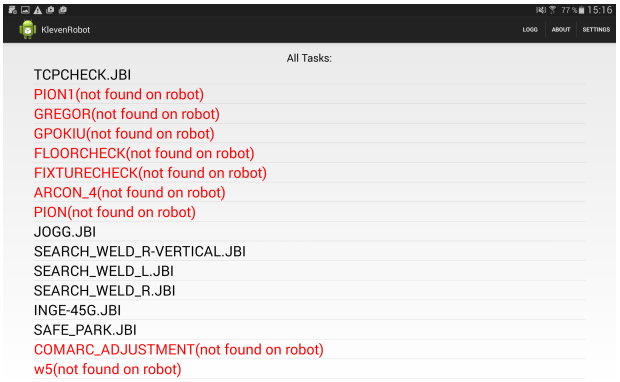


Figure 38: Alternativ 1, alle tilgjengelige jobber

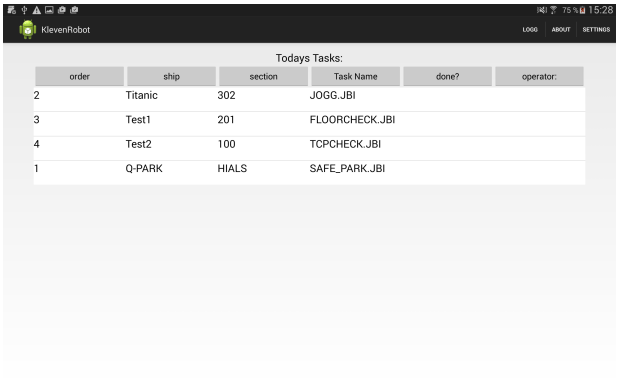


Figure 39: alternativ 2, dagens jobber

4.1.4.5 Tredje side (Parameter)

Når man går videre fra "dagens jobber"- og "alle tilgjengelige jobber"-listen kommer man frem til fragmentet hvor de resterende parameterne skal fylles inn. For å få vite parameterne som er knyttet til den valgte jobben blir det sendt en «getRow»-forespørsel til serveren(3.5.4 Lesing fra database). Det samme popup-vinduet som nevnt i seksjon "4.1.4.4 andre side", blir også her brukt for å informere om at data lastes ned og eventuelle feil.
Via PropertyChangeListener (3.3.12.4 PropertyChangeListener) får popup-vinduet og fragmentet vite når dataene er klare til å hentes.

Øverst i listen ligger informasjon som skal være med i sluttrapporten. Dette vil si ordrenummer, skip og seksjon. Hvis bruker valgte "dagens jobber" blir disse tatt fra bundle, men ellers må disse skrives inn manuelt.

Deretter kommer parameterne som er spesifisert i databasen. Dette kan være alt fra ingen til 10 stykker. I bakgrunnen ligger type parameter (INT, REAL, POS) og parameter-adresse, men dette blir ikke vist til bruker. I stedet blir bruker tatt automatisk over til jogg-skjermen hvis typen er en posisjon. Type og adresser blir definert i ASA DBM når jobben lages.
Jogg skjermen legger til et PositionVariable objekt(3.6.1.4 High-speed ethernet client, PositionVariable) og en tekst-representasjon av denne i bundelen. Tekst-representasjonen blir hentet ut av bundelen når bruker kommer tilbake til parametersiden. Her blir den vist i tekstfeltet knyttet til parameteret så brukeren kan ha et forhold til posisjonen.

Når brukeren trykker start blir all data lagret i bundle og sendt til det siste fragmentet.

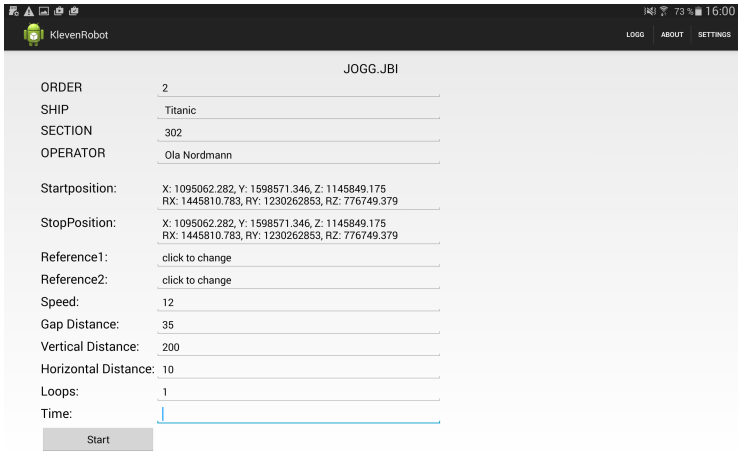


Figure 40: Brukergrensesnitt, side 3

4.1.4.6 Fjerde side (Jogging)

For å komme seg til "jogg"-seksjonen, må man trykke på en posisjonsvariabel i "Parameter" listen. Jogging var opprinnelig delt i 2 fragmenter, en ved bruk av joystick og en ved bruk av knapper. Joystick-metoden ble nedprioritert for andre tilleggsfunksjoner som ble sett på som viktigere og denne er derfor fjernet fra sluttproduktet.

I det endelige resultatet blir jogging utført ved bruk av knapper. Disse knappene er merket og plassert mest mulig likt løsningen på DX100. Figur 39 viser oppsettet på DX100, mens figur 40 viser oppsettet i applikasjonen. Fra jogg-skjermen er det mulig å velge mellom 3 hastigheter og jogge hver akse individuelt og i world-coordinates. I tillegg er det mulig å kjøre roboten(basen) frem og tilbake på en skinne ved hjelp av 2 knapper lokalisert nederst på skjermen.

Figure 41: DX100 layout

Når roboten jogges blir posisjonen til endestykket oppdatert i sanntid. Dette blir vist i 6 tekstfelt øverst i vindu. Dette blir gjort ved å bruke en timer med en timertask (3.3.4 Timer and TimerTask) som kjører i bakgrunn og ber om robotens posisjon. Innhenting av posisjoner går via HSEC og er forklart i *3.6.2.2 TimerTask Klasser*.
Fordi dette er en annen tråd enn den som lagde brukergrensesnittet, blir informasjonen oppdatert via `RunOnUiThread()` (3.3.5 RunOnUiThread). Dette sørger for at tråden som har eierskap til brukergrensesnittet får beskjed om å oppdatere tekstfeltene med den nye informasjon. Tekstfeltene angir endestykket sin posisjon i X, Y, Z, Rx, Ry og Rz.

Når roboten styres manuelt, er det alltid en fare for at roboten ikke kan komme seg til det ønskede punktet. Dette kan for eksempel skje om en av aksene er kjørt maksimalt i en retning. Dette vil føre til en feilmelding og at roboten stopper.
Her har vi valgt å la posisjons-tråden også lese eventuelle alarmer som skulle oppstå under jogging. Ved en feil åpnes et popup-vindu med den angitte feilmeldingen. Feilmeldingen forklarer hvilken akse det er som har nådd sin grense, og gjør det mulig å starte roboten igjen.

Når bruker trykker "set Position"-knappen blir posisjonen lagret i et PositionVariable-objekt. Dette er et objekt som ble laget for å holde all informasjonen om robotens posisjon(3.6.1.4 High-Speed Ethernet client, PositionVariable). Posisjonsobjektet blir lagret i bundelen i tillegg til base-posisjon og en tekst representasjon av posisjonen. Tekst representasjonen er en String satt sammen av X, Y, Z, Rx, Ry og Rz til endestykket. Denne Stringen er laget for å bli vist i parameter listen så bruker skal ha et forhold til posisjonen. Base-posisjon beskriver posisjonen til basen på roboten. Ettersom roboten kan kjøre frem og tilbake på skinner, er denne informasjonen nødvendig å sende med til roboten.

Bundelen blir sendt til Activity sammen med en beskjed om å bytte tilbake til parameter siden. Activity sender bundelen via MyArguments(3.3.12.2 MyArguments) og bytter til parameter-siden(3.3.11.1 Bytte mellom fragmenter) hvor listen har blitt oppdatert med den valte posisjonen.

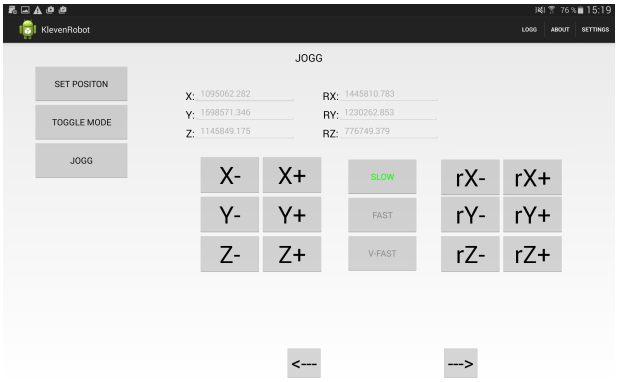


Figure 42: Brukergrensesnitt, side 4

4.1.4.7 Femte side (Utføring)

Når bruker kommer inn i det siste fragmentet, blir jobben satt på roboten, parameterne sendt, servoen skrudd på og jobben startes(3.6.2 Implementering av kommunikasjonsklient). All kommunikasjonen med roboten går via HSEC og utføres av klasser som arver fra AsyncTask klassen.

Øverst i brukergrensesnittet er det plassert en statuslinje som informerer om programmet kjører, er pauset eller ferdig. Denne informasjonen blir hentet via en timertask som kjører i bakgrunn. Via en Timer, blir tråden satt til å lese status hvert 0.5 sekund for å la bruker raskt få informasjon om status på programmet(3.3.4 Timer and Timertask). Fordi dette er en annen tråd enn den som lagde brukergrensesnittet, blir statuslinjen oppdatert via `RunOnUiThread()` (3.3.5 RunOnUiThread). Dette sørger for at tråden som har eierskap til brukergrensesnittet får beskjed om å oppdatere statuslinjen med angitt informasjon.

Under statuslinjen er det lagt inn en "loadingbar" som spinner så lenge programmet pågår. Utifra statusen på programmet blir denne erstattet med et rødt pause symbol når programmet er pauset, eller med en tekst som sier "ferdig". Alle disse ligger egentlig i brukergrensesnittet hele tiden, men ved å bruke `setVisibility(GONE/VISIBLE/HIDDEN)` blir kun den valgte synlig

for bruker. Når programmet er ferdig blir hele skjermen grønn. Dette er gjort for å gjøre det tydelig at jobber er ferdig. Bakgrunnsfargen blir endret ved å bruke

```
mainLayout.setBackgroundColor(Color.GREEN);
```

Her er mainLayout navnet på grensesnittet og Color er et medfølgene bibliotek i Android som definerer farger.

Knappen lengst til venstre brukes for å pause/gjenoppta programmet. Denne bytter mellom å vise PLAY og PAUSE utifra status på programmet. Hvis programmet kjører, sender den en melding til roboten om å pause. Hvis programmet er pauset, ber den roboten gjenoppta programmet. Meldingene blir sendt via klasser som arver fra AsyncTask(3.6.2.1 AsyncTask Klasser).

"Finish" knappen blir først tilgjengelig når statusen tilsier at jobben er ferdig. Dette ble enkelt og greit gjort ved å bruke `setEnabled(false)` når programmet starter og `setEnabled(true)` programmet er ferdig. Dette gjør at knappen er synlig, men grået ut og kan ikke trykkes. Knappen fjerner PropertyChangeListneren(3.3.12.4 PropertyChangeListener) fra tråden og stopper timeren(3.3.4 Timer and Timertask). Deretter sender den informasjon om jobben til databasen (3.5.3 Skrive til database), før den ber aktiviteten om å bytte til første fragment(3.3.11.1 Bytte mellom fragmenter). I aktiviteten blir historien over fragmentbytter fjernet (3.3.11.2 Pop To Backstack) og neste jobb kan bli utført.

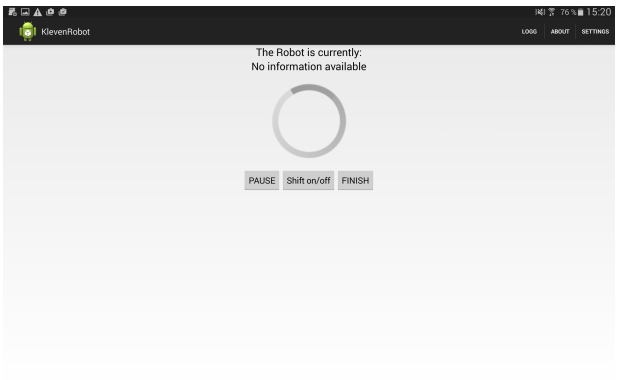


Figure 43: Brukergrensesnitt, side 5

4.2 ASA Database manager

ASA Database manager (ASA DBM) ble laget for å kunne manipulere databasen fra en pc, og applikasjonen ble laget i Java. I denne applikasjonen kan en se tabellene med tilgjengelige jobber, dagens jobber og utførte jobber. I tillegg er det implementert funksjoner for å legge til jobber, legge til dagensoppgaver, slette eksisterende data og skrive ut ønskede tabeller. Vår ≪back-end løsning≫ har vi valgt å kalle ≪ASA Database manager≫.

4.2.1 Oppbygging av ASA DBM

Det ble valgt å bruke en klasse som tar seg av hoved-brukergrensesnittet, og en klasse som tar seg av all kommunikasjonen med serveren. I tillegg har det blitt brukt popup-vinduer for å holde designet enkelt.

GUI-klassen står for brukergrensesnittet og er laget ved hjelp av windowbuilder(3.4.1 Windowbuilder) i Eclipse. GUI klassen håndterer hendelsene (knappe trykk, meny-valg) som oppstår og informerer de andre klassene. I tillegg brukes GUI klassen til å vise frem den valgte tabellen.

MySQL-klassen ble designet for å lage en klasse som kan ta seg av alt av sending og behandling av informasjon fra server. Denne klassen er i likhet med storagebox i ASA RC utformet som en singleton(3.3.2 Singleton). Det vil kort sagt si at det kun kan lages et objekt av denne klassen. Siden det finnes bare et objekt av denne klassen, kan vi bruke en metode som kalles `GetInstance()`. Dette gir oss enkel tilgang til dette objektet fra alle andre klasser uten bruk av masse referanser.
Klassen inneholder informasjon om brukernavn, passord, adresse, samt metodene som brukes for å sende meldingene. Denne informasjonen blir brukt for å sette opp en ny Http kommunikasjon hver gang en metode utføres. Kommunikasjonen blir satt opp ved å bruke Java.net-biblioteket og er beskrevet i seksjon 3.4.3-Tilkobling.
Hvis ikke denne informasjonen er korrekt får man ikke et fornuftig svar fra databasen. På grunn av dette blir sendingen og tilkoblingen lagt i en loop. I loopen blir login-vinduet med feilmeldingen vist og forespørsel sendt helt til tilkoblingen er vellykket.

Popup-vinduene er laget ved å arve fra Jdialog(3.4.2 Popup vinduer) og ved å bruke Windowbuilder(3.4.1 Windowbuilder). Under er popup-vinduene i applikasjonen beskrevet.

login brukes for å la bruker fylle inn informasjonen som er nødvendig for å koble seg til databasen. Dette innebærer brukernavn, passord og nettadresse. Popup-vinduet blir vist når bruker starter applikasjonen eller hvis tilkoblingen feiler. Feil brukernavn/passord blir oppdaget når man prøver å hente en tabell og er beskrevet i 3.4.4-Henting av tabell.

confirmDelete brukes for å be bruker konstatere at en oppføring skal fjernes fra databasen. Popup-vinduet blir åpnet fra GUI når bruker trykker på "delete selected row" og viser enten ordrenummer eller jobbnavn alt etter hvilken tabell som vises. Om bruker konstaterer at oppføringen skal fjernes, benyttes SQL-klassen for å utføre kommandoen (3.4.7 Sletting av eksisterende data).

addNewJob brukes for å la bruker fylle inn informasjon om en jobb som skal opprettes. Vinduet inneholder tekstfelt for jobbnavn, workcall, og 10 parameter med tilhørende type og adresse.

addNewTodaysTask brukes for å la bruker velge en jobb som skal utføres og fylle inn informasjon om denne. Vinduet inneholder tekstfelt for skip, seksjon, ordrenummer, om jobber er gjort(done) og operatør. I tillegg er det en dropdown-liste med tilgjengelige jobber å velge mellom.

settings brukes for å la bruker endre brukernavn, passord og adresse etter applikasjonen har startet. Denne var også tenkt å tilby flere valg som for eksempel type database, men gruppen rakk ikke implementere dette.

4.2.2 Virkemåte

Ved å starte applikasjonen blir man først bedt om å logge seg inn. Dette er brukernavnet og passordet som blir brukt til å koble seg til databasen(3.4.3 Tilkobling). I tillegg er det mulighet for å endre adressen til PHP-skriptet.

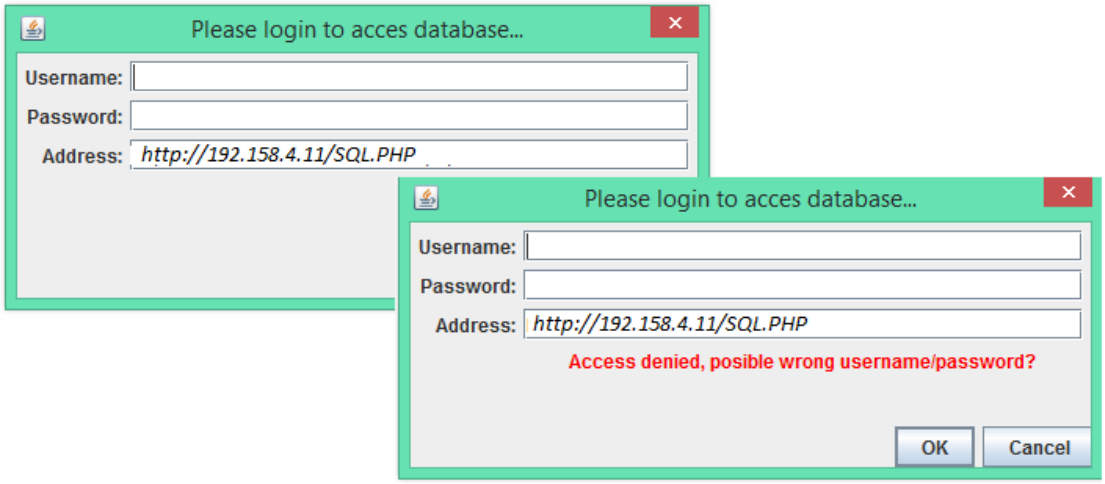


Figure 44: Login med og uten feil

Hovedvinduet består av den valgte tabellen, samt en meny på venstre side for å velge ønskede tabell. Ved å trykke på en av knappene, blir en forespørsel sendt til databasen og vinduet oppdatert(3.4.4 Henting av tabell). Tabellen er utstyrt med en "slider" for å kunne se hele tabellen. I tillegg kan alle kolonner strekkes og krympes etter eget ønske. Det er også mulig å sortere oppføringene alfabetisk ved å dobbeltrykke på kolonne navnet. Ved å markere en rad kan denne fjernes ved å trykke på knappen nederst i vindu (3.4.7 Sletting av eksisterende data).
Fra menylinjen er det mulig å endre avanserte innstillinger, skrive ut den valgte tabellen og legge til ny data.

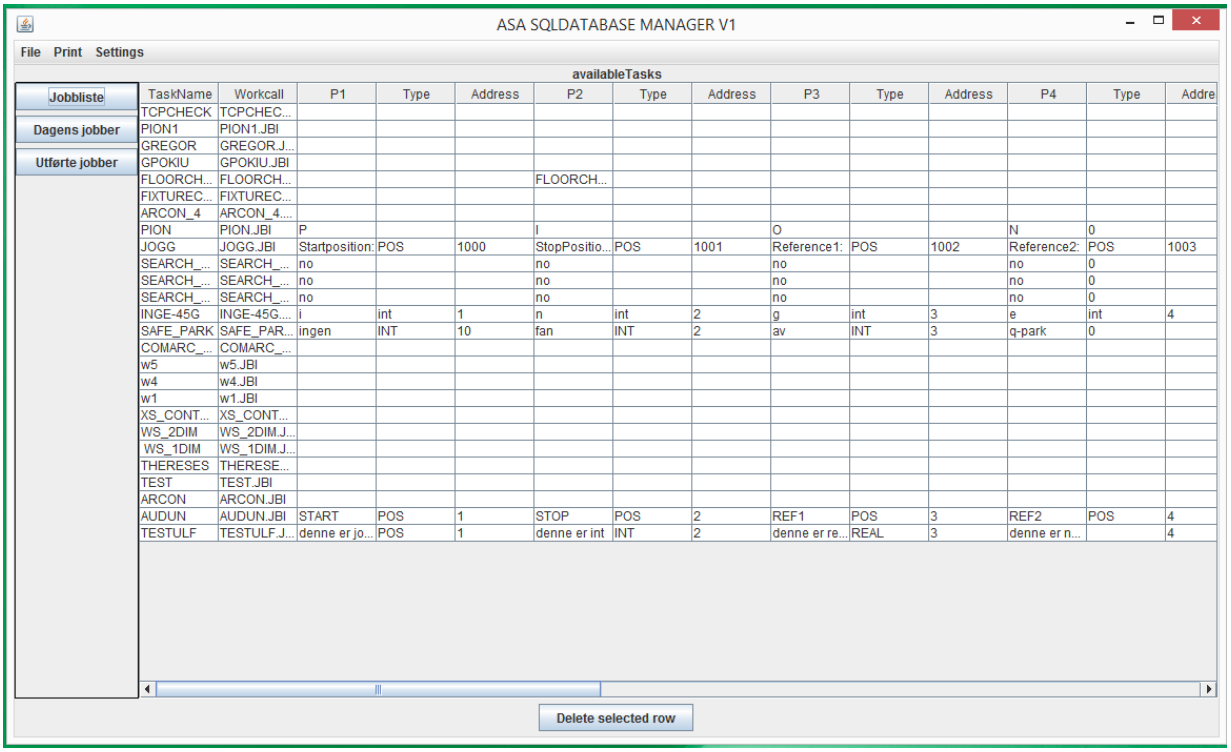


Figure 45: Hoved vindu med "alle tilgjengelige jobber"

Ved å velge avanserte innstillinger får man opp et vindu hvor passord, brukernavn, URL og SQL-type(ikke implementert) kan endres. Endring fører til at en forespørsel blir sendt til databasen om å oppdatere gjeldene tabell(3.4.4 Henting av tabeller). Hvis passord, brukernavn eller URL ikke er godkjent vil man bli bedt om å legge inn informasjonen på nytt.

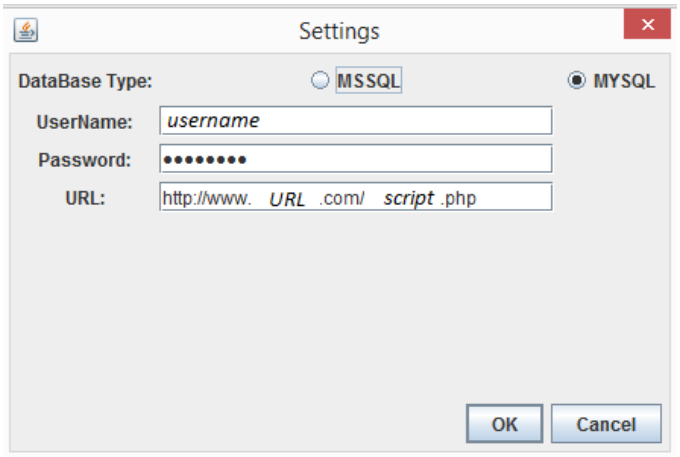


Figure 46: Advanced settings

Ved å velge "print" får man opp et standard utskriftsvindu. Vinduet er automatisk generert ved å bruke `.print() i Java`. Her er det mulighet for å velge skriver, endre innstillinger etter hva skriveren støtter.

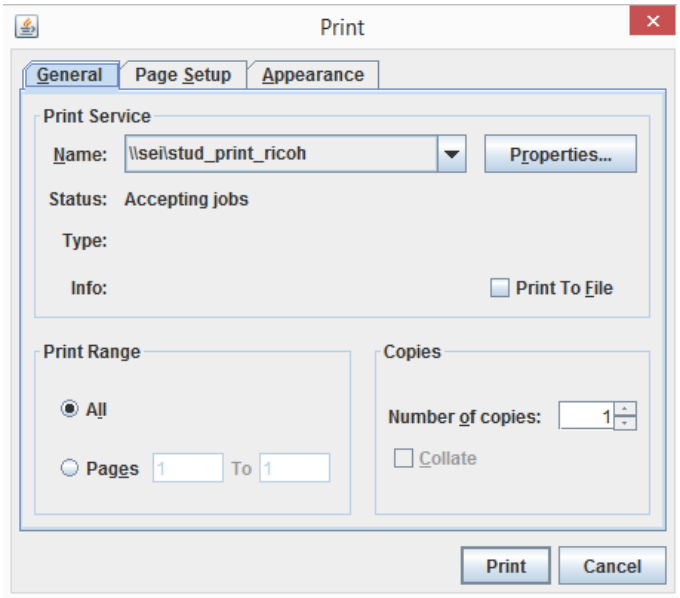


Figure 47: Skriv ut valgt tabell

Ved å velge "File" og "legg til ny jobb" får man opp et vindu hvor man kan føre inn all informasjonen om den nye jobben som skal legges til. I dropdown-listen under "Parameters types" kan man velge mellom støttede typer(INT,REAL,POS). Ved å trykke OK blir verdiene lagret i en String og sendt til serveren. Hvis tabellen med tilgjengelige jobber allerede blir vist i bakgrunnen, vil denne automatisk oppdatere seg med den nye jobben. Bildet under viser et skjermdump av popup-vindu med eksempel på utfylling. Parameterne som er "NONE" vil ikke komme opp i Android applikasjonen når jobben skal utføres(3.4.5 Legge til ny jobb).

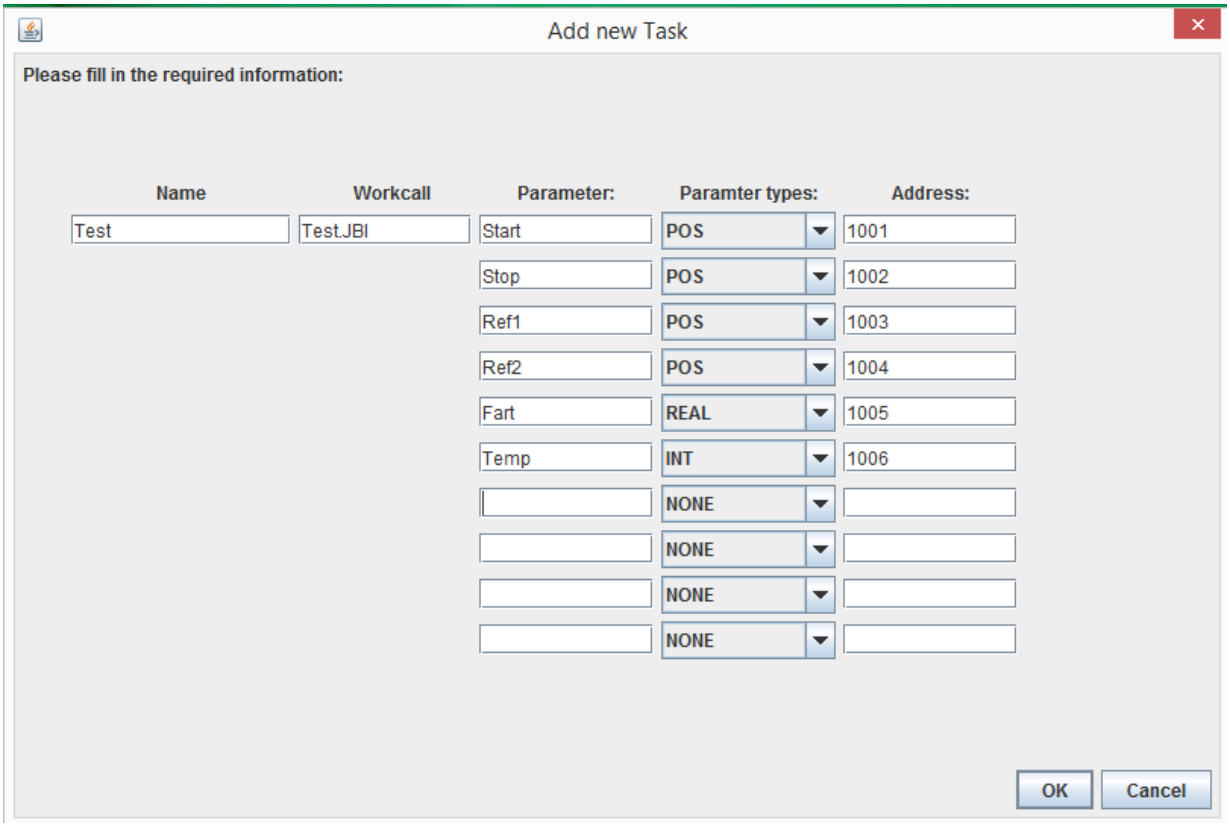


Figure 48: Legge til ny jobb

Ved å velge "File" og "legg til ny dagens jobb" får man opp et vindu hvor man kan velge hvilken jobb som skal utføres, samt legge med ordrenummer, navn på skip, og seksjon. De to siste tekstfeltene er ikke mulig å endre fra "ASA DBM" da disse blir oppdatert når jobber er fullført. For å unngå feil blir navn på jobb valgt utifra en liste av jobber som er tilgjengelige. Ved å trykke "Ok" blir informasjonen sendt via SQL-klassen og tabellen oppdatert.(3.4.6 Legge til dagens jobb)

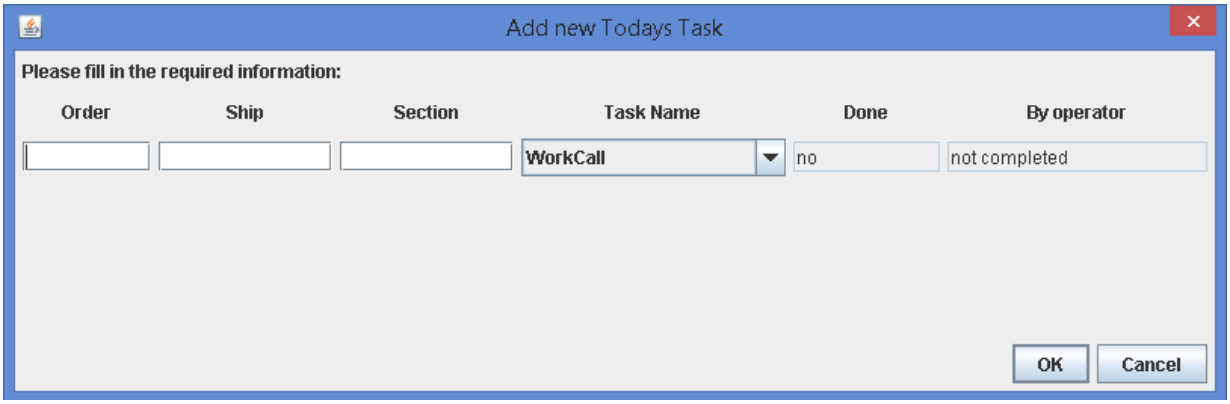


Figure 49: Legge til en jobb i dagens gjøremål

4.3 Kommunikasjon

4.3.1 Valg av database-system

Gruppen forstod det slik at Kleven benyttet en MySQL server, og at det var ønskelig å benytte denne videre. Gruppen satt derfor først opp kommunikasjon med MySQL. Etterhvert viste det seg at kleven benytter seg av Microsoft sin versjon, MSSQL og det var derfor nødvendig å tilpasse koden i PHP-skriptet. Sett fra enheten var det ikke nødvendig med noen forandring da all kommunikasjon går via PHP-skriptet. I PHP-skriptet måtte en del forandringer til ettersom MySQL og MSSQL bygger opp forespørslene på forskjellige måter. Da gruppen allerede hadde en fungerende MySQL kommunikasjon ble det valgt å ikke slette denne, men legge det med som en mulighet som kan benyttes for fremtiden.
I et møte lenger ut i prosjektet kom gruppen og oppdragsgiver frem til at gruppens system ikke skulle integreres i deres database. Dette var fordi at det trådløse nettet i produksjonshallen ikke var optimalt, så det er en sjanse for at en ikke får koblet seg opp mot databasen. Så før eventuell utbedring av dette, er det best at systemet kan stå på egne ben. Det ble da bestemt at en trådløs-router og en Raspberry Pi(3.5.1 Oppsett av Raspberry Pi) skal være i robotskapet, der Raspberry Pi kjører databasen og webserveren. Siden Raspberry Pi kjører linux måtte gruppen da gå tilbake til MySQL, dette var da ikke noe problem fordi gruppen sin applikasjon støtter både MSSQL og MySQL.

4.3.2 Database

Det er satt opp en MySQL database på en Raspberry Pi som inneholder 3 tabeller. Den første tabellen inneholder alle tilgjengelige jobber, inkludert alle parameter som trengs for å utføre den valgte jobben. Den andre tabellen inneholder alle dagens jobber, inkludert informasjon om ordrenummer, skip, seksjon, operatør og om den er utført. Den siste tabellen inneholder data om jobben som er utført.

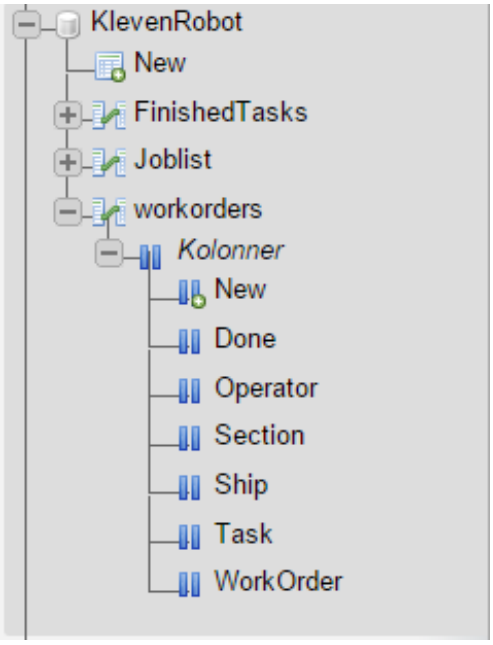


Figure 50: Oversikt over database tabeller

Fra den offisielle android utvikler siden får man informasjon om hvilke type lagrings medier Android støtter[63]. Utifra denne listen kan man se at Android støtter SQLite, men at hverken MySQL eller MSSQL er nevnt. For å kunne bruke disse var det nødvendig å benytte uoffisielle biblioteker eller gå via et nettskript.

Gruppen har valgt å bruke en server som kjører et PHP-skript(3.5.2 Laging av PHP-skript for MySQL). Dette skriptet virker som en tolk mellom applikasjonene og databasen. Skriptet ble skrevet i PHP ettersom dette er det mest vanlige språket å bruke for å sende SQL forespørsler. PHP støtter også Json typer, som gjør at man i Java kan motta og håndtere informasjonen på en enkel og oversiktelig måte.
Serveren ble lagt på en Raspberry Pi sammen med databasen og tar seg av all kommunikasjon opp mot databasen, både fra applikasjonen og back-end.

PHP-skriptet er laget slik at det henter ut verdiene fra HTTP-POST meldingen og lagrer verdiene. Det er laget et PHP-skript som tar seg av alle forespørsler opp mot databasen. Det er også samme skript som tar seg av forespørslene fra både ASA RC og ASA DBM. For at PHP-skriptet skal vite hva verdiene som kommer inn er, blir det sendt som key-value par. Nøklene er de samme som brukes i applikasjonene. Ved å se på verdien til "Method"-nøkkelen som kommer inn, utfører skriptet den valgte forespørselen. Listen under viser de tilgjengelige metodene som er implementert i PHP-skriptet.

- GETROW - alle tillhørende parameter for en angitt jobb.
- GETNAME - alle tilgjengelige jobber i databasen.

- GETTODAY - alle jobber planlagt for denne dagen.
- GETTABLE - hente ut den angitte tabellen i sin helhet.
- ADDNAME - legge til en ny jobb.
- ADDTODAY - legge til en jobb i dagens oppgaver.
- ADDSLUTT - legge til sluttdokumentasjon
- DELETENAME - slette en jobb fra listen over tilgjengelige jobber.
- DELETETODAY - slette en av dagens jobber.
- DELETESLUTT - slette en oppføring i sluttrapporten.

Når databasen har sendt en respons til skriptet, blir denne informasjonen overført til ASA RC i form av et JSON objekt. Det er også laget et PHP-skript som kan brukes opp mot MSSQL, men dette er ikke testet i sin helhet. PHP-skriptet som tar seg av MSSQL forespørsler har ikke blitt benyttet da vi endte opp med å bruke MYSQL.
I applikasjonen er det laget 2 klasser, SQLreader og SQLwriter. Disse klassene tar seg av kommunikasjonen med SQL databasen. Fordi det ikke er tillatt å gjøre nettverksforespørsler fra hovedtråden, arver disse klassene fra AsyncTask klassen. SQLreader tar seg av lesing fra databasen, mens SQLwriter tar seg av skriving til databasen.
I ASA DBM er det laget en egen klasse som tar seg av all kommunikasjon med SQL-databasen.Her brukes java.net biblioteket for å sette opp kommunikasjonen med serveren.
I begge klassene brukes Apache‘s http-biblotek for kommunikasjon med serveren. I begge applikasjonene sendes informasjonen over til serveren som en HTTP-POST melding.

4.3.2.1 Responstider

Ved testing av applikasjon har programmet blitt kjørt på en simulator. For å teste responstidene ble det skrive ut CurrentTimeMillis() ved start av sending, ved mottatt respons og etter dataene har blitt behandlet. Under vises resultatene gruppen kom frem til. Det er viktig å påpeke at resultatet kan i stor grad påvirkes av nettverks-kvaliteten på tidspunktet målingen blir tatt.

Forespørsel mot database	
TYPE	TID ms
Alle jobber	50ms
Dagens jobber	29ms
Hente sluttrapport	32ms
Legg til ny jobb	34ms
Legg til dagens jobb	30ms
Slett oppføring	37ms
Gjennomsnitt:	35ms

Figure 51: Responstid SQL

4.3.3 Robot

Kommunikasjonen mellom nettbrettet og roboten foregår etter klient-server prinsippet(2.1.4 Klient-server kommunikasjon). Dette krever en felles kommunikasjonsprotokoll(2.1.1 Kommunikasjonsprotokoller) som begge må forstå. Valget falt på motoman sin funksjon "High-Speed Ethernet server function"(2.1.10 High-Speed Ethernet Server), dette er en server som kjøres på robot-kontrolleren som kan brukes til å kommunisere med roboten. Protokollen som serveren bruker er definert i manualen til denne funksjonen (Vedlegg 9 HSEC.pdf). En klient som prater med serveren ble da programmert etter denne spesifikasjonen, og integrert i applikasjonen. I klassediagrammet(Vedlegg 2, Klassediagram ASA RC) ser vi at det er to hovedklasser som alle forespørsler og responser arver fra. De to klassene representerer strukturen til alle meldinger som sendes og mottas, mens underklassene inneholder det som er unikt med hver kommando. HSEC-klassen(3.6.1.4) blir da brukt til å sende og motta meldinger, til og fra roboten. Denne strukturen gjør det lett å legge til eller endre metoder, samtidig som det er oversiktlig. I figuren ser vi hvordan klassene blir brukt fra oppsettet av forespørsel til tolkning av respons.

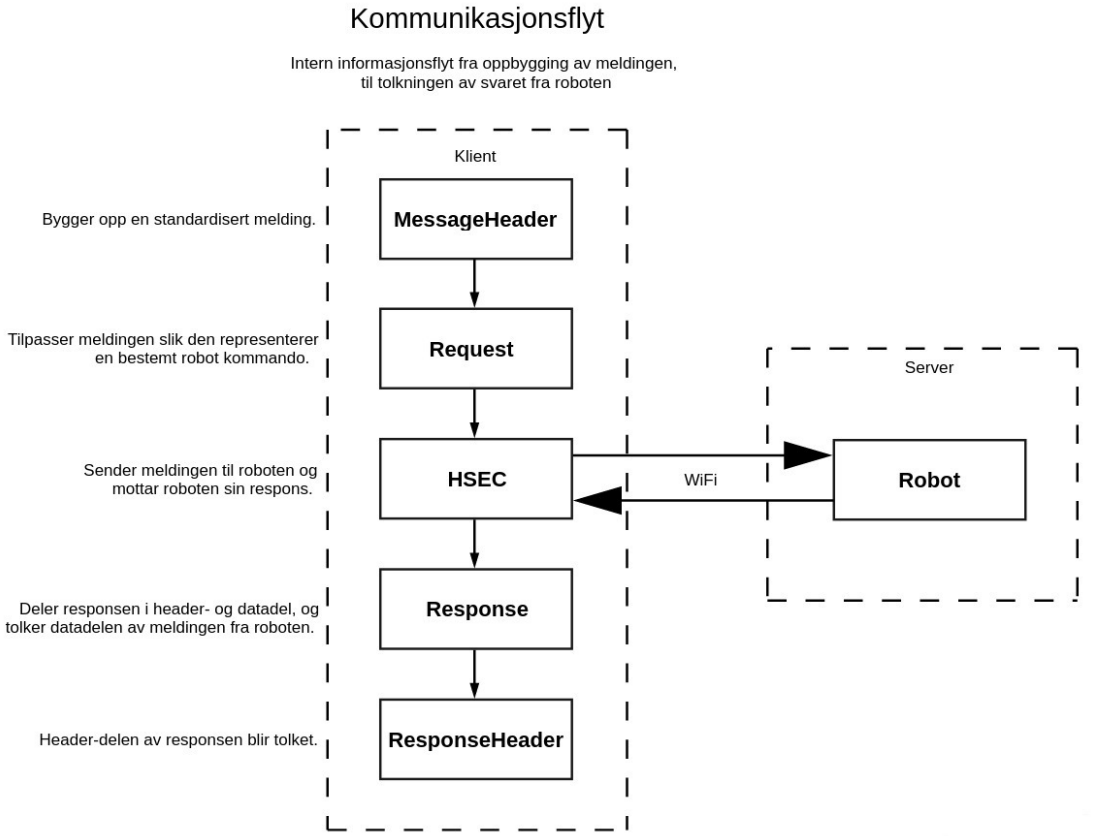


Figure 52: Programflyt

Hovedgrunnene til at denne protokollen er valgt er responstiden og utvalget av funksjoner den kan utføre. Sammenlignet med de andre protokollene fra Yaskawa er funksjonaliteten lik, men responstiden er vesentlig raskere. Gruppen vektet responstid høgt ved valg av protokoll, dette er fordi at ved jogging av roboten er det viktig at det er liten forsinkelse på knappene. Hvis joggingen av roboten fungerer dårlig vil dette ødelegge hele konseptet med å gjøre ting enklere for operatører. Responstiden ble da målt for funksjonen vi bruker til jogging og for funksjonen som henter jobblisten, med de aktuelle protokollene ≪High-speed Ethernet server≫ og ≪Ethernet server≫(2.1.9 Ethernet Server).

Get joblist request		
Måling nummer:	Ethernet Server	High-Speed ES
1	295ms	174ms
2	334ms	105ms
3	365ms	159ms
4	333ms	170ms
5	416ms	170ms
Gjennomsnitt:	348.6ms	155.6ms
Write integer request		
Måling nummer:	Ethernet Server	High-Speed ES
1	305ms	48ms
2	335ms	36ms
3	315ms	48ms
4	392ms	39ms
5	430ms	51ms
Gjennomsnitt:	355.4ms	44.4ms

Figure 53: Responstid robotkommunikasjon

Her ser vi at ved skriving av integer verdier, som er metoden vi bruker for å fortelle roboten at den skal jogge, er ca. 8 ganger så rask ved bruk av HSEC. Henting av jobbliste skjer på halvparten av tiden til «Ethernet server», men også andre funksjoner som er uoffisielt testet ligger på denne hastigheten i forhold til HSEC. Funksjonene som er implementert i gruppens klient:

- Lese robot status.
- Lese robot posisjon i koordinater, pulse-verdier og eksternakse.
- Hente jobbliste.
- Velge jobbliste.
- Skru på/av servo power.
- Skru på/av hold (pause).
- Skrive/lese posisjons variabler i pulse-verdier og for eksternakse.
- Skrive/lese IO data.
- Skrive/lese Integer-verdier.
- Skrive/lese Real-verdier.
- Velge mellom step/cycle/continuous utføring av jobb.
- Reset av alarmer.
- Fjerne error meldinger.

- Lese alarmer.

Med god planlegging var protokollen relativt lett å implementere i klienten, selv om den krever mye mer arbeid enn de andre. Siden gruppen ikke hadde tilgang til roboten på skolen, ble en simulator programmert i Java for å etterligne kommunikasjonsserveren. Dette gjør det mulig å teste konseptet og utbedre feil, uten å måtte reise til Kleven verft. Wireshark ble også brukt for å kontrollere at datagrammer som blir sendt er bygd opp etter spesifikasjonen, og sendes i rett rekkefølge. Etter klienten var testet mot roboten, implementerte gruppen den i applikasjonen. Resultatet ble at applikasjonen kontrollerer roboten raskt og feilfritt, som gir brukeren en god opplevelse ved tidsavhengige operasjoner. Av alle kommunikasjonsmetoder som er tatt i betraktning mener gruppen at dette er den desidert beste å bruke mot en Motoman DX100(3.1.3 DX100 kontroller).

5 DRØFTING

5.1 DRØFTING PROSJEKT

I planleggingen av prosjektet fordelte gruppen ansvarsområder til hvert av medlemmene. Der de med mest kunnskap og erfaring innen feltet fikk et overordnet ansvar for fremgangen og utføringen. Problemet med dette er at det kan føre til dårlig kunnskapfordeling, som gir dårlig læringsutbytte i det feltet man ikke er ansvarlig for. Gruppen løste dette ved å jobbe på samme sted, og ved å involvere hverandre i sitt arbeid. Ved å involvere alle, samtidig som en hadde det overordnede ansvaret, ble ny kunnskap tilegnet alle medlemmene, samtidig som vi holdt god fremgang.
Gruppen la hele tiden vekt på at det var minste kravet til Kleven som først måtte oppnås, før eventuelle tilleggsfunksjoner kunne implementeres. Det viktigste var å få til en applikasjon som kunne: jogge roboten, starte jobber og sende tall verdier. Etter dette var på plass, ble funksjoner som database støtte, back-end applikasjon, sende og lese posisjoner implementert.

For å kunne utnytte gruppens arbeidskapasitet, ble applikasjonen kopiert og fordelt mellom medlemmene via Dropbox. Ved at alle medlemmene hadde sin egen kopi, var det mulig å legge til og teste nye funksjoner/endringer separat. Dette gjorde det enkelt å fikse og lokalisere potensielle feil før disse ble implementert i hovedapplikasjonen.
Etter som gruppen kun har vært i besittelse av en Android enhet, ble det benyttet en Android Emulator fra Android Studio for testing. Selv om emulatoren til tider var treg og noe ustabil, var det fortsatt en lettvint måte å øke effektiviteten i prosjektet.

Fordi applikasjonen ble bygget opp etter "Objekt Orientert Programmering" prinsippet, var det relativt enkelt å legge til klasser og endre kode uten at dette førte til feil i andre klasser. Dette er kjent som "loose connection" og er en viktig faktor for å gjøre koden oversiktlig og lett håndterbar.

For å utvikle en applikasjon som kunne bli verdsatt av brukerne, var det viktig for gruppen å ta hensyn til brukerens behov og ønsker. For å tilfredsstille arbeiderne som skal bruke applikasjonen til daglig, har gruppen hatt en samtale med flere av de ansatte. Ved å prate direkte

med arbeiderne har gruppen fått førstehånds-informasjon om hvordan de ser for seg den nye løsningen, og hvilke behov de har. Det var også viktig å få vite hvordan en jobb blir utført til vanlig med tanke på å utvikle en funksjonell applikasjon.
Gruppen hadde sett for seg å ha individuelle intervju med operatørene. Da operatørene følte de ikke hadde så mye å komme med, endte det i stedet opp som en uformell samtale med en gruppe operatører. Gruppen fikk ikke så mye ut av denne samtalen hva angår design, men fikk god innsikt i hvordan jobber blir utført i dag. Dette er nødvendigvis ikke så relevant for hvordan applikasjonen skal utføre jobber i fremtiden, men likevel greit å ha i bakhodet.

5.2 ASA Robot Controller

Det ble laget mange forskjellige utkast av hvordan designet kunne se ut. Vi endte først opp med designet vist i figur 33, men visste det kom til å komme mange ideer om forbedringer jo mer og lenger gruppen jobbet med prosjektet. Det er selvfølgelig mange mulige måter å løse problemstillingen med designet på, og la dermed med i tidsskjemaet at siste utkast av designet skulle bli laget i midten av semesteret.

Når vi hadde kommet så langt at siste utkast skulle bli laget, var allerede den gamle applikasjonen begynt å ta form, og fungere i mindre grad. Men gruppen hadde begynt å få gode idéer til forbedringer og ville dermed forandre applikasjonen helt forfra. På grunn av funksjonalitet ville det være mer fornuftig at applikasjonen skulle ha et renere grensesnitt med mindre detaljer og heller gjøre det enkelt og ha en trinnvis gjennomgang av applikasjonen. Det vil da være selvforklarende og vanskelig å gjøre feil på vei igjennom applikasjonen.
Dette følte gruppen ble oppfylt i det endelige designet(Figure 34), på grunn av det rene designet og den gode programflyten. Det ene målet med å lage et nytt grensesnitt var at det skulle være mer dynamisk, både med tanke på å kunne bruke det på andre nettbrett, men også på tanke på videre utviding. For eksempel at param eterlisten oppdaterer seg etter hvor mange mulige parameter som er definert fra robot/database.

5.3 ASA Database Manager

Behovet for en back-end applikasjon kom frem etter et møte med oppdragsgiver og veileder. Det kom frem at en enkel og grei applikasjon som kun har de nødvendige funksjonene for databehandling var noe oppdragsgiver var interessert i. Det ble vurdert om det var mulig å bruke en av de mange alternativene som Navicat, SQLYog eller phpmyadmin, men ble valgt bort på grunn av for mange unødvendige funksjoner. Det ble derfor laget en applikasjon i Java som oppfylte behovet for Android applikasjonen og databasen. De ønskede funksjonene var; Legge til jobber, legge dagens jobber og fjerne jobber fra databasen, og hente ut en liste over utførte jobber.

5.4 Robot kommunikasjon

For å løse dette prosjektet har vi sett på mange forskjellige metoder å kommunisere med roboten på. Ved valg av metode ble det gjort en grundig prosess, der alle protokoller ble prøvd ut ved å programmere klienter som ble testet opp i mot roboten. Den metoden vi valgte å bruke

er «High-Speed Ethernet Client» (3.6.1.4 HSEC), grunnlaget for dette valget er de påfølgende vurderingene:

Modbus og PLS:
En Modbus klient ble programmert i Java og testet på nettbrettet. I denne testen ble det kun sett om responstiden fra vi trykket på knappen, til verdien på PLSen endret seg var tilfredsstillende rask for å få en god opplevelse ved jogging av robot. Siden denne metoden har så begrenset med funksjonalitet, er den ikke tilstrekkelig for å utføre alle funksjoner applikasjonen er laget for. Men kan eventuelt brukes i kombinasjon med en annen metode, der denne står for jogging.

Ethernet Client:
Denne protokollen har alle nødvendige funksjoner og klienten fungerte bra. Den var også lett å programmere, og implementere i programmet. Det som holder den tilbake er at den ikke er like rask som MotoNIS og HSEC.

MotoNIS:
For å ta i bruk denne protokollen ble det autogenerert en Java klient, den er rask og har alle nødvendige funksjoner. Den virket bra fra PC til robot, men denne klienten virket ikke på nettbrettet fordi Android ikke støtter alle nødvendige bibliotek. Det er kanskje mulig å gå rundt dette problemet, ved å kode denne klienten manuelt med andre bibliotek som er støttet. Men dette er mye arbeid for noe som gruppen ikke var helt sikker på kom til å virke. Det ble sett på som en stor risiko, og derfor ble ikke denne protokollen tatt i bruk.

High-Speed Ethernet Client:
Denne klienten var mye arbeid å programmere, men fordelene ble sett på som så store at risikoen ble tatt. HSEC er den raskeste(Figure 53) av alle de metodene som ble vurdert, og har alle funksjoner som applikasjonen trenger. Det var også en anbefaling fra Yaskawa, at denne metoden ble brukt i stedet for Ethernet Client.

5.5 Database

Det ble valgt å bruke et PHP-skript kjørende på en egen server, kontra bruk av JDBC. Dette ble valgt på grunn av anbefalinger fra andre utviklere på diverse forum. Gruppen skulle gjerne hatt mer tid til å sette seg inn i forskjellene mellom disse og gjort en vurdering basert på dette. Applikasjonen og serveren er ment for å være på samme lokale nett, og det er derfor ikke lagt vekt på sikkerhet mot omverdenen. Dette er en av tingene som vil være viktig å ta tak i om applikasjonen skal utvikles videre.

Bruk av SQL database var nytt for gruppen, men var ikke så vanskelig å sette seg inn i. En av gruppemedlemmene hadde allerede en database som var satt opp. Denne databasen hadde en webserver som brukte MySQL, og ble utgangspunktet for gruppen ved implementeringen og programmeringen av PHP-skriptene.

Det var lenge et problem at gruppen ikke hadde fått mulighet til å sette opp databasen på nettverket til Kleven. Dette ble løst ved å installere en Raspberry Pi i robotskapet som fungerer som database og webserver. Dette vil være en midlertidig løsning for at gruppen skal kunne løse prosjektet uten problemer på grunn av internett tilgang.

5.6 Videre utvikling

Ved videre utvikling av systemet i sin helhet, ville det vært veldig fornuftig om dette systemet kunne blitt brukt på flere roboter. Det var i utgangspunktet planlagt å se på muligheten til å sette opp kommunikasjon mot flere typer roboter som høgskolen hadde tilgjengelig. Dette ble nedprioritert på grunn av problemer med oppsett av kommunikasjon på roboten hos Kleven. Det å sette opp kommunikasjon mot flere typer roboter vil gjøre applikasjonen mer allsidig. Dette kan bli gjort ved å lage kommunikasjons-klasser som ≪HSEC≫, bare med nye protokoller for andre roboter typer (Kuka, ABB, og lignende).

Operatørene hos Kleven hadde også noen ønsker om metoder som ikke har blitt implementert, men er fullt mulig å implementere. Noen av de ble ikke implementert fordi de går i mot framtidsvisjonen for applikasjonen, som gjør at de ikke blir relevante i fremtiden. Mens andre har blitt lagt til rette for i applikasjonen. Det gjør det lett for videre utvikling av de ønskede metoder som ≪Joystick-jogging≫ og ≪Shift On Shift Off≫ som forskyver en jobb i en gitt retning.

En mulig forbedring til systemet vil være at databasen inneholder jobb filene til roboten, slik at den kan overføres hvis roboten ikke har jobben. Men med implementering av denne funksjonen, er det også nødvendig å definere hvilke roboter som kan kjøre jobben. Slik at en sjekk kan utføres før overføring av en jobb som kanskje ikke fungerer på roboten.
En annen mulighet er å gjøre applikasjonen web-basert, som vil gjøre den mye mer allsidig, og mulig å bruke på de fleste plattformer som nettbrett, iPad, pc og alle enheter med en nettleser.

5.7 Valg av IDE

Det finnes minst 5 mulige valg av IDE'er ved programmering av Android applikasjoner. Men for oss stod valget mellom Android Studio og Eclipse med Android SDK. Det ble satt igang et søk på internett om hvilken IDE som var best å bruke for vårt behov. Vi fant både fordeler og ulemper med begge IDE'ene. Men fant også ut at applikasjonsutviklere syntes de var veldig like hverandre, og mente de var gode IDE'er på hver sin måte[51]. Eclipse var inntil 2013 den offisielle Android programmeringsplattformen, og er fortsatt fullt funksjonelt. Fordeler med denne IDE'en er:

- meget utbredt
- til å stole på
- tilbyr flere moment
- visuelt layout skaper
- god oversikt i prosjekt mappe
- rask kompilering
- bra for nybegynnere

Mens ulempene ved bruk av Eclipse med Android SDK er:

- Android SDK må installeres ved siden av IDE'en
- ukjente krasj og feilmeldinger
- vanskeligheter ved design av applikasjon i xml fil

På den andre siden har vi Android Studio som ble lansert i 2013, og er "nytt", men er laget spesielt for utvikling av Android applikasjoner. Det er også blitt den offisielle programmeringsplattformen for Android. Fordeler ved bruk av denne IDE'en er[52]:

- god visuelt layout skaper
- rask IDE
- optimalisert for error behandling

Mens ulempene ved bruk av Android Studio er:

- mindre kilder til hjelp
- vanskelig å bruke for nybegynnere
- mindre oversikt av prosjektet
- vanskelig å installere

Etter å ha diskutert mulighetene bestemte gruppen seg for å bruke Eclipse med Android SDK, på grunn av kjennskap til plattformen og siden denne IDE'en er god for nybegynnere med mindre kunnskap innen Android programmering. Tatt i betrakting at kildene for fordelene og ulempene er gamle, er nok Android Studio det beste valget, men gruppen tok ikke sjansen på å bruke lenger tid på å lære seg en ny plattform istedenfor å bruke en plattform som de allerede hadde kjennskap til.

6 KONKLUSJON

Det endelige designet og utformingen av applikasjonen er enkelt, intuitivt og gruppen føler målet er nådd i henhold til kravspesifikasjon. I tillegg til kravene og ønskene fra oppdragsgiver, er det vedlagt flere funksjoner som forhåpentligvis vil komme godt med. Gruppen startet prosjektet med lite, til ingen kunnskap i å programmere en Android applikasjon. Dette førte til at mye av tiden i starten gikk med til å lære seg basis kunnskapen som er nødvendig. Gjennom prosjektet har gruppen prøvd og feilet, men står igjen med det vi selv velger å kalle god kunnskap om Android utvikling.

Valget med å holde oss til Eclipse fremfor å lære oss Android Studio, føler vi var et godt valg.

Gruppen testet Android Studio i en liten periode, men følte ikke det ga noen fordel over Eclipse, men heller førte til at det gikk tregere.

Det var mer omfattende å få opp kommunikasjonen mot roboten enn det gruppen hadde sett for seg. Dette førte til at det ble testet mange forskjellige metoder og protokoller. Metodene som er prøvd er blitt dokumentert i denne rapporten og har gitt stort læringsutbytte for gruppens medlemmer, samtidig som andre kan ha nytte av denne informasjonen. Den endelige løsningen ble å bruke HSEC, som viste seg å oppfylle alle behov gruppen hadde.

Gruppen endte opp med å bruke et PHP-skript som kjører på en webserver. Gruppen kan ikke stå for at dette er den beste metoden, hva angår sikkerhet eller kompleksitet, men det oppfyller applikasjonens behov. Som database-administrasjonssystem ble det laget løsninger for både MySQL og MSSQL. Fordi et PHP-skript tar seg av formidlingen mellom applikasjonen og databasen, vil det ikke være nødvendig å gjøre endringer i applikasjonen selv om valg av database-system skulle endres. Dette er en fin løsning som gjør applikasjonen mer tilpasningsdyktig.

For at applikasjonen skal være mest mulig brukervennlig var det viktig for gruppen å lage en løsning som la til rette for å enkelt styre informasjonen som skal ligge i databasen. Ettersom applikasjonen dekker "front-end" var det viktig for gruppen å også tilby en "back-end" for å kunne tilby et komplett system. Det er mange programmer tilgjengelig for styre databaser fra datamaskiner, men gruppen følte dette kunne gjøres på en enklere måte og i tillegg ved å lage det selv, kunne gruppen legge til rette så det passer perfekt til systemet.
Denne ble skrevet i Java da dette er språket gruppen behersker best og gjorde at det raskest mulig kunne få på plass en løsning. Gruppen er klar over at det finnes mange andre løsninger som kunne blitt brukt. Blant annet kunne det vært brukt en løsning som kjører rett ifra nettleser. Gruppen føler til tross for dette at vi kom opp med en god løsning, som vil gjøre det lettere å håndtere databasen.

Totalt sett er gruppen veldig fornøyd med prosjektet. Gruppen var klar over at ting ofte tar lenger tid enn planlagt, og at man stort sett alltid møter på uforutsette problemer og hendelser. Basert på dette satt gruppen opp en tidsplan som tok hensyn til dette, som igjen førte til at tidsplanen ble holdt gjennom hele prosjektet. Gruppen er godt kjent med hverandre etter å ha gjennomført flere store prosjekter sammen ved tidligere tidspunkt. Dette kom godt med da man kjenner til hverandres styrker og svakheter. Med hensyn til den relativt korte tiden gruppen har hatt på å gjøre dette prosjektet, føler gruppen at de har kommet opp med et godt produkt. Det vil selvfølgelig være ting som kunne vært gjort bedre, og andre løsninger som kunne vært brukt. Dette er stort sett alltid tilfelle når man ser tilbake på hva som er gjort. Men vi føler med dette at vi har laget et produkt som oppfyller oppdragsgivers ønsker, og som kan danne et godt grunnlag for videre utvikling.

Vi er godt fornøyd med hjelpen vi har fått av veiledere og samarbeidet med oppdragsgiver. Prosjektet har gitt godt utbytte i form av ny kunnskap, utføring av større prosjekter og samarbeid med oppdragsgivere. Gruppen har også sett hvor viktig det er å legge en god plan før man setter igang med prosjektet.

7 REFERANSER

References

[1] Dagens Næringsliv av Bjørn Segrov, sist modifisert:24.10.2014 http://www.dn.no/nyheter/naringsliv/2014/10/24/2149/Industri/kleven-p-diamantjakt *YASKAWA Motoman robot.* Automated Welding Applications, Hentet: 17. April 2015

[2] Yaskawa, sist modifisert: www.motoman.com/applications/robotic_arc_welding.php *YASKAWA Motoman robot.* Automated Welding Applications

[3] YASKAWA, sist modifisert: 2013 http://www.motoman.co.uk/en/products/robot-controllers/product-view/?tx_catalogbasic_pi1%5Buid%5D=282&cHash=28bd624477e92d63628669a743e82f1e *YASKAWA Motoman robot.* Automated Welding Applications

[4] Johnsen, Ragnar. (2009, 15. juni), I Store norske leksikon. https://snl.no/kommunikasjonsprotokoll *Kommunikasjonsprotokoll.* Hentet 1. mai 2015.

[5] Wikipedia, sist modifisert: 20. mai 2015 http://en.wikipedia.org/wiki/Client%E2%80%93server_model *Klient-server kommunikasjon.* Hentet 20. mai 2015.

[6] Wikipedia, 22 April 2015. http://en.wikipedia.org/wiki/SOAP *Simple Object Access Protocol.* Hentet 10. mai 2015.

[7] YASKAWA. *HSEC.pdf High-speed ethernet server function manual.* Hentet 30. februar 2015

[8] YASKAWA, sist modifisert: 21. desember 2012 *MotoNIS Users Manual.pdf MotoNIS manual.* Hentet 09. februar 2015

[9] Wikipedia, sist modifisert: 2015 04-13. http://no.wikipedia.org/wiki/Programmerbar_logisk_styring *For ethernet server function.* Hentet 15. Mai 2015

[10] YASKAWA, sist modifisert: 2009 09-10. *MotomanEthernetServerFunction.pdf Ethernet server function Manual.* Hentet 24. februar 2015

[11] Modbus. http://www.modbus.org/specs.php *Modbus.* Hentet 15. mai 2015

[12] Wikipedia, sist modifisert: 12.mai 2015. http://en.wikipedia.org/wiki/Modbus *Modbus.* Hentet 15. mai 2015

[13] Wikipedia, IEEE 802.3 Standard for Ethernet, sist modifisert: 6 May 2015, http://en.wikipedia.org/wiki/Ethernet *Ethernet standards* Hentet

[14] IEEE,24 June 2013, http://standards.ieee.org/news/2013/802.3_30anniv.html *IEEE 802.3 standard for ethernet.* Hentet 8. Mai 2015.

[15] sist modifisert: 7 May 2015, http://en.wikipedia.org/wiki/Transmission_Control_Protocol *Transmission Control Protocol.* Hentet 9 Mai 2015.

[16] Wikipedia, http://en.wikipedia.org/wiki/User_Datagram_Protocol *User Datagram Protocol*. 7 May 2015

[17] Java Oracle, http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html *About Java*. Hentet 5. Mai 2015

[18] Android official, http://developer.android.com/about/index.html *Android OS*. Hentet 11. Mai 2015.

[19] Android official, http://developer.android.com/reference/android/widget/AdapterView.html *Android adapterView*. Hentet 11. Mai 2015.

[20] Android official, http://developer.android.com/reference/android/widget/Adapter.html *Android adapter*. Hentet 11. Mai 2015.

[21] Android official, http://developer.android.com/guide/topics/manifest/manifest-intro.html *Android Manifest*. Hentet 18. Mai 2015.

[22] The Eclipse Foundation, 2015 http://www.eclipse.org/org/ *About Eclipse*. Hentet 8. Mai 2015

[23] 2015, Oracle Corporation, https://netbeans.org/about/index.html *About NetBeans*. Hentet 8. Mai 2015

[24] Android official, http://developer.android.com/tools/studio/index.html *About Android Studio*. Hentet 9. Mai 2015

[25] Wikipedia, sist modifisert: 01 May 2015 http://en.wikipedia.org/wiki/Android_Studio *About Android Studio*. Hentet 9. Mai 2015

[26] Android official,07 May 2015. http://developer.android.com/reference/android/app/Activity.html *Activity metode*. Hentet 8. Mai 2015

[27] Android official, http://developer.android.com/guide/components/fragments.html *Fragment*. Hentet 6. Mai 2015

[28] Developer.Xamarine http://developer.xamarin.com/guides/android/platform_features/fragments/part_1_-_creating_a_fragment/ *Fragment Lifecycle*

[29] JavaWorld by David Geary, 25 april 2003. http://www.javaworld.com/article/2073352/core-java/simply-singleton.html *Singleton*. Hentet 16. Mai 2015

[30] Android Official, 19 April 2014 http://developer.android.com/reference/android/os/Bundle.html *Bundle metoden*. Hentet 29. April 2015

[31] Android Official, 13 Mai 2015. http://developer.android.com/reference/android/view/View.html *View*. Hentet 16. Mai 2015

[32] Oracle. http://docs.oracle.com/javase/tutorial/essential/concurrency/ *Tråder*. Hentet 16. Mai 2015

[33] Android Official, 13. Mai 2015. http://developer.android.com/guide/components/processes-and-threads.html#Threads *Tråder*. Hentet 16. Mai 2015

[34] Oracle. http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html *Tråd states*. Hentet 16. Mai 2015

[35] Android Official, 13 Mai 2015. http://developer.android.com/reference/android/os/AsyncTask.html *Tråder*. Hentet 18. Mai 2015

[36] Android Official, 13 Mai 2015. http://developer.android.com/reference/java/util/Timer.html *Timer*. Hentet 16. Mai 2015

[37] Android Official, 13 Mai 2015. http://developer.android.com/reference/java/util/TimerTask.html *TimerTask*. Hentet 16. Mai 2015

[38] Android Open tutorials, 7 April 2014. http://androidopentutorials.com/android-sharedpreferences-tutorial-and-example/ *SharedPreferences*. Hentet 16. Mai 2015

[39] Rossen, Eirik. (2009, 14. februar). Database: IT. I Store norske leksikon. https://snl.no/database/IT *Info om databaser*. Hentet 6. April 2015

[40] Raspberry Pi, https://www.raspberrypi.org *About Java*. Hentet 12. Mai 2015

[41] Property of Quinstreet Enterprise, http://www.sqlcourse.com/intro.html *What is SQL?*. Hentet 20. April 2015

[42] Microsoft Official, http://www.microsoft.com/en-us/server-cloud/products/sql-server-editions/overview.aspx *What is SQL?*. Hentet 16. Mai 2015

[43] M6, http://blog.m6.net/rival-of-the-titans-mysql-vs-sql-server/ *Microsoft SQL vs. MySQL*. Hentet 16. Mai 2015

[44] MySQL, http://www.mysql.com/about/ *About SQLite*. Hentet 24. April 2015

[45] SQLite, https://www.sqlite.org/about.html *About SQLite*. Hentet 10. Mai 2015

[46] JDBC, http://www.oracle.com/technetwork/java/javase/jdbc/index.html *Java Database Connectivity*. Hentet 24. Mars 2015

[47] Rossen, Eirik. (2009, 14. februar). https://snl.no/objektorientert_programmering%2FIT *Objektorientert Programmering: IT. I Store norske leksikon.*. Hentet 15. April 2015

[48] PHP, http://php.net/ *Info om databaser*. Hentet 15. April 2015

[49] Raspberry Pi, https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md *Webserver for Raspberry Pi*. Hentet 13. Mai 2015

[50] Adam, Raspberry Pi, http://www.raspberry-projects.com/pi/software_utilities/mysql *About Java*. Hentet 12. Mai 2015

[51] Stackoverflow, sist modifisert: 8 september 2013. http://stackoverflow.com/questions/17849078/which-android-ide-is-better-android-studio-or-eclipse *Android Studio VS eclipse*. Hentet 14. Mai 2015

[52] Halfapp, sist modifisert: 15 desember 2013. http://halfapp.com/blog/eclipse-vs-android-studio-battle-android-ides/ *Tales from a newbie developer*. Hentet 14. Mai 2015

[53] Samsung official. http://www.samsung.com/no/tab-s/?gclid=Cj0KEQjwyoCrBRCl-aa97pKX_t8BEiQAbrs_9Ag0DcNbZMdWxXcGhEUkMNWIcnR4ZS0T7UNlweU-rqUaAi8s8P8HA *Samsung Galaxy Tab S*. Hentet 23. Mai 2015

[54] YASKAWA. http://www.motoman.com/datasheets/DX100%20Controller.pdf *DX100 controller*. Hentet 15. Mai 2015

[55] Draw.io(tegne program) https://www.draw.io/

[56] Zerokol, sist modifisert: 25 september 2014. https://github.com/zerokol/JoystickView *JoystickView*.

[57] Android Official. http://developer.android.com/reference/android/app/AlertDialog.Builder.html *AlertDialog.Builder*.

[58] Vasudevan, Nithya. (2012, 8. September) http://theopentutorials.com/tutorials/android/listview/android-custom-listview-with-image-and-text-using-baseadapter/ *Android: Custom ListView with Image and Text using BaseAdapter*

[59] www.Developer.Android.com http://developer.android.com/guide/topics/manifest/uses-sdk-element.html *¡USES-SDK¿*

[60] Eclipse official. https://eclipse.org/windowbuilder/ *WindowBuilder for Back-end*

[61] Oracle Java official. http://docs.oracle.com/javase/7/docs/api/javax/swing/event/ListSelectionListener.html *ListSelectionListener*

[62] Developer.Android http://developer.android.com/reference/android/os/NetworkOnMainThreadException.html *NetWork on mainthread exeption*

[63] Developer.Android http://developer.android.com/guide/topics/data/data-storage.html *Database. Storage options.*

[64] Yidong, Fang https://code.google.com/p/json-simple/

[65] Apache Commons https://commons.apache.org/ *Array Handling.*

[66] Jamod http://jamod.sourceforge.net/ *Java Modbus implementation*

[67] Teknisk ukeblad, 2014 http://www.tu.no/it/2014/07/03/dette-er-verdens-10-mest-populare-programmeringssprak *Dette er verdens 10 mest populære programmeringsspråk* Hentet 28.05.2015

List of Figures

8 VEDLEGG

Vedlagt i rapport:

Vedlegg 1 Forprosjekt
Vedlegg 2 Klasse diagram ASA RC
Vedlegg 3 Klasse diagram ASA DBM
Vedlegg 4 Brukermanual ASA RC
Vedlegg 5 Brukermanual ASA DBM
Vedlegg 6 Møtereferat 1 til 6
Vedlegg 7 Fremdriftsrapport 1 til 6 + Gantt

I tillegg til vedlegg nevnt over, er lagt ved på minnepenn:
Vedlegg 8 Kildekode
Vedlegg 9 HSEC.pdf
Vedlegg 10 MotomanEthernetServerFunction.pdf
Vedlegg 11 MotoNIS Users Manual.pdf
Vedlegg 12 DX100 E1102000034SE01 MANUAL.pdf
Vedlegg 13 JOGG.JBI Jogge program for robot
Vedlegg 14 Plakat
Vedlegg 15 Presentasjon
Vedlegg 16 JavaDoc

Vedlegg 1

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

HØGSKOLEN I ÅLESUND

Aalesund University College

TITTEL:
Kleven Robotics Controller

KANDIDATNUMMER(E):
Anders Kobbevik, Simon Langlo, Audun Westerskow

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
30.01.15	**IE303612**	**Bacheloroppgave**	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
AUTOMASJON	13/1	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Kleven Verft AS

OPPGAVE/SAMMENDRAG:

I sveisehallen hos Kleven Verft har de flere flyttbare roboter som brukes til å sveise sammen seksjoner. Disse er i dag styrt av en MOTOMAN DX100 kontroller. Gjennom flittig bruk av kontrolleren har Kleven hatt problem med at kablene mellom kontroller og styringsskap ofte tar skade og må byttes ut. Dette er en dyr utgift for Kleven og setter roboten ut av drift i reparasjonsperioden.

Som en løsning på dette problemet ser Kleven for seg en løsning hvor operatøren kan styre roboten trådløst via et nettbrett. Ved å erstatte den kompliserte robot kontrolleren, med et nettbrett. Nettbrettet skal ha et enkelt og intuitivt grensesnitt, med funksjoner som gjør jobben for operatøren lettere og raskere. Dette vil da redusere behovet for opplæring og vil øke effektiviteten i produksjonen.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved Høgskolen i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

1 INNLEDNING

Ved valg av oppgave, ble det sett etter en oppgave som var utfordrende og lærerik for alle i gruppen. En i gruppen har hatt kontakt med Kleven Verft som var villig til å lage en oppgave som var automasjonsrelatert og som begge parter tar nytte av.

Oppgaven går ut på å erstatte en komplisert robot kontroller med et nettbrett. Nettbrettet skal ha et enkelt og intuitivt grensesnitt, med funksjoner som gjør jobben til operatøren lettere og raskere. Dette vil da redusere behovet for opplæring og øke effektiviteten i produksjonen. Det vil også hindre at kabelen til robot kontrolleren går i stykker, som er et stort problem i dag.

Bakgrunnen for valg av oppgave, var at gruppen ville ha en oppgave gitt av en bedrift utenfor skolen, hvor produktet av oppgaven vil bli tatt i bruk i produksjonen ved fullføring av prosjektet. Det at produktet skal bli brukt i senere anledning er en ekstra motivasjon for å få til et ordentlig produkt.

Hensikten med prosjektet er å automatisere og øke effektiviteten i produksjonen i sveisehallen hos Kleven Verft.

Problemstillingene gruppen står overfor er kommunikasjon via robot og nettbrett, lage en android applikasjon, som i henhold til kravene fra Kleven er enkelt og intuitivt. Gruppen vil lage applikasjonen mest mulig generell, så det er mulig å bruke den på flere forskjellige roboter, med lignende behov som den det er utviklet til.

2 BEGREPER

Motoman DX100 – er robot kontrolleren som er brukt til å programmere roboten.
Android – Operativsystem for mobiler og nettbrett
PLS – Programmerbar logisk styring
Buss – Kommunikasjonssystem for utveksling av data i.e: modbus, profibus

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
120409
120150
120140

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget ID 302906

3.1.1 Oppgaver for prosjektgruppen – organisering

Prosjektgruppen består av tre personer og har fått tilegnet rolle som prosjektleder, sekretær og øvrig medlem.

3.1.2 Oppgaver for prosjektleder

Prosjektlederens ansvarsområde er å oppdatere framgangsrapport og planlegge møter med både veiledere og oppdragsgiver. Prosjektleder har ansvar for å organisere og tilpasse oppgaver for deltakerne i prosjektet, og for å se til at prosjektmålet oppnås etter kravene til prosjektet. Prosjektleder skal være oppmerksom på at oppgavene som er fordelt, blir utført som planlagt.

3.1.3 Oppgaver for sekretær

Hovedansvaret for sekretæren er å bidra til at gruppen fungerer effektivt og bra sammen. Sekretærens ansvarsområde er å føre møtereferat for hvert møte med veiledere og oppdragsgiver. Holde styr på at dokumentasjon er oppdatert.

3.1.4 Oppgaver for øvrig medlem

Pålegges oppgaver fra prosjektleder og sekretær. Har ansvar for å bedømme sakene og komme med konstruktiv kritikk til avgjørelsene fra prosjektleder og sekretær om nødvendig.

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Veileder - Hans Støle, høgskolelektor ved høgskolen i Ålesund

Veileder - Ottar Osen, høgskolelektor ved høgskulen i Ålesund

Kontaktperson – Simon Sundal, automasjonsingeniør ved Kleven Verft

4 AVTALER

4.1 Avtale med oppdragsgiver

Gruppen har inngått et samarbeid med Kleven Maritime AS om utføring av bachelor oppgave.

Oppdragsgiver vil stille til rådighet de ressurser gruppen trenger for å gjennomføre prosjektet. Det har også blitt utnevnt en kontaktperson som gruppen kan forholde seg til gjennom prosjektet og som vil legge til rette for utføring av oppgaven. Oppdragsgiver har påtatt seg ansvaret for å lage en liste over kommandoene til roboten, så dette ligger til rette for gruppen. Ved behov for spørsmål er kontaktperson tilgjengelig på telefon eller email hver dag.

4.2 Arbeidssted og ressurser

Ved behov vil Kleven Verft stille med arbeidsplasser, ellers vil mye av tiden bli brukt ved høgskolen i Ålesund. Det er ønskelig å unngå stopp i produksjonen, så gruppen vil hovedsakelig bruke roboter tilgjengelig på skolen for testing. Ved behov vil derimot Kleven stille sine roboter til disposisjon.

- Gruppen har gjennom høgskolen i Ålesund tilgang til arbeidsrom, samt roboter for utføring av tester.
- Gruppen har fått tildelt 2 veiledere fra høgskolen i Ålesund samt en kontaktperson ved Kleven Verft, alle med relevant kunnskap for prosjektet.
- Gruppen stiller med egne datamaskiner. Programvare som ikke tilbys via høgskolen eller som gratisvare, vil bli anskaffet via Kleven.

Ressurser som trengs for prosjektet blir dekt av Kleven. Dette inkluderer:

- Nettbrett(Samsung Tab S)
- MicroSD kort
- PLS
- Trådløs Router

- Buss-kort til robot

Kontaktperson fra Kleven Verft, er Simon Sundal.

4.3 Gruppenormer – samarbeidsregler – holdninger

Gruppen har jobbet sammen ved flere tidligere anledninger, og har dermed god kjennskap til hverandre og deres arbeidsrutiner.
For å fullføre prosjektet og for å oppnå målene innad i gruppen, er det viktig med god kommunikasjon og godt samarbeid.

Med dette har gruppen satt noen samarbeidsregler som gruppen syns er viktig å forholde seg til:
Gruppen er enig om at avgjørelser skal fattes i fellesskap, og at det skal være enighet før en beslutning tas. Arbeidsoppgaver skal deles likt, og det å komme for sent skal unngås.

Hovedsakelig skal gruppen sitte samlet når arbeid på og med prosjektet pågår, men det gis mulighet til å ta enkelt dager på andre steder. Ved arbeid borte fra gruppen skal dette avtales på forhånd, og de resterende på gruppen skal bli oppdatert på arbeide som har blitt utført.

Arbeidstid vil være mellom 8.15 og 16.00, men noen lenger dager må påregnes. Ved sykdom eller andre grunner for fravær må varsles til gruppens øvrige medlemmer. Dette gjelder også for å komme for sent.

Gruppen ønsker å bevise for seg selv, høgskolen og oppdragsgiver at de er en seriøs og målrettet gruppe. For å vise dette legges det stor vekt på å oppføre seg profesjonelt, komme tidsnok og holde avtaler. Gjennom å legge dette til grunn, ser gruppen for seg å kunne ha et godt og profesjonelt samarbeid med oppdragsgiver, samt høgskolen i Ålesund.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

I sveisehallen hos Kleven Verft har de flere flyttbare roboter som brukes til å sveise sammen seksjoner. Disse er i dag styrt av en MOTOMAN DX100 kontroller. Gjennom flittig bruk av kontrolleren har Kleven hatt problem med at kablene mellom kontroller og styringsskap ofte tar skade og må byttes ut. Dette er en dyr utgift for Kleven og setter roboten ut av drift i reparasjonsperioden.

Som en løsning på dette problemet ser Kleven for seg en løsning hvor operatøren kan styre roboten trådløst via et nettbrett.

Målet er å gjøre det mulig for operatøren å bruke roboten uten å måtte bruke kontrolleren. Dette vil gjøre det mulig for operatøren å kjøre ferdige programmer og styre roboten manuelt med nettbrett. Denne løsningen vil redusere bruk av kontrolleren til et minimum, noe som vil redusere skade på kabel og forhåpentligvis øke effektiviteten i produksjonen.

For å oppnå dette må gruppen:

- Designe og programmere et brukergrensesnitt som kan bli kjørt fra et nettbrett og som oppfyller oppdragsgivers krav.
- Sette opp en trådløs kommunikasjon mellom nettbrettet og PLS.
- Sette opp kommunikasjon mellom PLS og DX100.

Ved gjennomføring av dette prosjektet får gruppen en innsikt i hvordan det er å ha en ekte bedrift som oppdragsgiver. Løsningen skal integreres med bedriftens systemer og blir tatt i bruk av dens arbeidere. Dette krever programmering opp imot eksterne systemer, og design av et praktisk og intuitivt brukergrensesnitt. Dette gir gode erfaringer som en har med seg videre ut i arbeidslivet.

Resultatmålet er å få laget et produkt som vil bli tatt i bruk i produksjonen hos Kleven Verft, hvor det i dag blir brukt en MOTOMAN DX100 kontroller, vil det etter at oppgaven er ferdig, bli brukt et nettbrett med de samme funksjonene som på kontrollen. Applikasjonen på

nettbrett vil være å så generell som mulig, for å kunne utvikle henholdsvis like system for andre roboter.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Kravet fra oppdragsgiver er å lage en applikasjon for et nettbrett som skal erstatte MOTOMAN DX100 kontrolleren. Applikasjonen skal ha ett enkelt og intuitivt grensesnitt med de funksjonene som blir brukt fra kontrolleren.
Om løsningen er god vil den bli tatt i bruk i produksjonen, når oppgaven er ferdig og levert.

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

For å gjennomføre prosjektet på best mulig måte har vi delt prosjektet inn i tre hoveddeler, der hver prosjekt deltaker har ansvar for hver sin del. Det er ansvarspersonen som passer på att deres del er i rute i henhold til Gantt diagrammet. Gruppen har jevnlige møter der det blir diskutert framgangen i hver del, slik at eventuelle justeringer kan blir gjort.

5.4 Informasjonsinnsamling – utført og planlagt

Ved innhenting av informasjon for å beskrive oppgaven, ble det presentert en video om hvor enkelt ett grensesnitt kan gjøre tidkrevende jobber mye raskere og enklere.[1] Det har også blitt samlet inn informasjon om busser og protokoller som støttes av roboten. Videre vil informasjon om robot programmering og applikasjonsdesign bli anskaffet. Dette vil gruppen oppnå ved hjelp av veileder, fremtidige kontakt personer og internettsøk.

5.5 Vurdering – analyse av risiko

Risikoer ved utføring av prosjektet:

- Sykdom
- Fravær
- Utstyrssvikt
- Uenigheter

Det er stor mulighet for at prosjektet blir gjennomført. Oppgaven kan løses på enkle og raske måter som fører til et dårlig resultat, men kan også utføres på en grundig, smart og solid måte, hvor det er muligheter for videreutvikling og gjenbruk av applikasjonen ved senere anledning.

5.6 Hovedaktiviteter i videre arbeid

Nr	**Hovedaktivitet**	**Ansvar**	**Kostnad**	**Tid/omfang**
A1	Android Applikasjon	AW		
A11	Komponent valg/bestilling		3588 ,-	
A12	Design			
A13	Java Programmering			
B1	Kommunikasjon	SL		
B11	Komponent valg/bestilling		??	
B12	Tablet-> PLS			
B13	PLS -> Robot			
C1	Robot	AK		
C11	Komponent valg/bestilling		??	
C12	Montering av Hardware			
C13	Programmering			
	SUMMERING			

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Har delt oppgaven inn i 3 hoveddeler.
Del 1: Android Applikasjon, som inneholder å programmere en applikasjon for Android ved bruk av Eclipse som holder rammene for hvordan oppdragsgiver vil ha applikasjonen.
Del 2: Kommunikasjon fra nettbrett til PLS, og fra PLS til Robot. Det første gruppen må finne ut er hvilke komponenter som bør bli brukt, deretter vil det være fornuftig å prøve seg frem med å få til kommunikasjonen.

Del 3: Robot, er først og fremst å lage programmet for det som skal skje på robotsiden. Men også montering av hardware i robotskap.

5.7.2 Styringshjelpemidler

- Gantt diagram, vist i vedlegg 1
- Dropbox

5.7.3 Utviklingshjelpemidler

- Eclipse
- Netbeans
- Dropbox
- GanttProject

5.7.4 Intern kontroll – evaluering

For å oppfølge framdrift har gruppen satt opp faste møter med veiledere hver andre uke, der det blir gjennomgang av framgangen og eventuelle utfordringer som gruppen står ovenfor. Før hvert møte blir det skrive en framdriftsrapport som inneholder gjennomførte aktiviteter denne perioden, og de oppgavene som er planlagt å bli utført neste periode. Det er også blitt laget en prosjektplan i GanttProject som viser fremgangen i prosjektet og målene som er oppnådd.(Vedlegg 1)

5.8 Beslutninger – beslutningsprosess

I starten av prosjektet ble det diskutert og tatt viktige beslutninger over for hvilken oppgave som skulle bli valgt, det ble først diskutert og formulert internt i gruppen, der etter ble det presentert overfor veilederne for å høre deres side av oppgavene.
Fra Kleven fikk gruppen presentert to potensielle oppgaver som kunne bli valgt mellom. Den ene var «nettbrett styring av robot», og den andre var «robot-plassering og –sveising av avstivere på skrogplater ved hjelp av vision». Gruppen var redd for at den første oppgaven ikke var stor nok til å være en bachelor oppgave, og valget falt da først mot vision oppgaven.

Det viste seg da etter et møte med Kleven at den siste oppgaven var for stor til å kunne utføres på så kort tid. Som et alternativ til denne oppgaven, kom gruppen frem til at en mulighet var å utføre denne oppgaven i en nedskalert versjon.
Dette kunne utføres med de to Ekornes robotene i kjelleren hos høgskolen i Ålesund, og ville vært en god simulering av den andre oppgaven i en overkommelig størrelse.
Det ble kalt inn til møte med veilederne for å diskutere bekymringene til de ulike oppgavene. Etter diskusjonen ble det konstatert at i første oppgaven var det flere utfordringer enn gruppen først hadde forestilt seg. Og at den er stor nok for en bachelor oppgave. Valget sto dermed mellom «nettbrett styring av robot» og den skalerte versjonen av «Vision» oppgaven.
Gruppen bestemte seg i mellom til slutt å velge «nettbrett styring av robot».

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

- Arbeidsplan laget i Gantt.
- Javadoc
- Brukerveiledning applikasjon
- Møtereferat

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

12.01.15 – Første møte med Kleven Verft/ Simon Sundal.
Omvisning i produksjonen, og innsyn i de forskjellige oppgavene. Ene oppgaven var større enn først antatt. Avventer oppgavene, inntil møte med veilederne og hører deres meninger om hva som burde gjøres.

16.01.15 – Første møte med veiledere.
Presenterte 2 forskjellige oppgaver, første oppgave er bruker grensesnitt for styring

av robot fra nettbrett. Andre oppgaven er plassering og sveising av avstivere på skrogplate ved hjelp av vision og to roboter. Endte opp med oppgave 1, siden veileder konkluderte med at det var mer enn nok utfordringer innen kommunikasjon og masse arbeid med å lage et bra og funksjonelt grensesnitt.

7.1.2 Prosjektmøter

06.02.15 – Andre møte med veiledere, og oppdragsgiver.

20.02.15 – Tredje møte med veiledere.

06.03.15 – Fjerde møte med veiledere.

20.03.15 – Femte møte med veiledere.

03.04.15 – Sjette møte med veiledere.

17.04.15 – Sjuende møte med veiledere.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Framdriftsrapport blir levert før møte med veiledere, annenhver uke. Dette blir gjort for å oppdatere veiledere om hvor langt gruppen har kommet i henhold til planen og det som er planlagt for neste periode.

Det vil også bli skrive møtereferat for hvert møte med veiledere og oppdragsgiver.

Milepæler for prosjektet	
Milepæl 1	Første utkast av grafisk grensesnitt
Milepæl 2	Kommunikasjon
Milepæl 3	Android applikasjon ferdigstilt
Milepæl 4	Ferdig stilling av prosjektet

8 PLANLAGT AVVIKSBEHANDLING

Forsinkelser kan takles enten ved å droppe ekstra funksjonalitet, eller ved å arbeide overtid. Hvilket alternativ som blir valgt, vil vurderes i gruppen der det blir lagt vekt på viktighet av funksjonalitet og tid. Avvik grunnet tekniske problemer vil diskuteres i møte med veileder, der vi prøver å finne beste løsning på problemet. Hvis en endring må foretas vil dette bli rapportert til veiledere og oppdragsgiver. Det har blitt tatt hensyn til en ukes ferie i påsken, denne kan også brukes til å innarbeide forsinkelser.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

Som nevnt i kapittel 4.2 vill Kleven dekke de ressursene som trengs for å utføre prosjektet. Det som trengs til nå er:

- Nettbrett – Samsung Tab S, anskaffet
- PLS
- Buss til robot
- Trådløs Router

For at gruppen skal kunne gjennomføre prosjektet, er det en forutsetning at Kleven skaffer de nødvendige komponentene. For tidlig testing og utvikling, har gruppen tilgang til alt nødvendig utstyr på høgskolen i Ålesund.

10 REFERANSER

[1] http://vimeo.com/78353138

[2] http://www.kleven.no/

[2] Praktisk prosjektledelse, Fra idé til gevinst. Asbjørn Rolstadås, Nils Olsson, Agnar Johansen og Jan Alexander Langlo. 2014

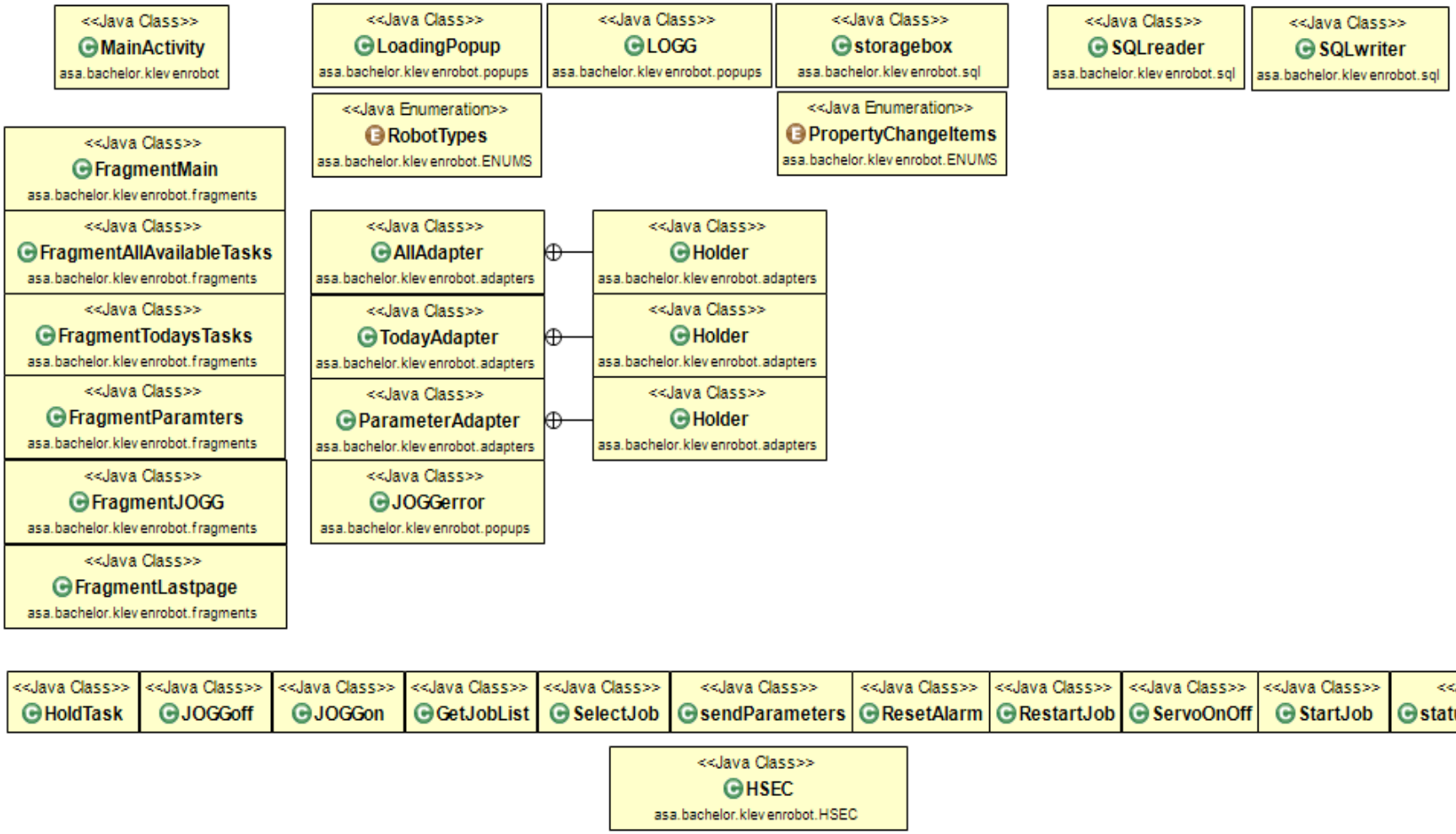
VEDLEGG

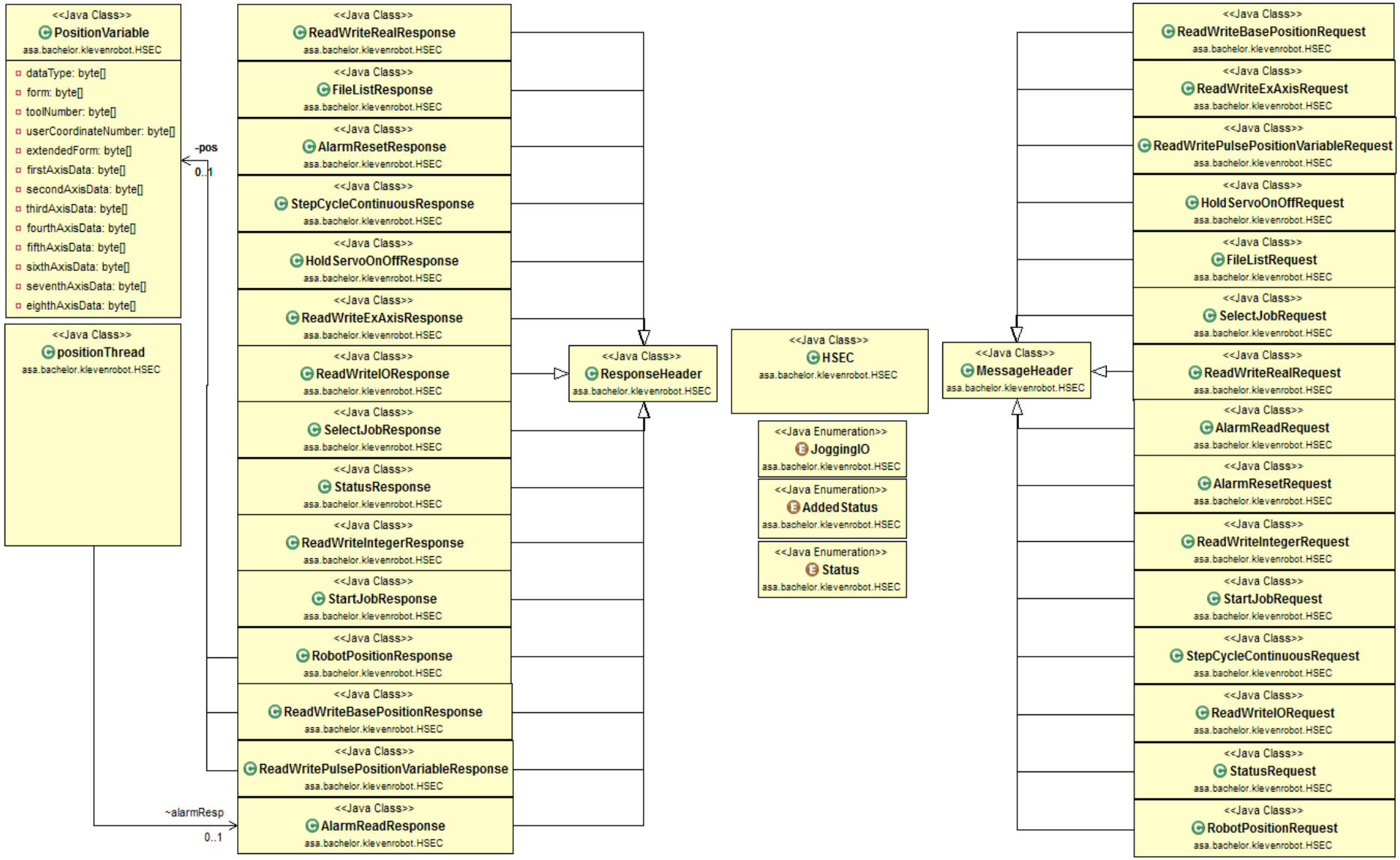
Vedlegg 1 GanttProject, Arbeidsplan for prosjektet

Vedlegg 1

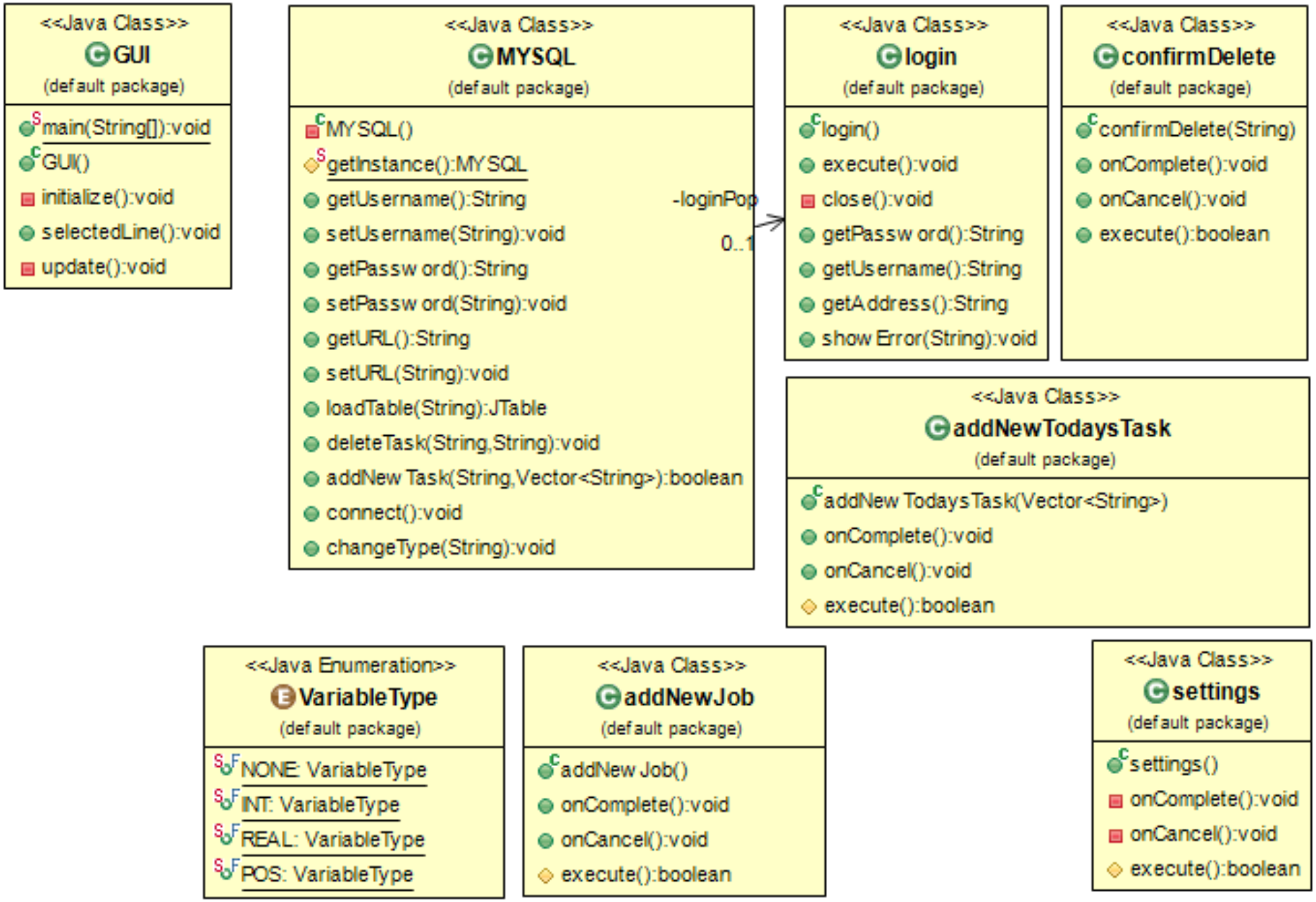
GANTT project
Name
Begin date
End date
2015
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Definere Oppgave 1/14/15 1/16/15
Forprosjekts Rapport 1/14/15 1/30/15
Komponent valg / bestilling 1/14/15 2/6/15
Utvikle Android App i java 1/14/15 4/24/15
Sette seg inn i fragments og activities 1/14/15 1/30/15
Plan for funksjoner som skal implementeres 1/26/15 2/6/15
Lage første utkast av grafisk brukergrensesnitt 1/26/15 2/13/15
Implementering av knapper og funksjoner 2/16/15 4/17/15
Finne løsninger for lesing og skriving til enhetsminne 3/2/15 3/27/15
Automatisk innlesing av "robot calls" fra dokument 3/2/15 3/13/15
Lage jobbjournal og lagre på enhetsminne 3/16/15 3/27/15
Siste utkast av grafisk brukergrensesnitt 3/30/15 4/13/15
Teste funksjoner og finne forbedringer 4/13/15 4/24/15
Kommunikasjon 2/2/15 4/17/15
Opprette java klasse for kommunikasjon mellom tablet og PLS 2/2/15 2/20/15
Sette opp PLS for kommunikasjon med nettbrett, og teste 2/16/15 2/27/15
Lage liste for IO mot robot 3/2/15 3/20/15
Sette opp PLS for kommunikasjon med robot 3/23/15 4/17/15
Sette opp robot for kommunikasjon med PLS 3/23/15 4/17/15
Robot Programering 4/7/15 4/24/15
Montering hos kleven 4/20/15 5/1/15
Bachelor rapport 2/2/15 5/8/15

Vedlegg 2


<<Java Class>>
MainActivity
asa.bachelor.klev enrobot
<<Java Class>>
LoadingPopup
asa.bachelor.klev enrobot.popups
<<Java Class>>
LOGG
asa.bachelor.klev enrobot.popups
<<Java Class>>
storagebox
asa.bachelor.klev enrobot.sql
<<Java Class>>
SQLreader
asa.bachelor.klev enrobot.sql
<<Java Class>>
SQLwriter
asa.bachelor.klev enrobot.sql
<<Java Enumeration>>
RobotTypes
asa.bachelor.klev enrobot.ENUMS
<<Java Enumeration>>
PropertyChangeItems
asa.bachelor.klev enrobot.ENUMS
<<Java Class>>
FragmentMain
asa.bachelor.klev enrobot.fragments
<<Java Class>>
FragmentAllAvailableTasks
asa.bachelor.klev enrobot.fragments
<<Java Class>>
FragmentTodaysTasks
asa.bachelor.klev enrobot.fragments
<<Java Class>>
FragmentParamters
asa.bachelor.klev enrobot.fragments
<<Java Class>>
FragmentJOGG
asa.bachelor.klev enrobot.fragments
<<Java Class>>
FragmentLastpage
asa.bachelor.klev enrobot.fragments
<<Java Class>>
AllAdapter
asa.bachelor.klev enrobot.adapters
<<Java Class>>
TodayAdapter
asa.bachelor.klev enrobot.adapters
<<Java Class>>
ParameterAdapter
asa.bachelor.klev enrobot.adapters
<<Java Class>>
JOGGerror
asa.bachelor.klev enrobot.popups
<<Java Class>>
Holder
asa.bachelor.klev enrobot.adapters
<<Java Class>>
Holder
asa.bachelor.klev enrobot.adapters
<<Java Class>>
Holder
asa.bachelor.klev enrobot.adapters
<<Java Class>>
HoldTask
<<Java Class>>
JOGGoff
<<Java Class>>
JOGGon
<<Java Class>>
GetJobList
<<Java Class>>
SelectJob
<<Java Class>>
sendParameters
<<Java Class>>
ResetAlarm
<<Java Class>>
RestartJob
<<Java Class>>
ServoOnOff
<<Java Class>>
StartJob
<<Java Class>>
statuscheckTHREAD
<<Java Class>>
HSEC
asa.bachelor.klev enrobot.HSEC


<<Java Class>>
PositionVariable
asa.bachelor.klevenrobot.HSEC
dataType: byte[]
form: byte[]
toolNumber: byte[]
userCoordinateNumber: byte[]
extendedForm: byte[]
firstAxisData: byte[]
secondAxisData: byte[]
thirdAxisData: byte[]
fourthAxisData: byte[]
fifthAxisData: byte[]
sixthAxisData: byte[]
seventhAxisData: byte[]
eighthAxisData: byte[]
-pos
0..1
<<Java Class>>
positionThread
asa.bachelor.klevenrobot.HSEC
~alarmResp
0..1
<<Java Class>>
ReadWriteRealResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
FileListResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
AlarmResetResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
StepCycleContinuousResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
HoldServoOnOffResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteExAxisResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteIOResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
SelectJobResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
StatusResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteIntegerResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
StartJobResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
RobotPositionResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteBasePositionResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWritePulsePositionVariableResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
AlarmReadResponse
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ResponseHeader
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
HSEC
asa.bachelor.klevenrobot.HSEC
<<Java Enumeration>>
JoggingIO
asa.bachelor.klevenrobot.HSEC
<<Java Enumeration>>
AddedStatus
asa.bachelor.klevenrobot.HSEC
<<Java Enumeration>>
Status
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
MessageHeader
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteBasePositionRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteExAxisRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWritePulsePositionVariableRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
HoldServoOnOffRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
FileListRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
SelectJobRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteRealRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
AlarmReadRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
AlarmResetRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteIntegerRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
StartJobRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
StepCycleContinuousRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
ReadWriteIORequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
StatusRequest
asa.bachelor.klevenrobot.HSEC
<<Java Class>>
RobotPositionRequest
asa.bachelor.klevenrobot.HSEC

Vedlegg 3


<<Java Class>>
GUI
(default package)
main(String[]):void
GUI()
initialize():void
selectedLine():void
update():void
<<Java Class>>
MYSQL
(default package)
MYSQL()
getInstance():MYSQL
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
getURL():String
setURL(String):void
loadTable(String):JTable
deleteTask(String,String):void
addNewTask(String,Vector<String>):boolean
connect():void
changeType(String):void
-loginPop
0..1
<<Java Class>>
login
(default package)
login()
execute():void
close():void
getPassword():String
getUsername():String
getAddress():String
showError(String):void
<<Java Class>>
confirmDelete
(default package)
confirmDelete(String)
onComplete():void
onCancel():void
execute():boolean
<<Java Class>>
addNewTodaysTask
(default package)
addNewTodaysTask(Vector<String>)
onComplete():void
onCancel():void
execute():boolean
<<Java Enumeration>>
VariableType
(default package)
NONE: VariableType
INT: VariableType
REAL: VariableType
POS: VariableType
<<Java Class>>
addNewJob
(default package)
addNewJob()
onComplete():void
onCancel():void
execute():boolean
<<Java Class>>
settings
(default package)
settings()
onComplete():void
onCancel():void
execute():boolean

Vedlegg 4

ASA ROBOT CONTROLLER

Manual

1 — Last update: 2015/05/27

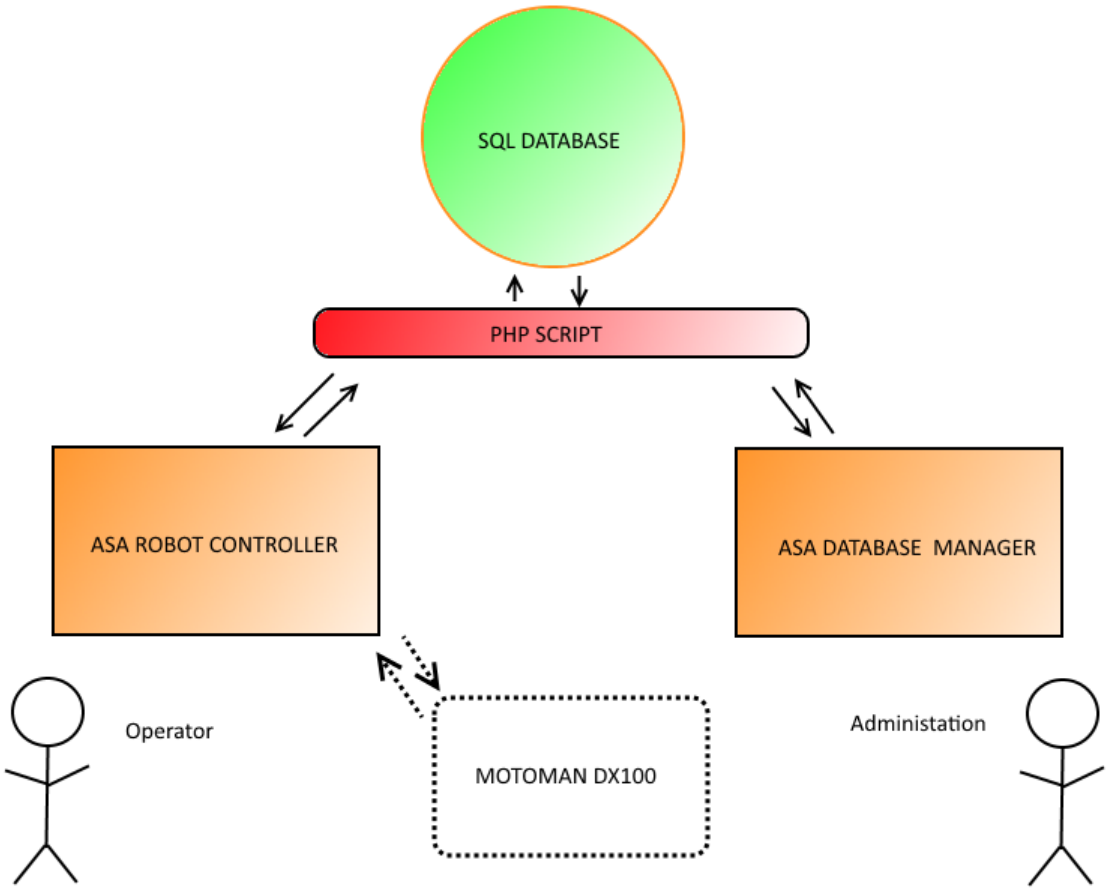
ASA

Table of Contents

General Information

This application is a part of the ASA robot controller project.
The system consist of:

ASA Robot Controller – Android application used to control the robot.
ASA Database Manager – Java application used to handle the SQL-database.



The ASA Robot Controller is used to execute programs on the robot that has been predefined in the database. Upon completion of a task, the application stores a record of the task in the same database.

The ASA Robot Controller communicates with the database through a server(hosting a php-scrict) and with the robot through High-speed ethernet.

Installation

As the application is not publicly released it cannot be found in the Android store.

To install the application, transfer the ASA_RC.apk to the android unit. This can be done by saving a copy from the computer on the unit-sd card.

Because the application is not official the unit has to allow for unknown sources. This is done on Android 4.4 by checking the box next to “unknown sources” (found on settings —> general —> Security).

When this is done, the application should be able to be installed by clicking on the ASA_RC.apk from the Android unit.

BE AWARE: After installing the application the login-data and ip addresses will be empty. This has to be changed under settings.

How to use

This section explains how to use the application:

1. Login
2. Start page
3. List of available tasks
4. List of scheduled tasks
5. Fill in parameters
6. Manual controll (JOG)
7. Executing tasks
8. System log

Login

The information needed to access the robot (robot IP, port 1, port 2) and the database/server (Username, Password, server address) is stored in the application memory.
This information is persistent and will only need to be provided on first use of the application.

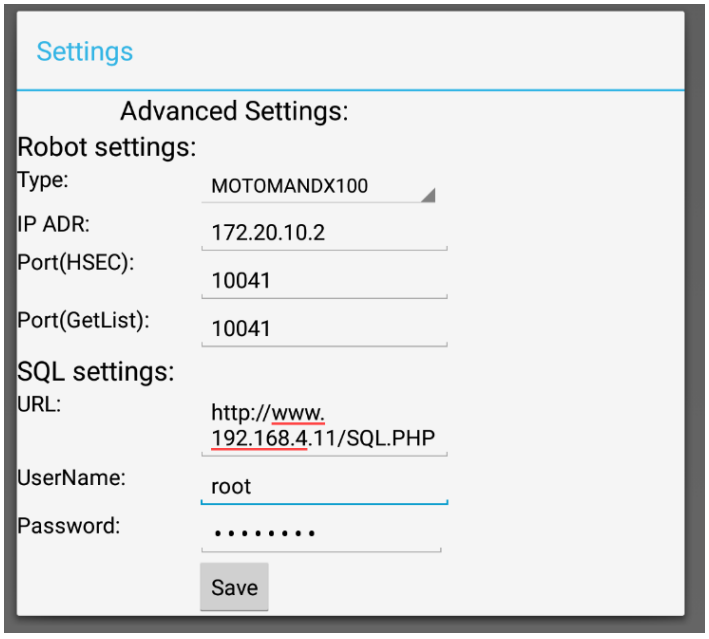
To provide the application with this information, press **settings** on the menubar.

For the connection to the robot:

- Type: Type of robot controller (only MOTOMAN DX100 is supported for the time being).
- IP address: the ip address of the robot.
- Port(HSEC) : The port for High-speed ethernet client (used for all commands exept getting the tasks on the robot)
- Port(GetList): The port for High.speed ethernet client (only used for getting the tasks on the robot)

For connection with the database/server:

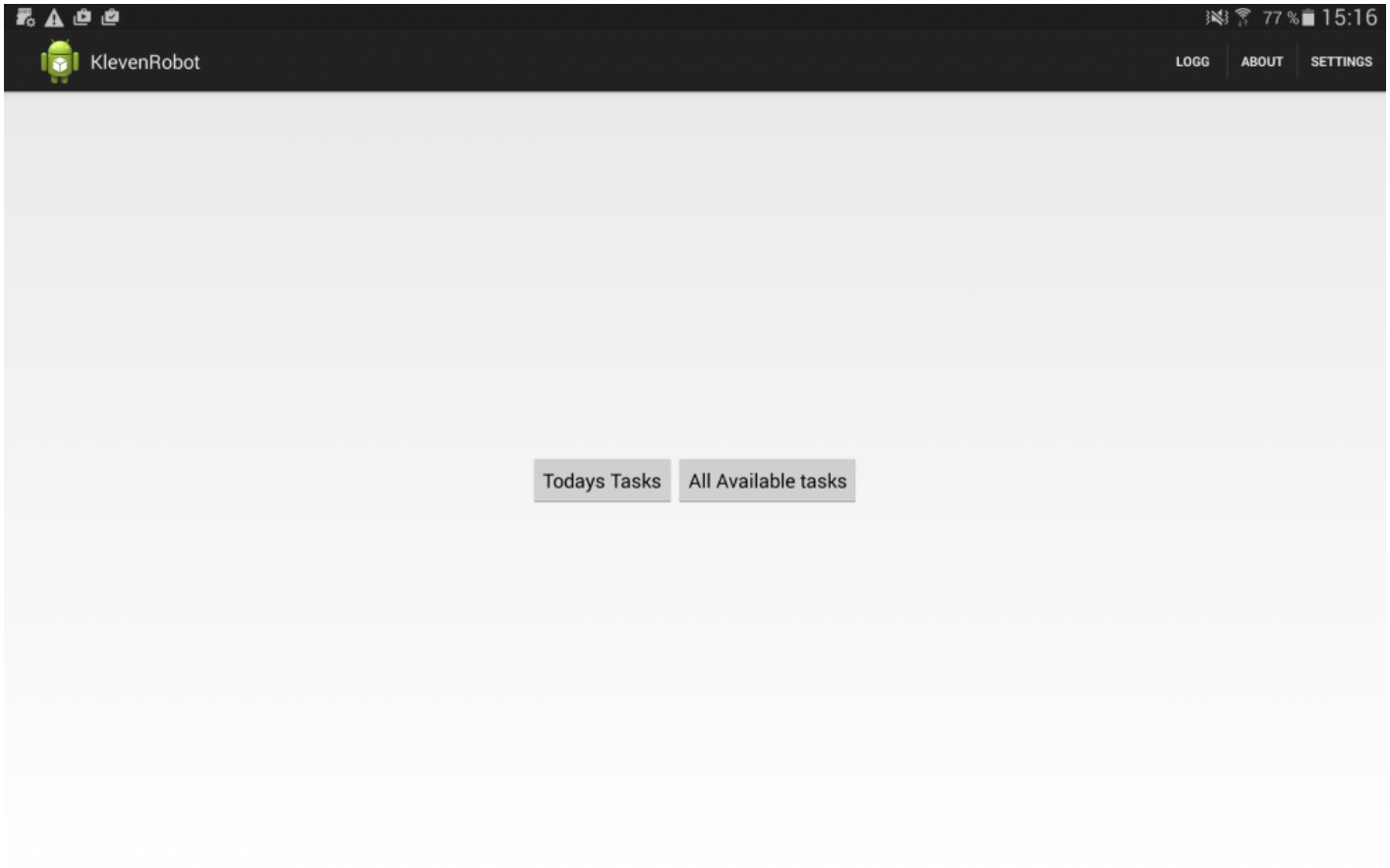
- URL: The location of the php-script (must include *http://*).
- UserName: The username to access the database.
- Password: The password to access the database.



Start page

The first page provides the ability to choose between all available tasks or scheduled tasks.

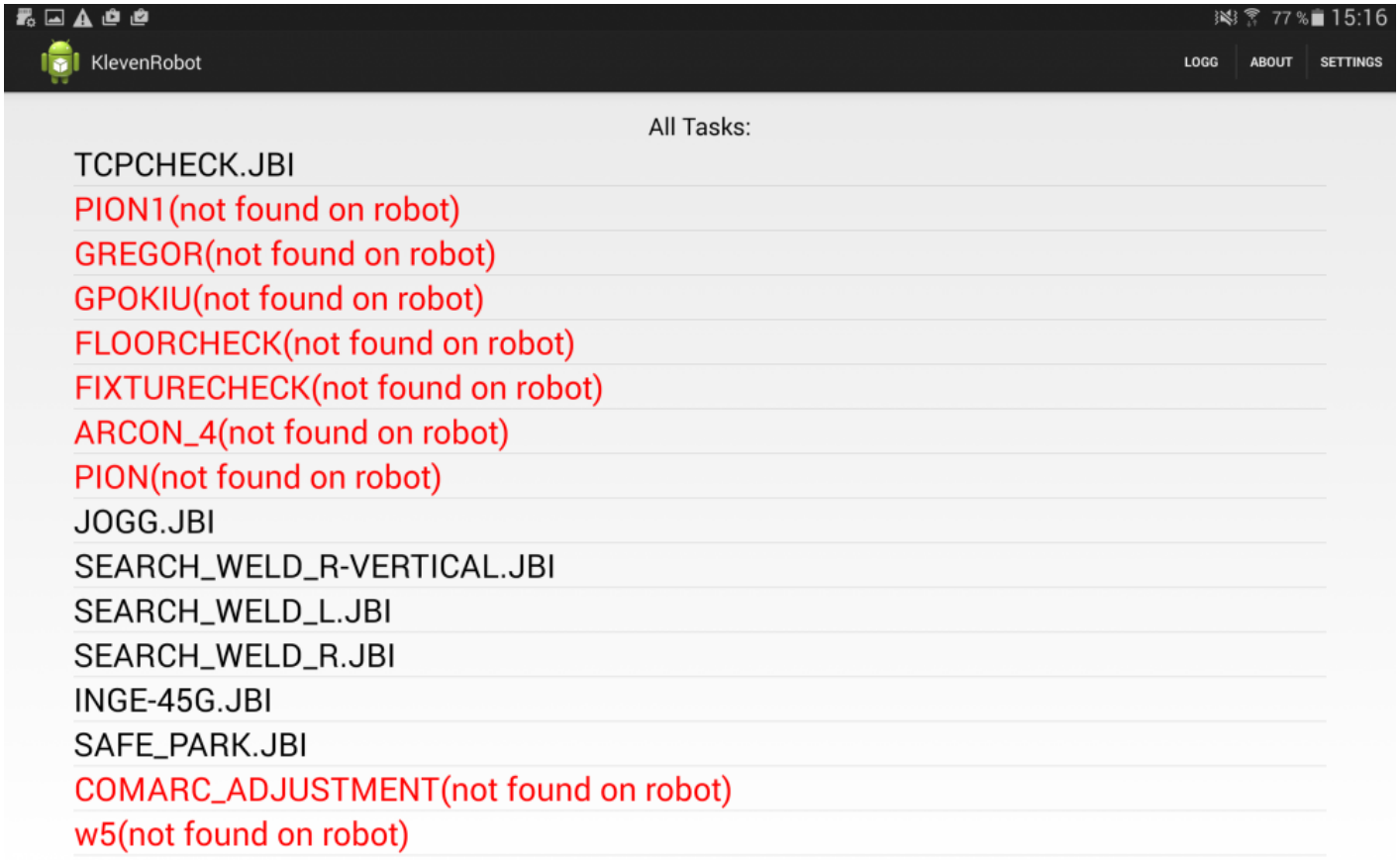
When pressing one of the buttons, a request is sent to the server to gather the necessary information. If the server is unresponsive or the provided login-data is wrong, an error will occure and be presentet in the loading window. If this is the case, the login-data can be updated using the settings tab located at the menubar.



List of available tasks

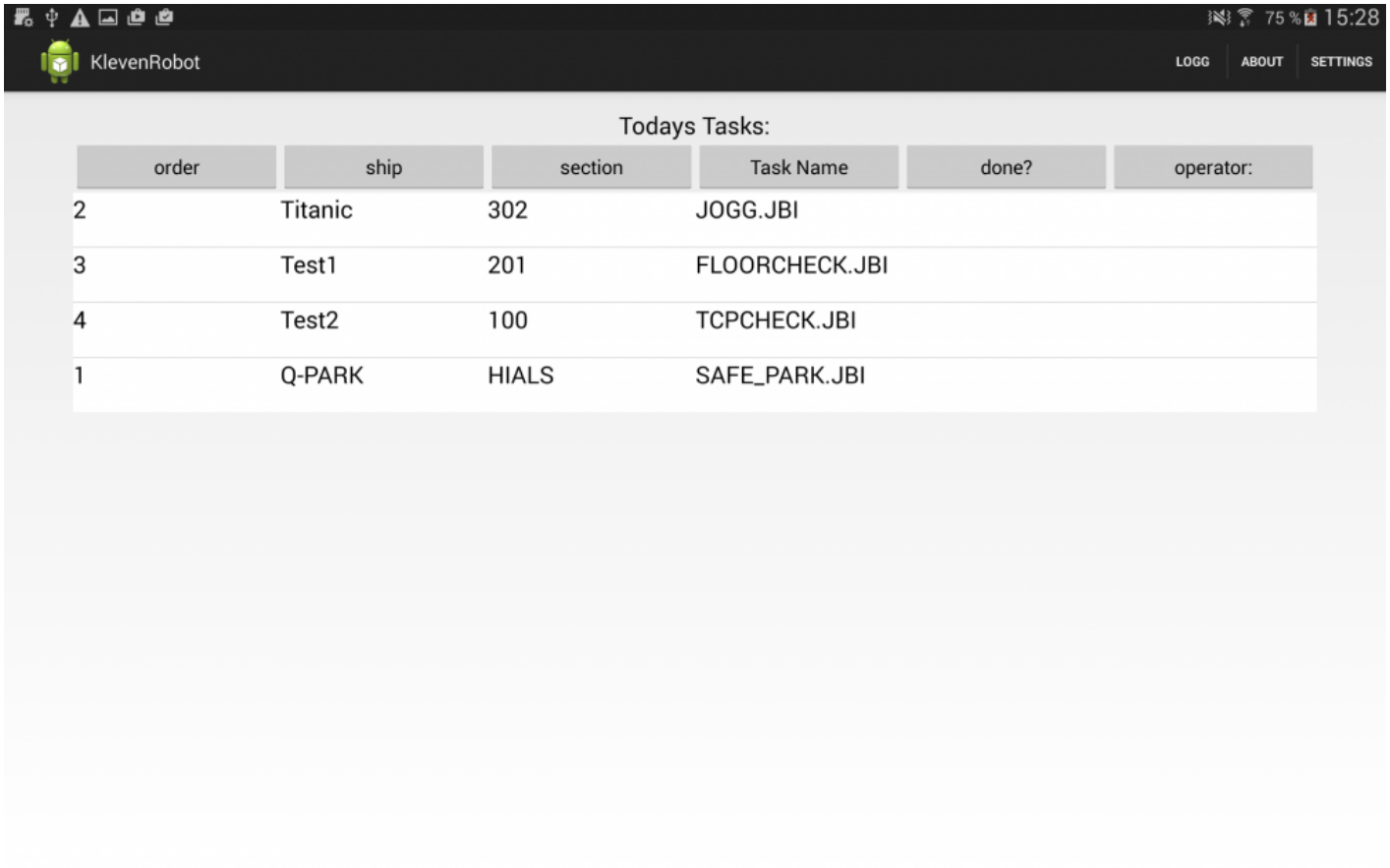
This list presents all tasks stored in the database. Tasks not found on the robot will be marked in red with the text: “not found on robot”.
If a task is missing, make sure the **taskname** stored in the database is the same as the name of the program on the robot (without .JBI).

To select a task, simply click on the desired row and the application will continue on to the next step.
To go back, press the “back button” on the unit itself, which will make the application go back to the startpage.



List of scheduled tasks

This list presents all scheduled tasks stored in the database.
If the task is completed, the “operator” field will tell the name of the operator, and the “done?” field will say yes.
The scheduled task will not be removed upon completion, this will have to be done from the ASA Database manager.



To select a task, simply click on the desired row and the application will continue on to the next step.
To go back, press the “back button” on the unit itself, which will make the application go back to the startpage.

Fill in parameters

Upon choosing a task from either the list of available task or the list of scheduled tasks, a request is sent to the database and the parameter page is opened.

The parameter page is used to fill in all the required information necessary to execute the task.

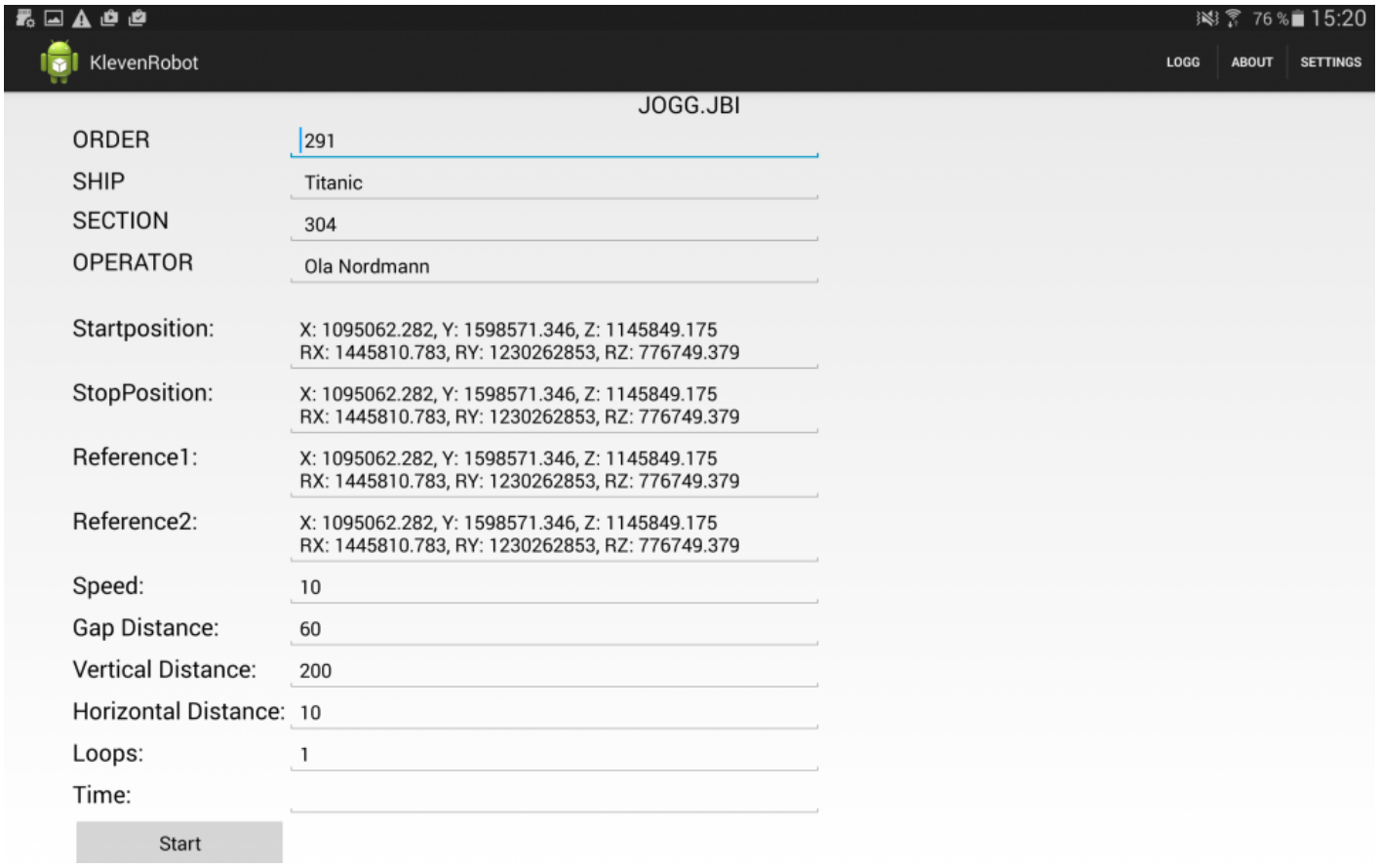
The four first parameters is used for documenting the finished tasks:
Order: order number (if scheduled, this is already filled in)
Ship: name of the ship (if scheduled, this is already filled in)
Section: section number (if scheduled, this is already filled in)
Operator: name of the person that is executing the task.

The rest of the parameters are dynamicly adapted to fit the selected task. In the left column the name of the parameters are listed. The application currently supports up to 10 parameters and 3 types of parameters. The type of the parameter is stored in the background and is not visible to the user.

1. INT: whole number
2. REAL: decimal number
3. POS : position variable

To change a parameter, click on the textfield to the right of the wanted parameter. If the parameter is **INT** or **REAL**, the onscreen-keyboard will be displayed.
If the type is **POS**, the application will change to the manual control(JOG) screen.

Upon returning for the manual control screen, the selected position will be displayed next to the selected parameter.



When all the parameters has been filled in, the task is executed by pressing the start button located at the buttom of the screen.
To go back, press the “back button” on the unit itself, which will make the application go back to the previous page.

Manual control (JOG)

The manual control page is automaticly opened when the user select a position variable from the parameters page.

The application is able to JOG the robot in two different kind of ways:

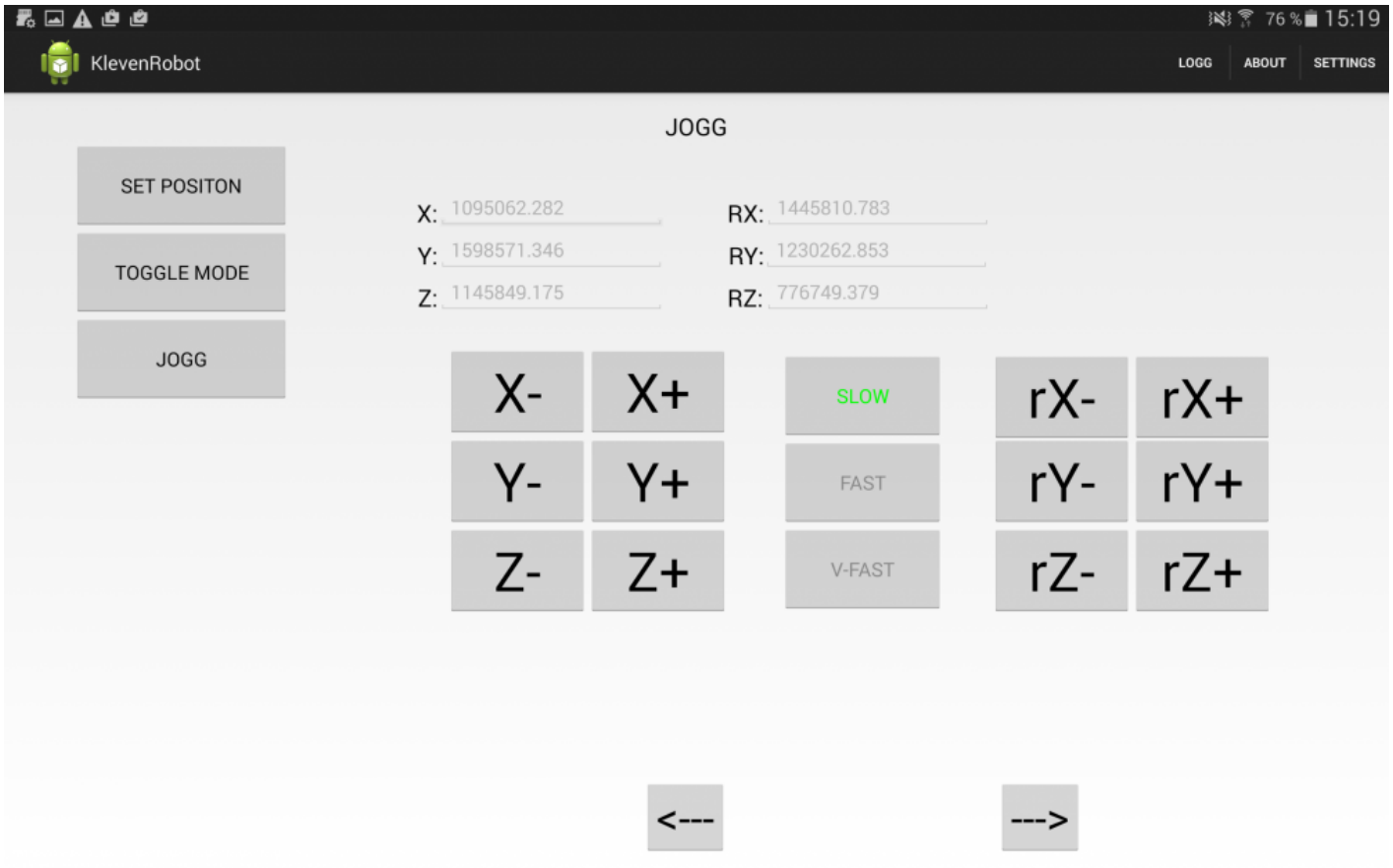
1. World Coordinates – controlling the position of the tool centerpoint. The robot will control the joints to reach the desired position.
2. Joint Coordinates – Controlling each joint individually.

Changing between modes are done by clicking on the “TOGGLE MODE” button.

To manually control the robot, the “JOGG” button needs to be pressed and hold. When the “JOGG” button is pressed, the robot can be controlled by using the 12 buttons located at the center of the screen.
In World Coordinates the buttons will say X+-, Y+-, Z+- and rX+-, rY+-, rZ+-. X, Y, Z are the axis of the workspace, while the small “r” is rotation around the selected axis.
in Joint Coordinates the buttons will be marked S+-, L+-, U+-, R+-, B+- and T+-. Each letter is representing one joint (the letter is printed on each joint on the robot) where S is the first joint and the T is the endjoint.

The 3 buttons in the middle are used to change the speed of the movement. The default speed value is SLOW, and the current speed will be marked in green.

On the button of the screen, the two arrow buttons are used to move the base of the robot (the robot is running on rails).



When satisfied with the position of the robot, the SET POSITION button is used to save the current position and return to the previous side.
If the back button on the unit is pressed, the application will go back to the previous page without saving the position.

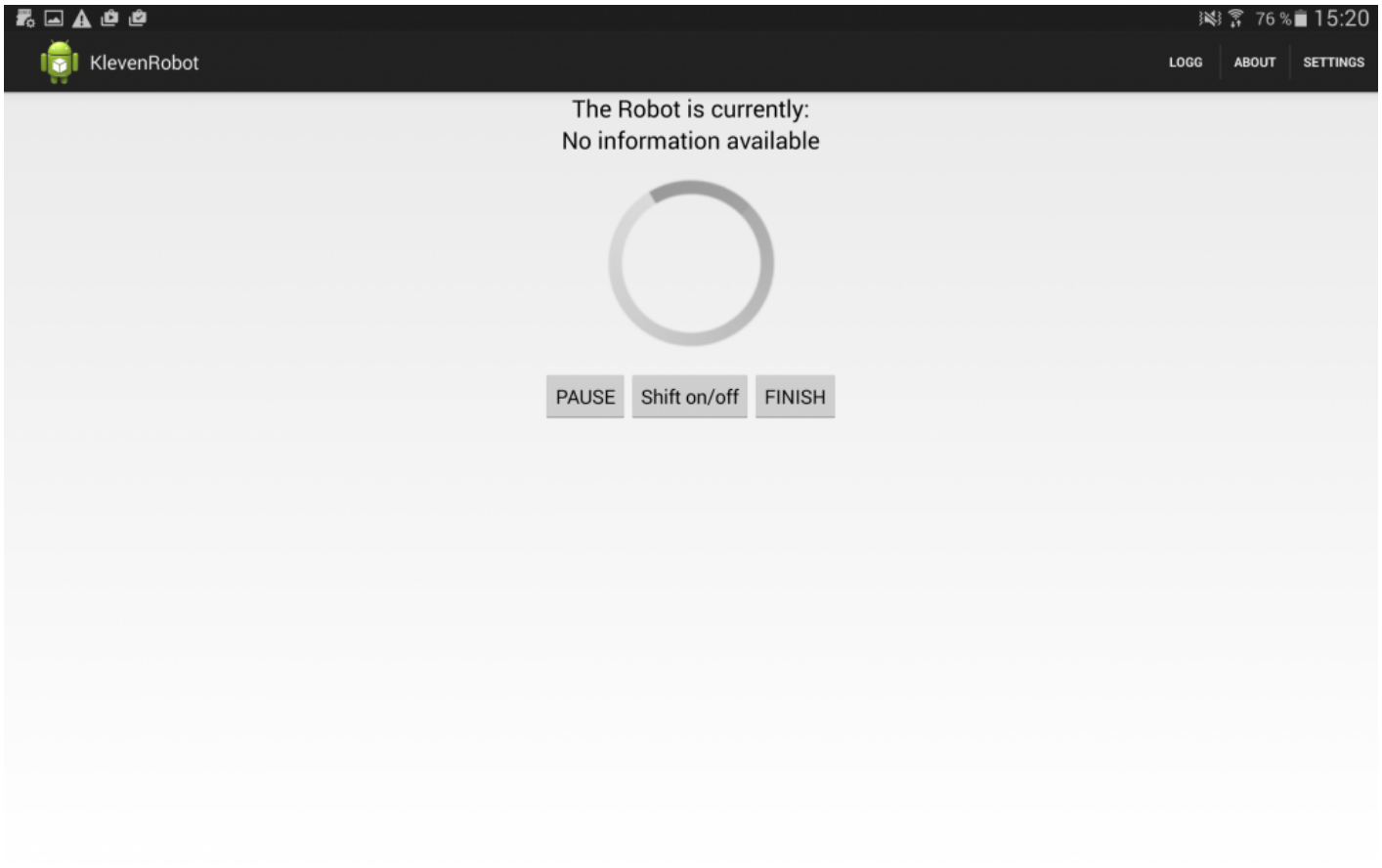
Executing task

Executing a task is done by pressing the start button located at the bottom of the parameters page.

When the button is pressed. The application will transfer the information to the robot and execute the task. On the application, a spinner will be displayed together with the status of the robot:

1. Running – The program is running.
2. Paused – The program is paused.
3. Finished – The program is finished with the task.

When the program is paused, either by pressing the pause button on the application or by using the DX100, a red pause icon will be displayed.
The application is able to resume the task by pressing PLAY.

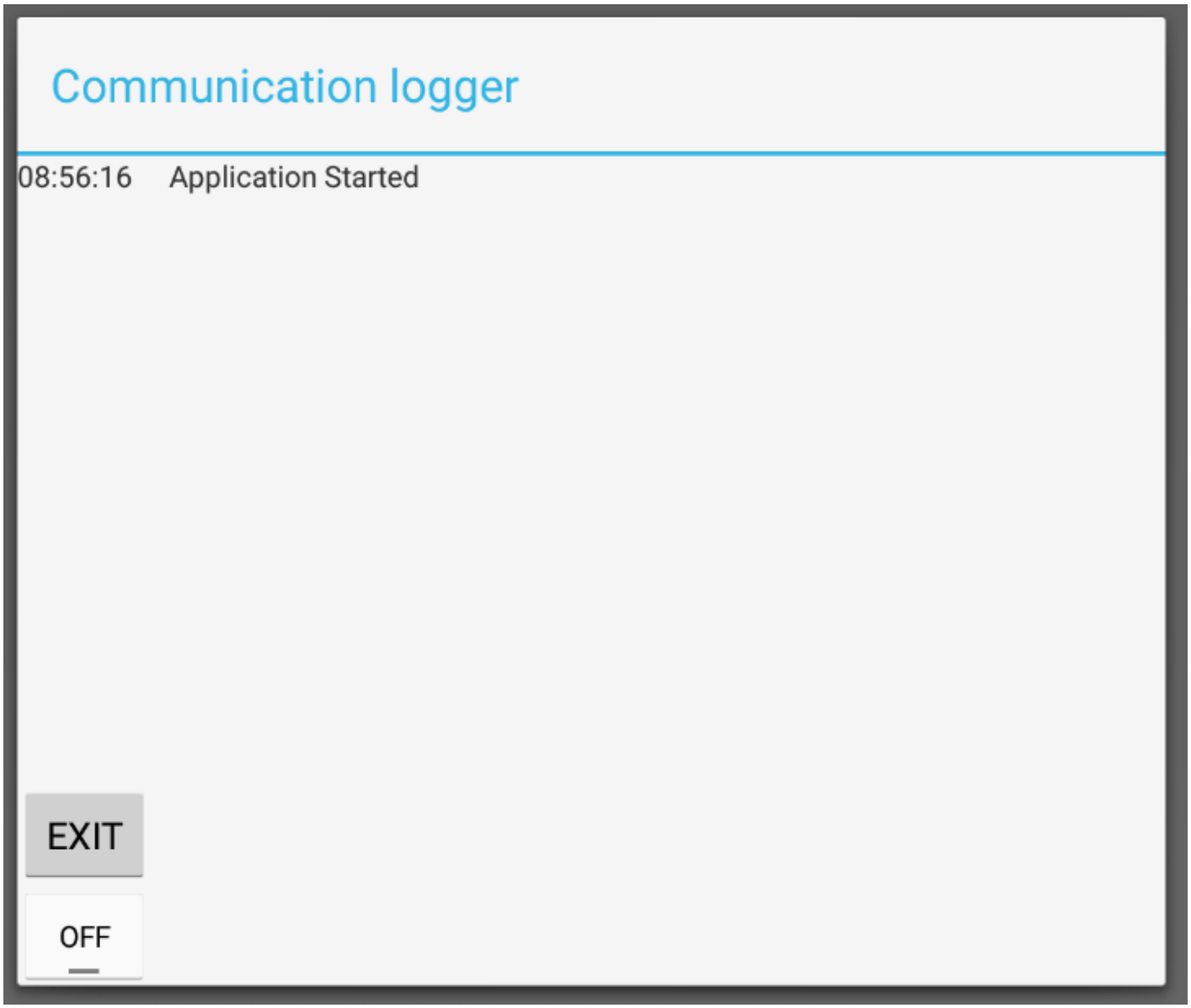


When the program is finished, the whole screen will be green and a “Finished icon” will be displayed. This also makes the FINISH button available for the user.

When the finish button is pressed, the documentation of the executed task is sent to the database and the application goes back to the first page.

System log

The system log can be found in the menubar on the top right corner.
In the system log, all requests and errors are displayed among with the time of occurence.
The log is automaticly updated, and by pressing “ON”-button, the window will automaticly scroll to display the last message.



Vedlegg 5

ASA DATABASE MANAGER

Manual

1 — Last update: 2015/05/26

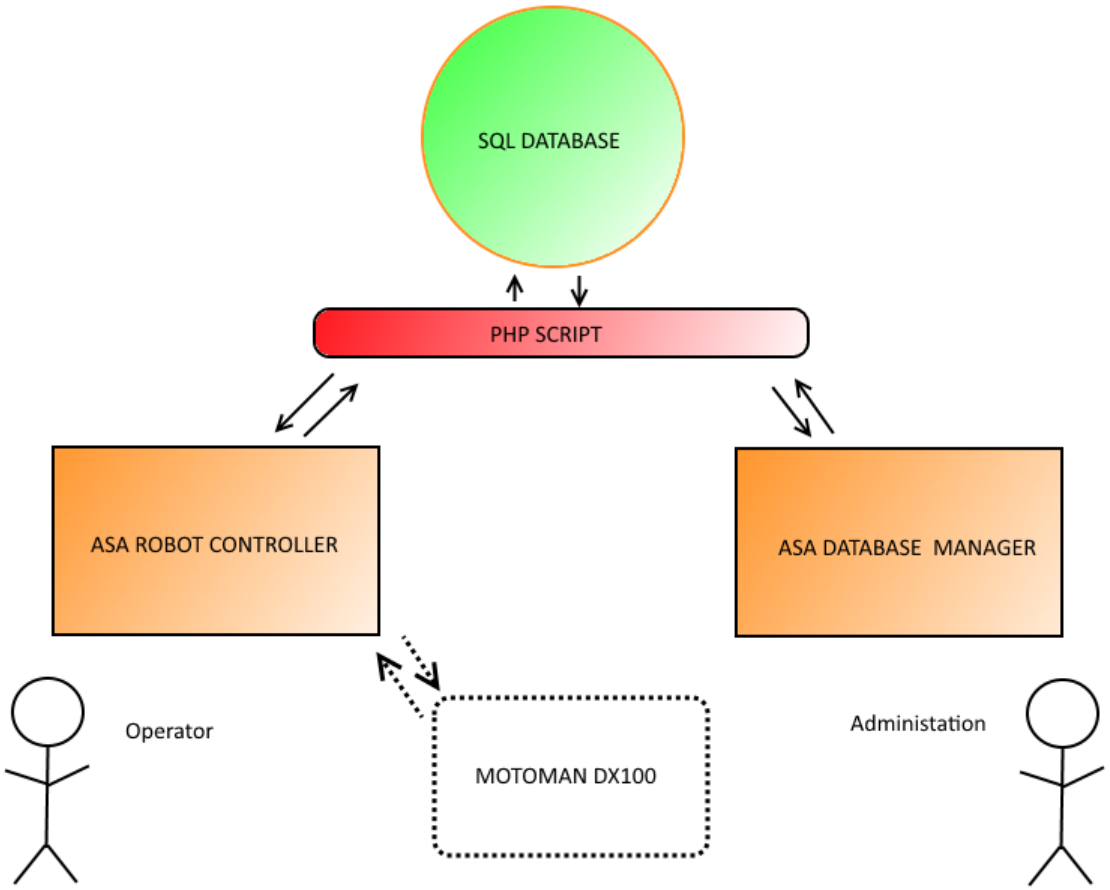
ASA

Table of Contents

General Information

This application is a part of the ASA robot controller project.
The system consist of:

ASA Robot Controller – Android application used to control the robot.
ASA Database Manager – Java application used to handle the SQL-database.



The ASA Database Manager takes care of the handling of the SQL-database in the project. The ASA Database Manager provides the ability to add new tasks, schedule a task for execution and remove entries from the database.
It also provides the ability to print the chosen table.
The application is made in Java and communicates with the SQL-database through a PHP-script.

Installation

The ASA_DBM.JAR file can be downloaded directly from the server running the PHP-script by writing:

http://www. **Server ip** /ASA_DBM.JAR

The application can run on any computer that supports .JAR files and no installation is required.

How to use

This part will explain how to use the application.

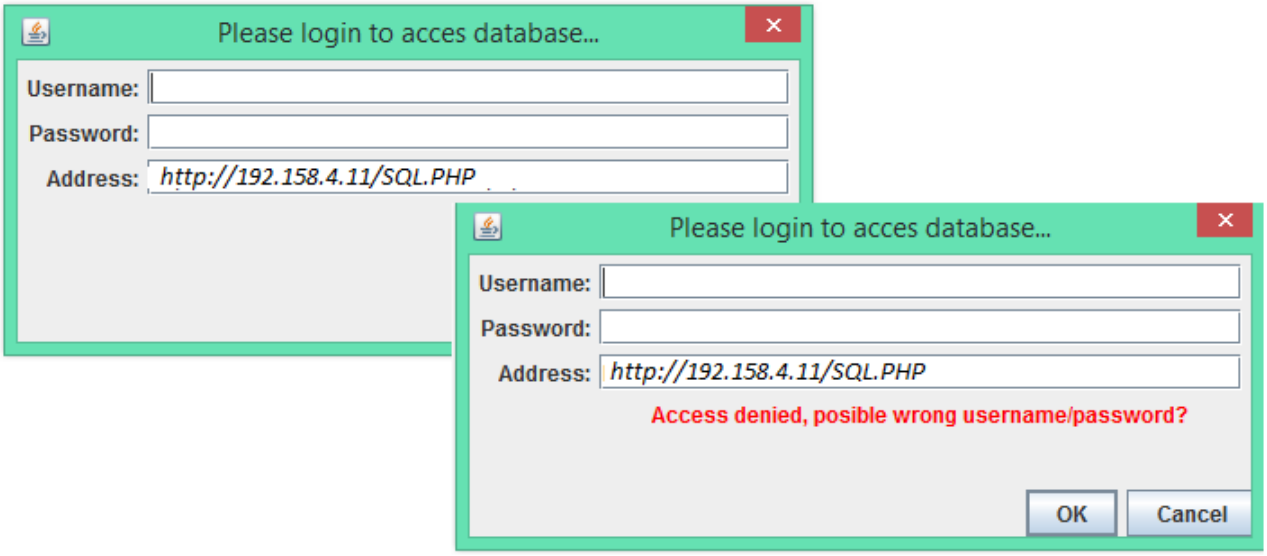
1. Login
2. Main window
3. Add new task
4. Add new sheduled task
5. Remove entry
6. Print table

Login

When starting the application, the Login-window is displayed.

This information is required to connect to the database. If the provided information is wrong, the application will provide information about the error.
Upon completion, the provided information is used to get the first table from the database, and main window is presented.

Username: The username used in the database.
Password: The password used in the database.
Address: The location of the PHP-script. (the complete address to the PHP-script, including **http://**



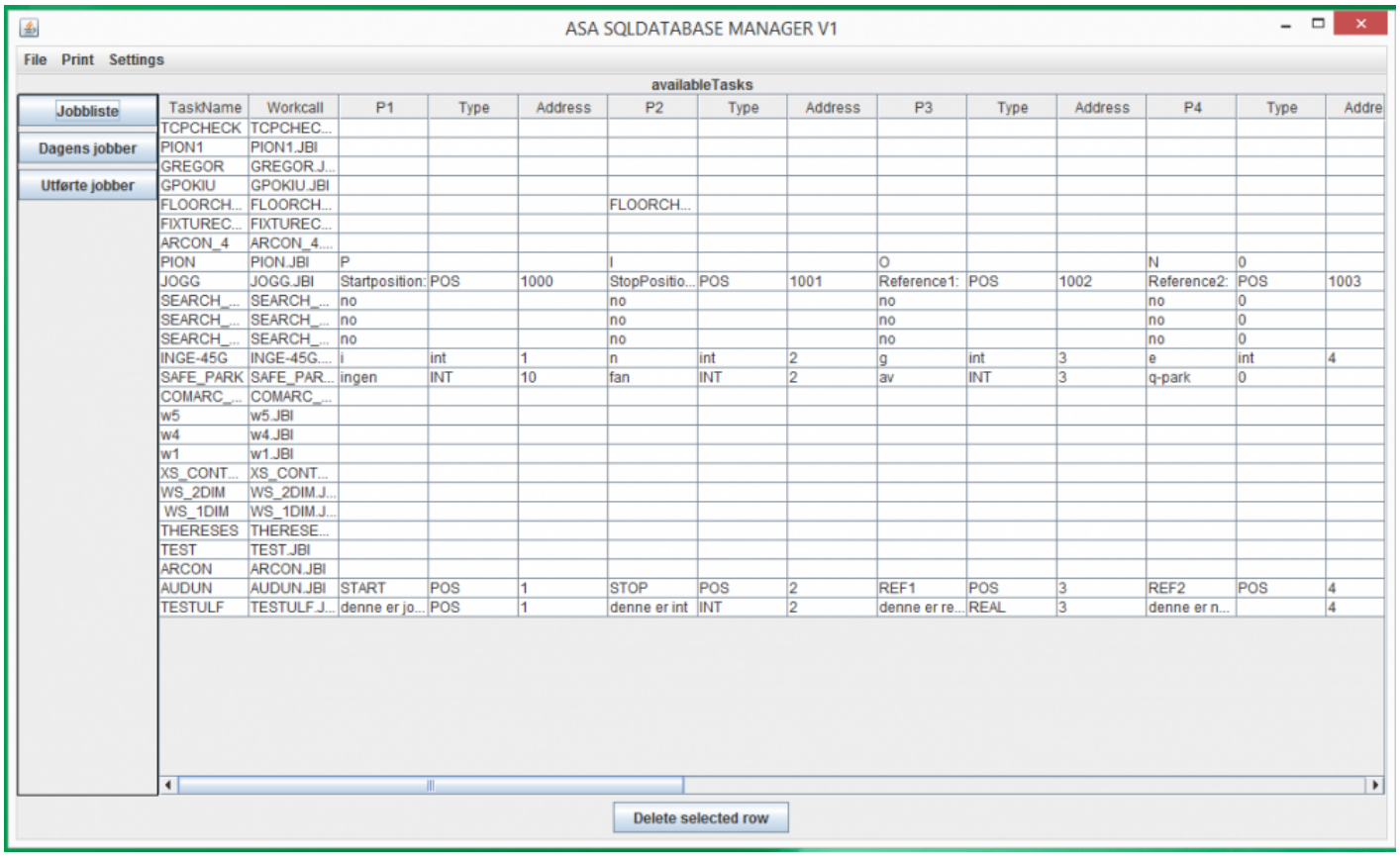
Main window

The main window displays the chosen table from the database. Changing between tables are done by using the provided buttons located in the left menu.

The current database consist of 3 tables:

1. Jobbliste – Table with all available tasks and the required parameters.
2. Dagens jobber – Table with all scheduled tasks and the accompanying information.
3. Utførte jobber – Table of all finished tasks and the accompanying information.

From the menubar it is possible to add new tasks, schedule task execution, change login-data and print the displayed table. It is also possible to delete any entry by selecting a row and press “Delete selected row” located at the bottom of the page.



TaskName	Workcall	P1	Type	Address	P2	Type	Address	P3	Type	Address	P4	Type	Addre
TCPCHECK	TCPCHEC...												
PION1	PION1.JBI												
GREGOR	GREGOR.J...												
GPOKIU	GPOKIU.JBI												
FLOORCH...	FLOORCH...				FLOORCH...								
FIXTUREC...	FIXTUREC...												
ARCON_4	ARCON_4....												
PION	PION.JBI	P			I			O			N	0	
JOGG	JOGG.JBI	Startposition:	POS	1000	StopPositio...	POS	1001	Reference1:	POS	1002	Reference2:	POS	1003
SEARCH_...	SEARCH_...	no			no			no			no	0	
SEARCH_...	SEARCH_...	no			no			no			no	0	
SEARCH_...	SEARCH_...	no			no			no			no	0	
INGE-45G	INGE-45G....	i	int	1	n	int	2	g	int	3	e	int	4
SAFE_PARK	SAFE_PAR...	ingen	INT	10	fan	INT	2	av	INT	3	q-park	0	
COMARC_...	COMARC_...												
w5	w5.JBI												
w4	w4.JBI												
w1	w1.JBI												
XS_CONT...	XS_CONT....												
WS_2DIM	WS_2DIM.J...												
WS_1DIM	WS_1DIM.J...												
THERESES	THERESE...												
TEST	TEST.JBI												
ARCON	ARCON.JBI												
AUDUN	AUDUN.JBI	START	POS	1	STOP	POS	2	REF1	POS	3	REF2	POS	4
TESTULF	TESTULF.J...	denne er jo...	POS	1	denne er int	INT	2	denne er re...	REAL	3	denne er n...		4

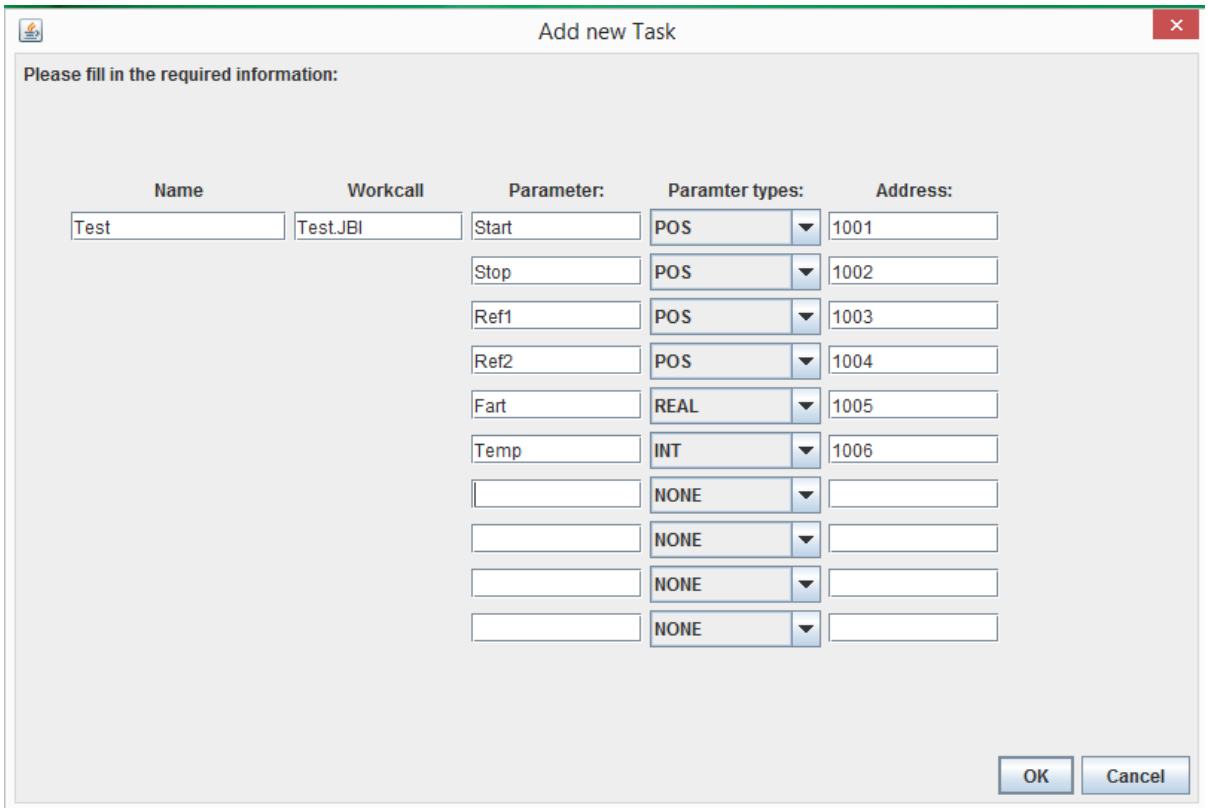
Add new task

The add new task window is opened by pressing *file—>legg til ny jobb* in the main window.
The add new task window is used for adding new tasks to the database.

To make a new task, the listed information needs to be provided:

- Name : The name used to locate this task in the ASA Robot Controller.
- Workcall: The name of task located at the robot. (This should be the same as Name + .JBI extension)
- Parameter: Name of the parameter. (Start point, Stop point, Speed, Height)
- Parameter types: The type of the parameter. The supported types is provided in the dropdown list (INT, REAL, POS, NONE). (by selecting NONE, the parameter will not be used)
- Parameter address: The address of the variable in the program. This is defined when writing the program at the robot.

By pressing **OK** the task is added to the database and can be found in the “alle tilgjengelige jobber” table.
The displayed table should automaticly update as soon as the database has updated the table.
By pressing **Cancel** the window is closed and no message is sent to the server.



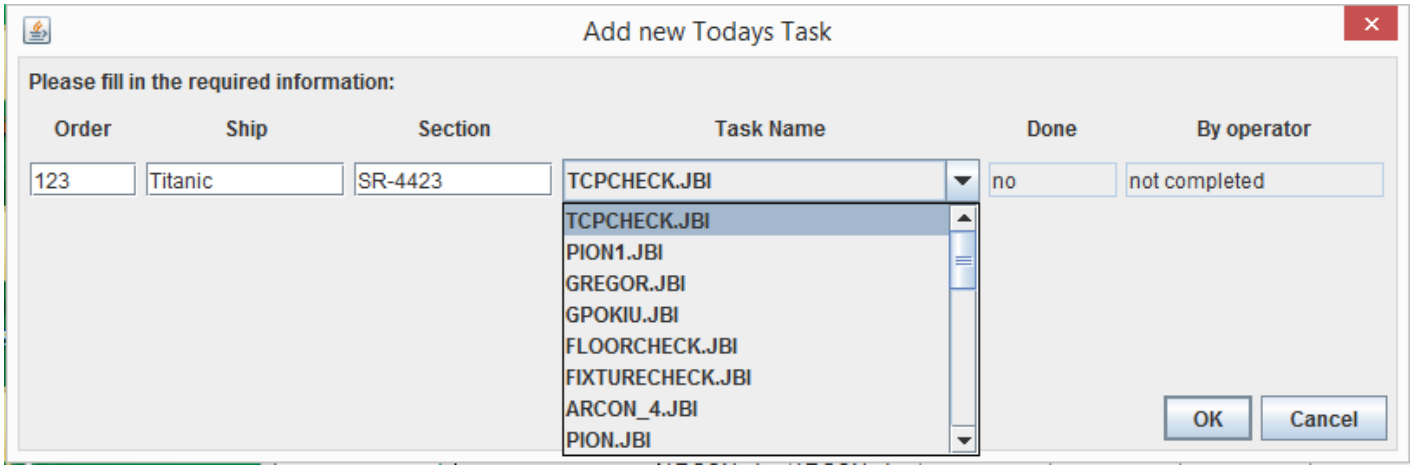
Add new scheduled task

The *schedule task* window is opened by pressing *file—>legg til dagens jobber* in the main window.

To schedule a task the listed information needs to be provided:

- TaskName : The name of the selected task. The available tasks to choose from is listed in the dropdown-list.
- Order: The order of the scheduled task.
- Ship: Name of the ship.
- Section: Section number.

By pressing **OK** the scheduled task is added to the database and can be found in the “Dagens Jobber” table. The displayed table should automaticly update as soon as the database has updated the table.
By pressing **Cancel** the window is closed and no message is sent to the server.

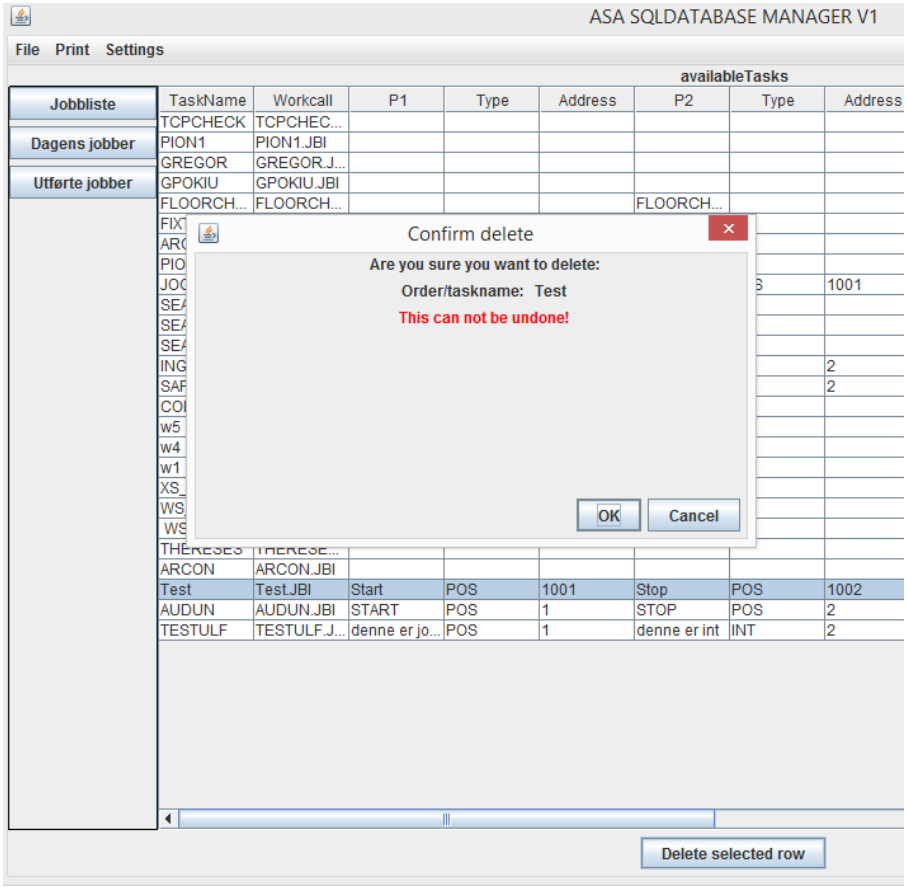


Remove entry

To remove an entry from any of the 3 tables, first select the desired row by clicking on it (the row will then become blue).
If you are sure you want to delete the selected line, simply press the “Delete selected task” located on the bottom of the page.

This will open a popup-window asking for a confirmation and display either the ordernumber and the taskname of the chosen entry.

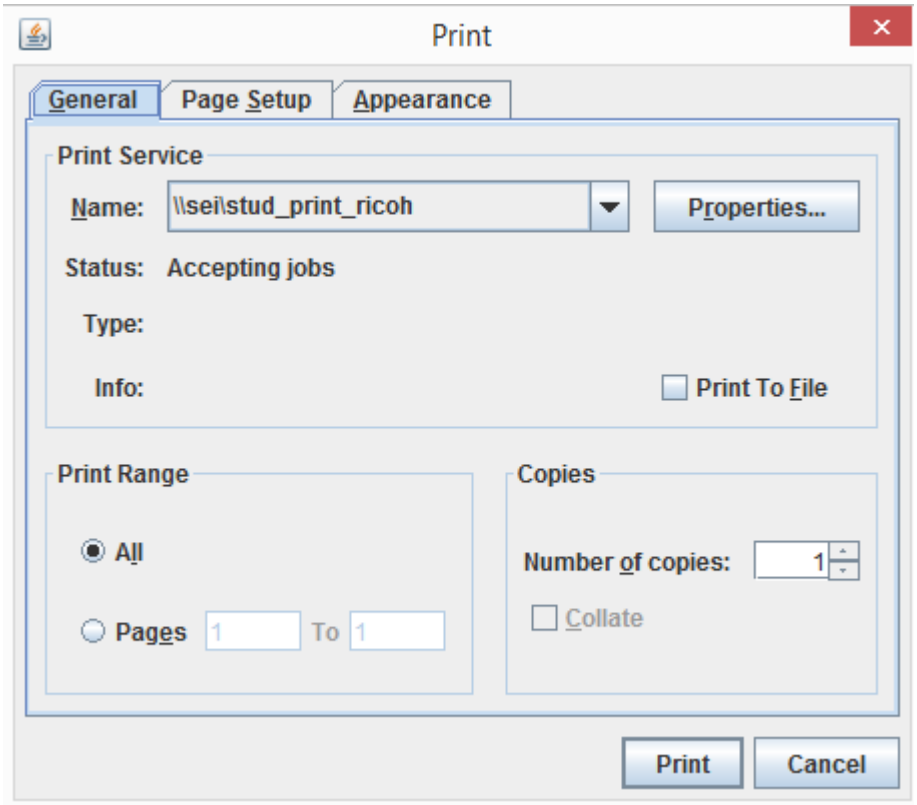
By pressing **OK** the entry will be permanently removed from the database (**THIS CAN NOT BE UNDONE!**).
By pressing **Cancel** the operation is aborted and the entry will not be removed.



Print table

To print a table make sure the correct table is displayed, as this is the only table to be printed.
When the desired table is displayed, press the “Print” button located at the menubar.

By pressing the **Print table** button, a popup-window will be displayed containing settings for the printer.
After choosing the right settings for your printer, press the **Print** button located at the bottom of the window.
By pressing **Cancel** the window will close, and the print will be aborted.



Vedlegg 6

Møtereferat

Den 16.01.15 hadde vi første møte ang. Bachelor oppgaven, på rom F420.

Tilstede: Simon Langlo, Anders Kobbevik, Audun Westerskow, Hans Støle, Ottar Osen.

Vi presenterte de to forskjellige oppgavene for veilederne, første oppgave er bruker grensesnitt for styring av robot fra tablet, andre oppgave er plassering og sveising av avstivere på skrogplate ved hjelp av vision og to roboter.

Den første oppgaven antok gruppen for å vere litt for liten til å vere en bachelor oppgave, og den andre oppgaven for stor. Eit alternativ vi foreslo var å skalere ned oppgave 2 slik at den vart overkomlig, og utføre den på de 2 ekornes robotene i kjelleren.

Ottar konkluderte med at oppgave 1 ikke var for liten, og la fram at der er mange utfordringar innen kommunikasjon mellom tablet og roboten. Det er også masse arbeid i å lage eit bra og funksjonelt grensesnitt.

Etter oppveiing av for og mot argumenter, kom vi fram til å gå for den første oppgaven å lage en Robot kontroller på tablet. Det ble også vedtatt og koble inn Helge Tor som har god kompetanse på dette området.

Det vart planlagt å kalle inn til nytt møte etter forprosjektsrapporten er innlevert, om ca 2 uker.

Møtereferat

Den 6.02.15 var det møte med veileder og oppdragsgiver ang. Bachelor oppgaven, på rom F420.

Tilstede: Simon Langlo, Anders Kobbevik, Audun Westerskow, Hans Støle, Ottar L. Osen, Simon Sundal.

Møtet startet med at gruppen viste frem applikasjonen, og la frem ideer som jobbliste i excel og generering av pdf kvittering for hver jobb med data som: Operatør, båt, seksjon, dato osv. Det vart mottatt som en god ide og det kom da forslag fra oppdragsgiver og veiledere om å bruke bedriftens mySQL database for dette. Det vart også diskutert muligheter for at jobbene kan ligge lagret i en ekstern database å lastes opp til roboten når det skal brukes.

Det ble diskutert forskjellige måter å jogge roboten på: ledd for ledd, robot koordinater og eventuelt å teste alternative metoder som hastighets styring. Siden roboten er uten sikkerhetssoner, vart det vedtatt å legge til en sikkerhetsfunksjon som f.eks. "dødmannsknapp" som må holdes inne for at det skal være mulig å jogge roboten.

Gruppen kom med forslag å bruke "motoman dx100 ethernet server" for kommunikasjon mellom nettbrett og robot. Dette gjør at en kan unngå å bruke PLS som mellomledd, og kan da kommunisere direkte med roboten. Veileder kom også med forslag om å legge til støtte for andre roboter som f.eks kuka og universal robot, for å få en mer allsidig applikasjon.

Til slutt vart det diskutert måter å forbedre grensesnittet på, det vart nevnt ting som å gjøre det mer visuelt og fjerne innstillinger operatører ikke har bruk for. Veileder oppfordret gruppen til å utføre intervju av noen operatører, der det bør bli spurt om irritasjonsmoment med den eksisterende løsningen og forlag til forbedringer. Det vart konkludert med at gruppen skal reise til kleven og gå gjennom prosessen som operatørene gjør steg for steg, og utføre intervjuene.

Møtereferat

Den 16.02.15 var det møte med Helge Tor Kristiansen ang. design av applikasjon på rom F424.

Tilstede: Anders Kobbevik, Audun Westerskow, Helge Tor Kristiansen.

Møtet startet med å vise frem skisser til forslag av applikasjon fra https://www.draw.io/ og applikasjonen som har blitt laget. Ble snakk om negative og positive sider av forslagene; som bakgrunnsfarge, kontinuitet av plassering av knapper og diverse.

Med hensyn til lys i sveisehallen, bør det være svart eller hvit bakgrunn for å få frem synligheten best mulig? Ved hvit skjerm brukes det mer batteri, vil det derfor være bedre med svart bakgrunn? Vil det bli mye refleksjon på skjerm, på grunn av lys? Bør diskuteres og finne ut av ved neste bedriftsbesøk.

Gruppen lurte på regler for copyright, og hvordan en kan unngå å bli tatt for copyright ved å referere til eigeren. Forslaget fra Helge, var å lage en info knapp, hvor det blir fortalt hvem utviklerne av applikasjonen og referere til diverse eigere av logoer, og «takk til:» felt.

Ved neste bedriftsbesøk vil vi spørre operatør om eventuelle forbedringer, og spurte Helge om gode tips til å få frem operatørens meninger til produktet. Ved å ha mange skisser og spør operatør om hva han synes og la ham koble sammen den «beste» løsningen vil være en god måte å få pekepinner for hvordan applikasjonen bør ende opp.

Spør operatør om nyttige funksjoner som;

- Lyd
- Lommelykt
- Vibrering

Til slutt ble det sagt at, jo flere skisser, jo bedre vil utviklingen av selve applikasjonen bli. Vi planla at neste møte vil bli planlagt over mail, og at Helge bare er tilstede hver 14. dag, men tilgjengelig på mail ved behov for veiledning.

Møtereferat

Den 27.02.15 var det møte med veiledere angående bachelor oppgaven, på rom B434.

Tilstede: Simon Langlo, Anders Kobbevik, Audun Westerskow, Ottar L. Osen og Hans Støle.

Møtet startet med at gruppen presenterte fremgang og problemstillinger ved prosjektet. Problemstillingene denne perioden var kommunikasjonen mellom motoman og pc. Gruppen var på bedriftsbesøk hos Kleven 24.02 og brukte hele dagen på å prøve å fikse dette , uten hell. Har derfor begynt å se på Motonis protokollen som går over en annen port enn med MotoCom som gruppen har laget programmet for. Har også hatt litt problem med multitouch funksjoner på «jogging»- siden.

Gruppen fikk litt tips om å ha en logg i databasen hvor pdf av jobber som har blitt gjort skal bli lagret, istedenfor å lagre desse lokalt på nettbrettet.

Gruppen forklarte også at det vil være mer hensiktsmessig å bruke PLS for å «jogge» roboten, på grunn av forsinkelser over motocom, og vil derfor ikke være skikkelig realtime. Veileder ville at gruppen skulle teste ut dette, og dokumentere for dette i rapporten. Og vil derfor føre til å finne den beste løsningen.

Gruppen hadde også bedt om å få en virtuell maskin med Høgskolen, men hadde egentlig ikke bruk for den lenger. Men veileder ville vi skulle teste denne, for å vite om dette er en bra løsning for videre utvikling av tidligere bachelor prosjekt.

Veilederne la frem at gruppen kunne begynne å lage plakaten som skal vise og forklare hva resultatet av oppgaven. Det vil også være ett seminar hvor alle bachelor oppgavene skal bli presentert gruppevis.

Møtereferat

Den 18.03.15 var det møte med veilederne angående bacholer oppgaven, på rom A436.

Tilstede: Simon Langlo, Anders Kobbevik, Audun Westerskow, Simon Sundal og Hans Støle.

Gruppen startet med å presentere det som hadde blitt gjort denne perioden. Første siden på applikasjonen har blitt oppdatert, etter hvordan operatørene ville ha det. Det endte opp med en etterligning av pendant styringen, med mulighet til å endre fra tradisjonell styring til Joystick styring. Det har også blitt lagt til forskjellige funksjoner som operatørene syntes var nyttig. PHP-scripta for SQL/Web - serveren har også blitt implementert fra MySQL til Microsoft SQL.

Simon Sundal la fram forslag om å få idéer for forbedringer på grensesnittet fra en annen robot på Kleven ved navn IGM. Han nevnte også at om flere problem ville oppstå, kunne gruppen eventuelt bruke et windows nettbrett som støtter protokollene kommunikasjonen går over. Men dette vil dermed føre til at gruppen må lage en helt ny applikasjon for windows.

Det ble bestemt å ha eit telefonmøte med IT-avdelingen på Kleven for å få satt opp SQL server og webserver etter behov fra det gruppen trenger.

Applikasjonen for å manipulere databasen ble planlagt å lage i løpet av den neste perioden. En enkel applikasjon laget i Java, hvor det er mulig å sette opp det som skal gjøres på de ulike robotene til en hver tid. Det vil også være mulig og printe ut lista.

Det vil bli et nytt besøk til Kleven i løpet av denne eller neste veke. Dette for å fullføre kommunikasjonen, og komme videre med å implementere funksjonaliteten i applikasjonen.

Til slutt ble det diskutert hvordan bachelor rapporten vil bli utført. Gruppen har startet på selve rapporten, og trengte litt veiledning i hvordan oppsettet skulle vere. Det ble også diskutert litt om innhold i hvert kapittel og hva som burde vere med som vedlegg.

Møtereferat

Den 17.04.15 var det møte med veileder angående bachelor oppgaven, på rom A433.

Tilstede: Simon Langlo, Anders Kobbevik, Audun Westerskow og Ottar L. Osen.

Denne perioden har gruppen lagt SQL databasen på en Raspberry Pi, laget siste utkast av applikasjon, lagt til funksjonalitet på knapper i applikasjonen og testet funksjonalitet opp mot robot på Kleven.

Møtet startet med presentasjon av applikasjon og backend program. Gruppen ville si seg ferdig med applikasjonen og konsentrere seg om rapport og testing ved Kleven, men vil ta siste finpuss på design og funksjonalitet den neste perioden.

Veileder testet applikasjonen, og kom med forslag til små endringer og eventuelle dynamiske knapper.

Gruppa følger planen og er fortsatt i rute. Det ble spurt litt spørsmål angående den endelige rapporten, og veileder påpeker at gruppen må skrive om situasjonen applikasjonen er i i dag, og forklare hva som kan forbedres for en eventuell bachelor gruppe neste år. For eksempel legge ved i rapporten flere forskjellige forslag til videre utvikling(konklusjon/drøfting).

Vedlegg 7

ID301702 Hovedprosjekt	Prosjekt Kleven Vision	Antall møter denne periode 1).	Firma - Oppdragsgiver Høgskolen i Ålesund / Kleven	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) UKE 3	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Audun, Anders og Simon	Dato 14.01.15

Hovedhensikt / fokus for arbeidet i denne perioden

Finne ut hvilket prosjekt vi skal gjøre. «Kleven Vision»-miniatyr eller «Kleven Robot Controller»

Planlagte aktiviteter i denne perioden

Definere oppgaven, first draft av GUI på «Robot Controller»-oppgaven.

Virkelig gjennomførte aktiviteter i denne perioden

Mer spesifikk oppgave

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Oppgave større enn først antatt.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Skulle egentlig utføre oppgaven full scale, men dette ble for stort for ett bachelor prosjekt. Endret dette fra å bruke 5 3D kamera til 1 3D kamera, og bruke ekornes robotene i kjelleren til høgskolen.

Hovederfaring fra denne perioden

- Innsyn i hvordan Kleven ville at vi skulle løse oppgaven.

Hovedhensikt/fokus neste periode

Skrive forprosjekt, og lage plan for prosjektet. Begynne å drøfte mulige løsninger for oppgaven.

Planlagte aktiviteter neste periode

- Lage prosjektplan
- Skrive forprosjektsrapport
- Finne ut og evt få tak i komponenter som trengst for prosjektet

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere
Anders Kobbevik	Simon Langlo Audun Westerskow

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

ID301702 Hovedprosjekt	Prosjekt Kleven Vision	Antall møter denne periode 1).	Firma - Oppdragsgiver Høgskolen i Ålesund / Kleven	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) UKE 3	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Audun, Anders og Simon	Dato 30.01.15

Hovedhensikt / fokus for arbeidet i denne perioden

Bli ferdig med forprosjektsrapport, og begynd å sette lære seg android programmering

Planlagte aktiviteter i denne perioden

Skrive forprosjektsrapport, starte med android app design

Virkelig gjennomførte aktiviteter i denne perioden

Forprosjektrapporten er skrevet, og vi er godt i gang med å lage første utkast av appen

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Prosjektet er i rute og vil fortsette etter planen

Hovederfaring fra denne perioden

- Vi har lært basic android programmering (fragments og activities)
- Design av app er godt i gang

Hovedhensikt/fokus neste periode

Det vi vil fokusere på neste periode er å få ferdig første utkast, slik at vi kan begynde å teste kommunikasjons funksjoner og forbedre designet.

Planlagte aktiviteter neste periode

- Lage en plan for all funksjonane som skal vere implementert i appen
- Fullføre første utkast av appen
- Få opp kommunikasjon mellom nettbrett og en pls

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere
Anders Kobbevik	Simon Langlo Audun Westerskow

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

ID301702 Hovedprosjekt	Prosjekt Kleven Vision	Antall møter denne periode 1).	Firma - Oppdragsgiver Høgskolen i Ålesund / Kleven	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) UKE 3	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Audun, Anders og Simon	Dato 26.02.15

Hovedhensikt / fokus for arbeidet i denne perioden

Lage funksjonalitet for applikasjonen

Planlagte aktiviteter i denne perioden

Lage en plan for funksjoner som skal implementeres.
Opprette en klasse for kommunikasjon mellom nettbrett og motoman robot.
Lage første utkast av applikasjonen.
Opprette en SQL database

Virkelig gjennomførte aktiviteter i denne perioden

Det er opprettet en klasse for kommunikasjon, mellom nettbrett og motoman robot.
Laget knapper og tilrettelagt for funksjoner som skal implementeres i applikasjonen.
En SQL database for testing er opprettet, denne kan nå skrives til og leses fra.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Etter å ha funnet en alternativ kommunikasjons metode, har mellom leddet med pls blitt droppet.
Data som vi planla å lagre i excel ark, skal nå lagres i sql database.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Prosjektet er i rute og vil fortsette etter planen

Hovederfaring fra denne perioden

- Lært om flere metoder for å skrive/lese til sql database.
- Utforsket måter å kommuniskere med robot
- Jobbet videre med app design.

Hovedhensikt/fokus neste periode

Det vi vil fokusere på neste periode er å fare på besøk til Kleven, der vi skal intervjue operatører og teste kommunikasjon mot en motoman robot.

Planlagte aktiviteter neste periode

- Test kommunikasjon mot motoman robot
- Forbedre design av applikasjon ved å utføre intervju og gå gjennom arbeidsprosessen selv
- Hente og skrive data til sql-database gjennom app'en
- lage flere utkast av grensesnitt design.

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere
Anders Kobbevik	Simon Langlo Audun Westerskow

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

ID301702 Hovedprosjekt	Prosjekt Kleven Vision	Antall møter denne periode 1).	Firma - Oppdragsgiver Høgskolen i Ålesund / Kleven	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) UKE 3	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Audun, Anders og Simon	Dato 18.03.15

Hovedhensikt / fokus for arbeidet i denne perioden

Få opp kommunikasjon mellom nettbrett og robot.

Planlagte aktiviteter i denne perioden

Få testet MotoNIS og eventuelt High Speed Ethernet.
Intervju av operatører.
Oppdatere applikasjon etter behov fra operatører.
Gjør om fra MySQL til Microsoft SQL

Virkelig gjennomførte aktiviteter i denne perioden

MotoNIS er testet, men vi støtte på problem når vi skulle legge til MotoNIS biblotekene i Android. Det ble derfor bestemt å teste ut High Speed Ethernet, for å ha en reserveplan om MotoNIS ikke fungerer. Har gjort om php-scriptene fra MySQL til Microsoft SQL.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

Måtte endre SQL server, fra MySQL til Microsoft SQL. Har laget reserveplan for kommunikasjonen dersom MotoNIS ikke fungerer.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Prosjektet er i rute og vil fortsette etter planen

Hovederfaring fra denne perioden

- Lært måter å bruke MotoNIS og High Speed Ethernet på. Testklasser har blitt testet.
- Satt opp microsoft database og gjort om php-script fra MySQL til Microsoft SQL.
- Forbedret design og funksjonalitet i applikasjon.

Hovedhensikt/fokus neste periode

Neste periode vil bli brukt til å sette opp SQL databasen på Kleven. Det vil også bli lagt inn funksjonalitet som jobblister og diverse fra robot til noen av knappene. Kanskje få testet den tradisjonelle jogging metoden fra nettbrett.

Planlagte aktiviteter neste periode

- Implementering av knapper
- Implementere SQL database på Kleven.
- Ferdigstille kommunikasjon. Enten MotoNIS eller High Speed Ethernet
- Begynne på plakat

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere
Anders Kobbevik	Simon Langlo Audun Westerskow

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

ID301702 Hovedprosjekt	Prosjekt Kleven Vision	Antall møter denne periode 1).	Firma - Oppdragsgiver Høgskolen i Ålesund / Kleven	Side 1 av 1
Rapport fra prosess Framdriftsrapport	Periode/uke(r) UKE 3	Antall timer denne per. (fra logg)	Prosjektgruppe (navn) Audun, Anders og Simon	Dato 14.01.15

Hovedhensikt / fokus for arbeidet i denne perioden

SQL database på Kleven, funksjonalitet i knapper og jogging av robot

Planlagte aktiviteter i denne perioden

- Implementering av knapper
- Implementere SQL database på Kleven.
- Ferdigstille kommunikasjon. Enten MotoNIS eller High Speed Ethernet
- Begynne på plakat

Virkelig gjennomførte aktiviteter i denne perioden

Funksjonalitet på alle knappene fungerer, mens jogging ikke er helt ferdig. Har ferdigstilt kommunikasjonen, og vi gjekk for High Speed Ethernet. Databasen har ikke blitt satt opp på Kleven, men det ble bestemt å bruke en Raspberry PI som server med databasen på som bare ligger lokalt i robotskapet pga. problemer med internett tilgang for roboten.
Har også laget en «Back end»-applikasjon for å manipulere databasen fra en pc.

Beskrivelse av/begrunnelse for eventuelle avvik mellom planlagte og virkelige aktiviteter

SQL databasen, vil ligge på en raspberry PI i første omgang, pga. problemer med internett tilgang for roboten.
Prøver å forbedre design av applikasjon.

Beskrivelse av /begrunnelse for endringer som nå ønskes i selve prosjektets innhold eller i den videre framgangsmåten - eller framdriftsplanen

Prosjektet er i rute og vil fortsette etter planen

Hovederfaring fra denne perioden

- Forbedret design og funksjonalitet i applikasjon
- «Back End» applikasjon klar og fungerer

Hovedhensikt/fokus neste periode

Ferdigstilling og testing hos Kleven. Jogging av robot via IO. Skrive rapport og lage plakat.

Planlagte aktiviteter neste periode

- Implementere knapper og funksjoner
- Teste funksjoner og finne forbedringer
- Robot Programmering
- Bachelor rapport

Annet

Ønske om /behov for veiledning, tema i undervisningen – drøfting ellers

Godkjenning/signatur gruppeleder	Signatur øvrige gruppedeltakere
Anders Kobbevik	Simon Langlo Audun Westerskow

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

Hovedprosjekt

28.mai.2015

http://

HIALS

Prosjektleder	
Prosjekt start/slutt datoer	14.jan.2015 - 30.mai.2015
Avsluttet	100%
Oppgave	22
Deltakere	0

Oppgave

Navn	Startdato	Sluttdato
Definere Oppgave	14.01.15	16.01.15
Forprosjekts Rapport	14.01.15	30.01.15
Komponent valg / bestilling	14.01.15	06.02.15
Utvikle Android App i Java	14.01.15	24.04.15
Sette seg inn i fragments og activities	14.01.15	30.01.15
Plan for funksjoner som skal implementeres	26.01.15	06.02.15
Lage første utkast av grafisk brukergrensesnitt	26.01.15	13.02.15
Implementering av knapper og funksjoner	16.02.15	17.04.15
Finne løsninger for lesing og skriving til enhetsminne	16.02.15	27.03.15
Automatisk innlesing av "robot calls" fra SQL database	02.03.15	13.03.15
Lage jobbjournal og lagre i SQL database	16.03.15	27.03.15
Implementere SQL database	16.02.15	27.02.15
Siste utkast av grafisk brukergrensesnitt	07.04.15	13.04.15
Teste funksjoner og finne forbedringer	13.04.15	24.04.15
Kommunikasjon	02.02.15	17.04.15
Lage klasse for kommunikasjon mellom tablet og Motoman Robot	02.02.15	20.02.15
Teste Kommunikasjon med Motoman Robot	16.02.15	27.02.15
Lage klasse for alternativ robot kommunikasjon	02.03.15	20.03.15
Teste forskjellige kommunikasjons metoder mot robot	23.03.15	17.04.15
Robot Programering	07.04.15	24.04.15
Montering hos Kleven + testing	20.04.15	01.05.15
Bachelor rapport	02.02.15	29.05.15

Ganttskjema

Navn	Startd...	Sluttdato
Definere Oppgave	14.01.15	16.01.15
Forprosjekts Rapport	14.01.15	30.01.15
Komponent valg / bestilling	14.01.15	06.02.15
Utvikle Android App i Java	14.01.15	24.04.15
Sette seg inn i fragments og activities	14.01.15	30.01.15
Plan for funksjoner som skal implementeres	26.01.15	06.02.15
Lage første utkast av grafisk brukergrensesnitt	26.01.15	13.02.15
Implementering av knapper og funksjoner	16.02.15	17.04.15
Finne løsninger for lesing og skriving til enhetsminne	16.02.15	27.03.15
Automatisk innlesing av "robot calls" fra SQL database	02.03.15	13.03.15
Lage jobbjournal og lagre i SQL database	16.03.15	27.03.15
Implementere SQL database	16.02.15	27.02.15
Siste utkast av grafisk brukergrensesnitt	07.04.15	13.04.15
Teste funksjoner og finne forbedringer	13.04.15	24.04.15
Kommunikasjon	02.02.15	17.04.15
Lage klasse for kommunikasjon mellom tablet og Motoman R...	02.02.15	20.02.15
Teste Kommunikasjon med Motoman Robot	16.02.15	27.02.15
Lage klasse for alternativ robot kommunikasjon	02.03.15	20.03.15
Teste forskjellige kommunikasjons metoder mot robot	23.03.15	17.04.15
Robot Programering	07.04.15	24.04.15
Montering hos Kleven + testing	20.04.15	01.05.15
Bachelor rapport	02.02.15	29.05.15