Norges Teknisk Naturvitenskapelige Universitet

Eksperter i Team

Instrumentering og styring over Internett

Prosjektrapport gruppe 3

Forfattere:
Torbjørn Aase
Andreas Bertheussen
Rune Holmgren
Sigurd Holsen
Jostein Munz

Veileder:
Sven Fjeldaas

30. april 2014

Sammendrag

Denne rapporten presenterer et verktøy for å visualisere og styre en Dynamixel robotarm basert på målinger av vinkler og kjente konstante parametere. Robotarmens kinematikk er blitt modellert, og inverskinematikken er dokumentert. Brukergrensesnittet for produktet er en arbitrær nettleser som kommuniserer med en server over internett. C++ med rammeverket Qt har blitt brukt i tillegg til JavaScript og WebGL.

Video av systemet: `http://youtu.be/avh6Fbj8tRQ`

Innhold

1 Innledning

Over flere tiår har det blitt utviklet en rekke styresystem til roboter. Mangfoldige millioner kroner har blitt brukt på forskning innen dette feltet, og industrien har kommet langt på dette området. I applikasjoner hvor operatøren ikke kan se den fysiske robotarmen er kamera og målinger av vinklene de mest brukte verktøyene for visualisering. Denne rapporten tar for seg en alternativ visualiseringsmetode hvor en operatør vil se en 3D-modell som kontinuerlig beskriver posisjon og orientasjon til robotarmen. Operatøren har mulighet til å visualisere både faktisk og ønsket posisjon av robotarmen ved at den ønskede posisjonen vises som en transparent modell, og den faktiske modellen vises som en solid modell.

2 Problemstilling

Prosjektet har krav om at det er tverrfaglig, involverer styring over internett og at det skal ha stor grad av samfunnsnytte.

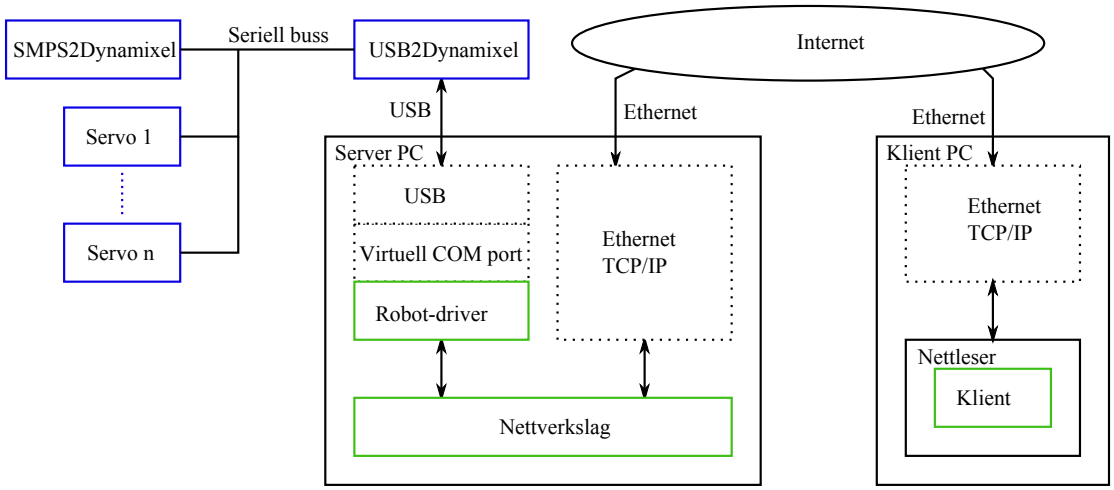
Prosjektet er tverrfaglig med tanke på at det krever kompetanse innenfor både robotikk, hardware, programvare og 3D-grafikk. Styring av robotarmen skjer over internett og implementeres i JavaScript på klientsiden og C++ på serversiden. Prosjektet vil være aktuelt både i industrielle og akademiske applikasjoner. Det vil være mest aktuelt hvor robotmanipulatorer opererer i ukjente omgivelser hvor et kamera ikke vil gi tilstrekkelig informasjon og det stilles høye krav til presisjon. Eksempel på dette vil være subsea-operasjoner eller operasjoner på en satellitt.

3 Systembeskrivelse

Komponentene som gruppa hadde til rådighet, og som dannet utgangspunktet for prosjektet, er et servomotorsystem kalt Dynamixel[1], produsert av ROBOTIS. Servomotorene og tilhørende komponenter er vist i blå farge i figur 1, og utgjør "roboten". Roboten kan i utgangspunktet ikke styres direkte fra internett, men kan styres via et serielt bussgrensesnitt, eller med en overgang til USB. I tillegg til roboten inngår derfor en server-PC som skal gjøre servomotorene tilgjengelige over internett.

Klienter som ønsker å styre roboten benytter en nettside vist i en nettleser. Denne nettsiden kommuniserer med serveren direkte, og presenterer et grensesnitt der brukeren kan styre og visualisere roboten. I prinsipp kan flere klienter koble seg til systemet samtidig.

Ikke vist i figur 1 er hvordan nettsiden leveres til klienten. Dette kan skje lokalt fra klienten sitt filsystem, fra server-PC-en, eller fra en tredje datamaskin som fungerer som en dedikert webserver.



Figur 1: Systemdiagram

3.1 Robotkomponenter

Følgende utstyr har blitt benyttet:

Figur 2: Dynamixel AX-12+ servomotor, USB2Dynamixel og SMPS2Dynamixel

AX-12+ [2] er en servomotor som kan rotere til en bestemt vinkel mellom 0°og 300°, med en oppløsning på 0.29°(1024 steg). Alternativt kan den settes i hjulmodus uten vinkelbegrensing, men der man kun kan angi rotasjonshastighet.

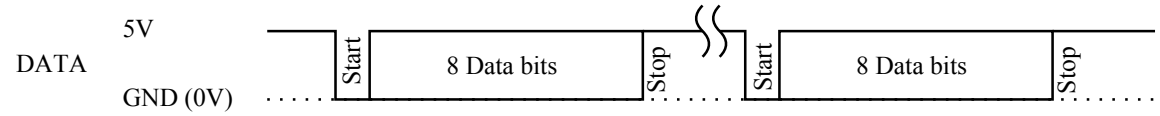
USB2Dynamixel er en komponent som fungerer som grensesnitt mellom servomotorene og en PC. Denne konverterer signaler mellom den serielle bussen der servoene er koblet sammen, og en virtuell serieport for PC-en.

SMPS2Dynamixel gjør det mulig å bruke en 12V DC nettadapter til å forsyne servoene med strøm.

Komponenter i dynamixelsystemet kobles sammen via et *daisy chain*. Det vil si at komponentene er koblet sammen på samme buss, der hver enhet har to konnektorer. Én konnektor for å

kople seg til bussen, og én konnektor for å koble neste enhet til bussen. Dynamixel-bussen består kun av tre ledere; VDD, DATA og GND. På DATA-linjen vil PC og servomotorer utveksle informasjon via *TTL level multi-drop UART*.

Signaleringen for en UART er vist i figur 3. Når det ikke kommuniseres på bussen, holdes spenningen ved 5V. Under sending vil først en start bit (0V) sendes, etterfulgt av 8 data bit, og avsluttes med ett stopp-bit (5V). Hensikten med start-bittet er å synkronisere mottaker til senderens sendefrekvens. Hensikten med et stopp-bit er å returnere linjen til 5V. *TTL* refererer til spenningsnivåene 0V og 5V, som er brukt. *Multi-drop* vil si at det er flere mottakere som er koblet på samme buss.



Figur 3: Signalering for en UART

Datamaskinen som styrer systemet sender ut en rekke bytes (en pakke) serielt på DATA-linjen. Hver enhet på bussen har en unik ID, og dersom pakken er adressert til enheten sin egen ID, vil enheten kunne svare. Pakker inneholder instruksjoner for enten lesing eller skriving til minnelokasjoner. Verdien som lagres i hver minnelokasjon har en betydning avhengig av enheten som er koblet til. For AX-12+ for eksempel, kan man skrive til adresse 30 og 31 for å sette ønsket vinkel.

3.2 Qt

For å sikre kryssplattform kode ble Qt valgt som C++-rammeverk. Qt er en samling C++-bibliotek som er plattformuavhengig. Det vil si at kode som utvikles for Qt-rammeverket krever minimale endringer for å kunne kjøre på andre plattformer. De forskjellige bibliotekene fungerer også godt med hverandre, og man får en konsistent kodebase. Qt-versjonen som ble brukt er 5.2, men programmet burde også kompilere på senere versjoner.

En annen fordel med å bruke Qt, er at siden det allerede inneholder biblioteker for svært mange anvendelser, er det ikke nødvendig å lete etter et eget bibliotek for hver enkelt anvendelse. Qt sine biblioteker er allerede tilgjengelige, og godt dokumentert.

3.3 CMake

Store prosjekter med mange kildefiler får ofte en komplisert kompileringsprosess. For å sikre at prosjektet blir bygget på samme måte hver gang, benytter man seg ofte av et byggesystem. Det er to byggeverktøy som har offisiell støtte for Qt: QMake og CMake.

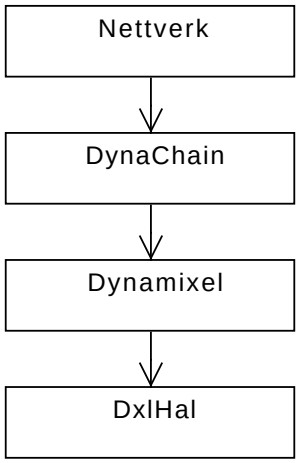
CMake ble valgt siden det er nyere, mer brukt, og mer fleksibelt. Prosjektet genererer flere kjørbare filer, flere testprogrammer og selve serveren. Her var CMake sin løsning mer elegant, og det som var utslagsgivende.

3.4 Robotdriver

Dynamixel har lagt ut et eksempelbibliotek til Windows og Linux som gjør det mulig å kople seg opp til roboten og sende pakker. Qt har et bibliotek som kan gjøre dette kryssplattform, kalt `QtSerialPort`. Utgangspunktet var da at biblioteket levert av leverandøren skulle skrives om til Qt ved hjelp av dette. Dette viste seg å være utfordrende, og det ble i stedet bestemt at eksempelbiblioteket skulle brukes. CMake gjør det mulig å ha *if*-betingelser for plattform, og gjorde det dermed lett å velge mellom plattformversjonene ved kompilering.

3.4.1 Arkitektur

Robotdriveren er lagdelt i fire moduler (se figur 4). Lagdelingen er gjort slik at hver modul kun kommuniserer med laget som ligger rett nedenfor. Dette sikrer en klar fordeling av ansvar, og gjør koden mer ryddig.



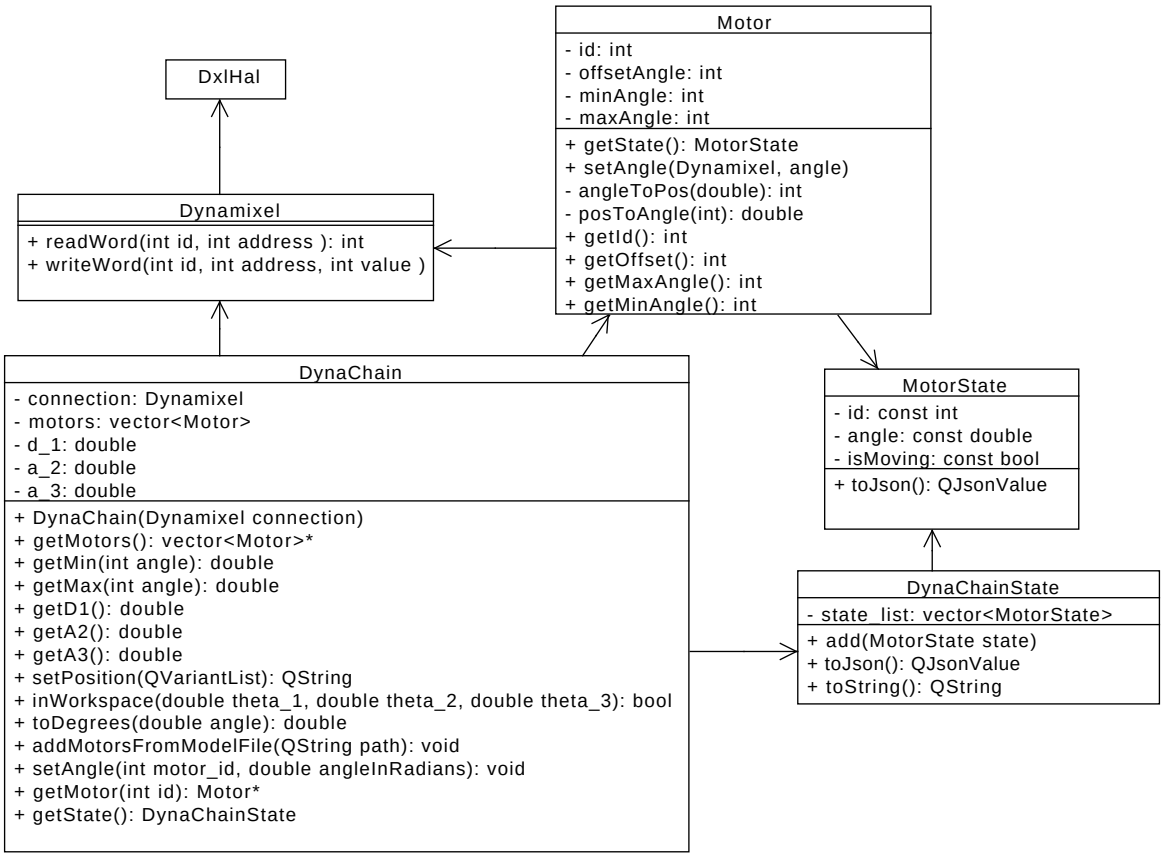
Figur 4: Lagdeling for serverprogramvaren

Det nederste laget er kalt for `DxlHal`. Dette er det laget som tar seg av det plattformspesifikke med å sende kommando-pakker over en serieport.

Modulen over dette kalles for `Dynamixel`. Denne er ikke plattformspesifikk, og er på et høyere abstraksjonsnivå enn `DxlHal`. Mens `DxlHal` sender én og én pakke, kan `Dynamixel` sende mer sammensatte pakker. Til sammen utgjør `DxlHal` og `Dynamixel` den koden lagt ut av Dynamixel.

Det nest øverste laget er kalt `DynaChain`, der all robot-logikk ligger. Et klassediagram for robotdriveren kan sees i figur 5. Det er ikke blitt brukt arv, og alle piler i figuren representerer bruk.

Det øverste laget er nettverkslaget som tar seg av sending og mottaking over internett.



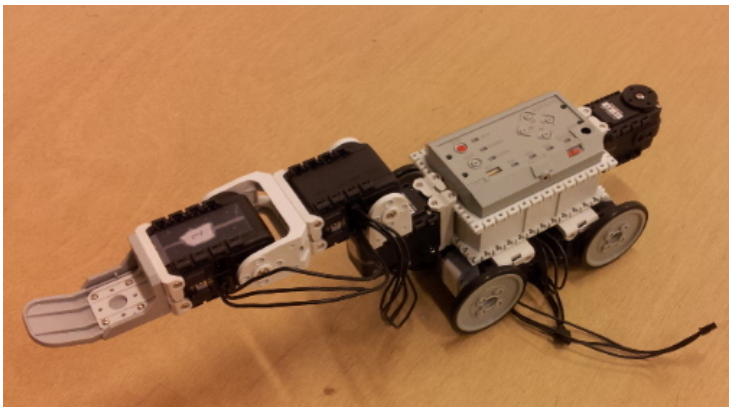
Figur 5: Klassediagram over `DynaChain`

3.4.2 DynaChain

`DynaChain` er den modulen som inneholder all robotlogikk. Denne modulen har i hovedsak to oppgaver: å hente ut og endre motorposisjonene.

Den enkleste klassen i hierarkiet er `Motor`-klassen. Denne klassen innkapsler *en* enkelt motor. Det som er felles for hver motor er at de har en motor-ID, en maks-vinkel, en minimum-vinkel og et vinkel-avvik. Motor ID-en er den ID-en motoren har som sikrer at meldinger kommer frem til rett motor. Som nevnt tidligere, er posisjonen til motoren gitt ved et heltall mellom 0 og 1023. For å gjøre det lettere å tolke disse tallene, ble det bestemt at det skulle brukes grader der 0 grader på alle motorer skulle representere posisjonen vist i figur 6. For å implementere dette trengs derfor et vinkelavvik som gir forskjellen i vinkel mellom posisjon 0 og vinkel 0. Dette

kan betraktes som en slags kalibrering og er avhengig av robotarmen som jobbes med. De to resterende vinkel-variablene, maks-vinkel og min-vinkel, angir hvilke vinkler som er gyldige.



Figur 6: Posisjon til manipulatorarmen når alle vinkler er satt til 0

For å håndtere en serie med slike motorer, brukes klassen `DynaChain`. `DynaChain` inneholder en liste med `Motor`-objekter, samt metoder for å hente ut og styre alle motorer samtidig. Den viktigste metoden her er `setAngle(motorid, vinkel)`, som tar inn motorid og vinkel, og setter motoren til denne posisjonen.

For å hente ut tilstanden til roboten brukes to klasser: `DynaChainState` og `MotorState`. Disse klassene er *immutable*, som vil si at et objekt av disse klassene ikke kan endres etter at objektet er initialisert. Dette forsikrer mottageren av et slikt objekt om at objektet aldri har blitt endret.

For å hente ut disse tilstandene brukes metoden `getState()`, som er implementert for både `Motor` og `DynaChain`, og som henholdsvis returnerer et `MotorState`- og `DynaChainState`-objekt. Begge klassene har metoden `toJson()` for å representere tilstanden som en JSON-streng, slik at den lett kan dekodes av klienten.

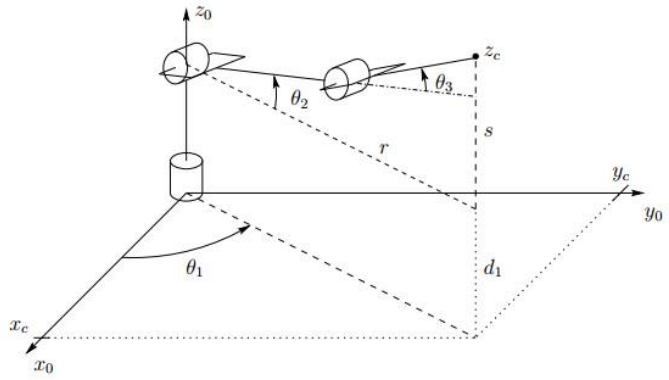
3.5 Kinematikk

Robotarmen ble modellert og parameterisert i tråd med Denavit-Hartenberg konvensjonene.

Link	a_i	α_i	d_i	θ_i
1	0	90°	0.025m	θ_1^*
2	0.068m	0	0	θ_2^*
3	0.1m	0	0	θ_3^*

Tabell 1: DH tabell

Følge figur illustrerer koordinatsystemet. x_0 går langs kjøreretningen til fartøyet som robotarmen er festet på.



Figur 7: Koordinatsystem for robotarmen. Figur hentet fra [3, p. 90]

Foroverkinematikken beskriver robotarmens endepunkt som funksjon av vinklene.

$$
\begin{aligned}
x_e &= \cos\theta_1(d_2\cos\theta_2 + d_3\cos(\theta_2+\theta_3)) \\
y_e &= \sin\theta_1(d_2\cos\theta_2 + d_3\cos(\theta_2+\theta_3)) \\
z_e &= d_1 + d_2\sin\theta_2 + d_3\sin(\theta_2+\theta_3)
\end{aligned}
$$

Hvor x_e. y_e og z_e definerer koordinatene til endepunktet på armen som funksjon av vinklene.

Inverskinematikk beskriver, gitt et visst koordinat, hvordan vinklene må settes for at enden av robotarmen skal oppnå den gitte posisjonen. Fra [3, p. 99].

$$
\begin{aligned}
\theta_1 &= \text{atan2}(y_c, x_c) \\
\theta_2 &= \text{atan2}(z_c - d_1, \sqrt{x_c^2 + y_c^2}) - \text{atan2}(a_3\sin\theta_3, a_2 + a_3\cos\theta_3) \\
\theta_3 &= \text{atan2}(\pm\sqrt{1-D^2}, D) \\
D &= \frac{x_c^2 + y_c^2 + (z_c - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3}
\end{aligned}
$$

D må da være mindre eller lik 1 for at løsningen skal være innenfor robotens gyldighetsområde. Det er i tillegg begrensninger på vinklene basert på motordriverne sine begrensninger. Verdiene for disse ligger i model.json-filen. Dersom både x_c og y_c settes til null, eksisterer det uendelig antall løsninger til problemet da verdien for θ_1 er arbitrær i løsningen. θ_1 blir da satt til null. For alle andre inputs av x_c og y_c vil man få to løsninger på problemet. Begge løsningene vil bli sjekket, og den første som finnes i løsningsrommet blir valgt. Det sendes en feilmelding tilbake til klienten dersom den forsøker å gi en posisjonskommando som ikke er oppnåelig.

3.6 Nettverkslag

3.6.1 Valg av transportlag

For å gjøre det mulig å bruke webapplikasjonenen til å styre roboten over internett måtte det velges en egnet teknologi. Styring over *internett* begrenser oppgaven i praksis til IP-basert (Internet Protocol) kommunikasjon. Fordi det var ønskelig å kunne kommunisere fra en webapplikasjon/nettleser, var valgmulighetene videre begrenset. Nettlesere i dag støtter JavaScript, som gjør det mulig å programmere oppførselen til nettsiden, samt diverse plugins for multimedia og mer avansert programvare.

Forskjellige plugins er tilgjengelige til nettlesere for å kommunisere over nettverk. Så godt som alle nettlesere har Flash installert. Flash støtter både TCP og UDP. Java er også et annet alternativ der man kan benytte både TCP og UDP.

Dersom man ikke ønsker å bruke en nettleser-plugin, finnes det to andre valg. Begge disse kan aksesseres gjennom JavaScript:

HTTP-forespørsler innebærer at klienten må initiere alle overføringer[4]. Denne funksjonaliteten er innebygget i nettleseren sin JavaScript-funksjonalitet, men er mer egnet for å hente data fra en nettside, og ikke en robot. For roboten var det ønskelig å kontinuerlig motta siste tilstand fra roboten slik at dette kan vises. Dersom klienten hele tiden må sende HTTP-forespørsler (polling), blir dette tungt. Fordelen med HTTP-forespørsler er at trafikken slipper gjennom de aller fleste proxyer og brannmurer, fordi den er lik vanlig nettsidesurfing.

Et eksempel på HTTP er vist i figure 8.

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Figur 8: Eksempel på HTTP

WebSockets [**?**] tilbyr toveis kommunikasjon, og bygger på TCP. Opprettelsen av en WebSocket likner på en HTTP-forespørsel, men tilkoblingen blir ikke lukket når forespørselen er over. Tilkoblingen fortsetter å være åpen, og klient og server kan utveksle meldinger (frames). WebSockets ser ut som HTTP-forespørsler, og kan slippe gjennom brannmurer, men ikke alle proxyer. WebSockets er standardisert av IETF, og er støttet av alle de populære nettleserne.

Å bruke HTTP-forespørsler ville innebære enten å tilpasse en eksisterende server, eller å skrive vår server fra grunnen av. En slik server vil være mer omfattende enn en WebSocket-server. HTTP i seg selv tilfører ikke noe nyttig funksjonalitet utover det å være tilgjengelig overalt, så valget falt på WebSockets.

Opprettelsen av en WebSocket består av følgende type forespørsel fra klienten:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Deretter følger toveis utveksling av *frames*, der hvert frame består av en header etterfulgt av data som skal overføres. Innholdet i disse defineres av applikasjonslaget.

3.6.2 Valg av applikasjonslag

For enkelhets skyld ble det valgt å opprette to toveis forbindelser mellom hver klient og serveren. Den ene forbindelsen brukes til styring av roboten, der man kan sende kommandoer og motta responsen på kommandoene (styringsforbindelse). Den andre forbindelsen (tilstandsforbindelse) vil serveren kontinuerlig strømme informasjon om robotarmens tilstand, slik at klienten kan oppdatere visningen sin. Å gjøre dette i to forskjellige forbindelser gjør implementasjonen enklere, og det er naturlig fordi det er to vidt forskjellige typer trafikk.

Tilstandsforbindelsen blir kun brukt i én retning, fra server til klient. Det ble også sett på som fordelaktig å formatere data som JavaScript Object Notation (JSON), ettersom dette er enkelt å dekode i JavaScript på klienten, og det er tilgjengelig i Qt-rammeverket.

Et eksempel på JSON-formatert data sendt fra server til klient er vist i figur 9. Dataen som er vist er representert som et JSON array med tre objekter der hvert av objektene representerer en servomotor. Servomotorene inneholder en id og en tilstand. For servomotorene ble det hentet ut to tilstander: vinkel og hvorvidt motoren var i bevegelsesmodus eller ikke.

```
[
    {
        "id": 1,
        "state": {
            "angle": -42.1875,
            "is_moving": false
        }
    },
    {
        "id": 2,
        "state": {
            "angle": 24.66796875,
            "is_moving": false
        }
    },
    {
        "id": 3,
        "state": {
            "angle": -0.29296875,
            "is_moving": false
        }
    }
]
```


Figur 9: Eksempel på JSON-formatert data sendt fra server til klient

For styringsforbindelsen var det ønskelig å tilby APIet til robot-driveren, som består av en rekke funksjoner. Nettverkslaget burde være så tynt som mulig, og tilby dette APIet til klienten (over internett). Det var derfor naturlig å gjøre det i form av RPC (Remote Procedure Calls).

Det vil si at klienten (i JavaScript) kan kalle `setAngle(45, 3)` for å sette vinklen på servo med id 3, til 45 grader. Informasjonen om dette kallet ('setAngle') og dets parametere (3, 45) overføres over WebSocket, og på mottakersiden tolkes det og utføres på serveren. Eventuell returverdi for prosedyrekallet sendes så tilbake. Det finnes flere RPC-standarder, blant annet JSON-RPC[6] og SOAP[7]. I JSON-RPC og SOAP formateres informasjonen om prosedyrekallet og parameterne som henholdsvis JSON eller XML. JSON er kompakt og godt støttet i både JavaScript og C++, mens SOAP er mer kompleks og dermed vanskeligere å implementere. På grunnlag av dette ble JSON-RPC valgt.

Et eksempel på et JSON-RPC-kall og respons fra `setAngle(45, 3)` er vist under. `-->` indikerer data sendt fra klient til server, og `<--` er det motsatte.

```
--> {"jsonrpc": "2.0", "method": "setAngle", "params": [45, 3], "id": 1}
<-- {"jsonrpc": "2.0", "result": 0, "id": 1}
```

JSON-RPC støtter også notifications, som er kall uten returverdi. Kallet vil da ikke inkludere et

``i''-felt.

3.6.3 Implementasjon av nettverkslag

Fordi robot-driveren var skrevet i C++/Qt, var det naturlig å skrive serveren i noe kompatibelt. Alternativene som ble vurderte var QtWebsocket (C++/Qt)[**?**], WebSocket++ (C++/Boost)[**?**] og libWebSockets (C)[**?**].

Det ble avgjort at QtWebSockets tilbød ønsket funksjonalitet, og kom med flere eksempler - både for servere og klienter. QtWebsocket tilbyr klassene `QWsServer` og `QWsSocket`. En `QWsServer` kan instanseres og bli bedt om å lytte på en gitt IP-adresse og port.

For å implementere JSON-RPC fantes flere muligheter; oppe til vurdering var JsonRpc-Cpp[11] og QJsonRpc[12]. Førstnevnte er minimal og burde være enkel å bruke. Ulempen er at den bruker et tredjeparts Json-bibliotek. qjsonrpc er en Qt-basert implementasjon, som baserer seg på `QTCPServer` og `QTCPSocket` fra Qt. Det er mer komplekst enn JsonRpc-Cpp, men fordi det er bygget på Qt, er det lettere å integrere med en Qt-applikasjon.

QJsonRpc-biblioteket tilbyr fra før en `QJsonRpcTcpServer`-klasse. Denne benytter en `QTCPServer`, og tillater flere TCP-tilkoblinger. For å støtte WebSockets ble det valgt å utvide biblioteket med en ny klasse, QJsonRpcWebSocketServer som benytter `QWsServer` i stedet for `QTCPServer`. For å gjøre dette mulig, måtte `QWsSocket` endres for å oppføre seg mer som en `QTCPSocket`. Dette innebar å alltid emittere signalet `readyRead()` når et WebSocket-frame er mottatt, og å implementere `readData()` og `writeData()` som kan sende og motta WebSocket-frames. En `QJsonRpcWebSocketServer` instansieres i fila `main.cpp`.

For tilstandsforbindelsen ble Qt sine funksjoner benyttet for JSON-formatering sammen med `QWsServer`. Denne er implementert i klassen `Server`, som instansieres i fila `main.cpp`.

3.6.4 Implementasjon for web-klient

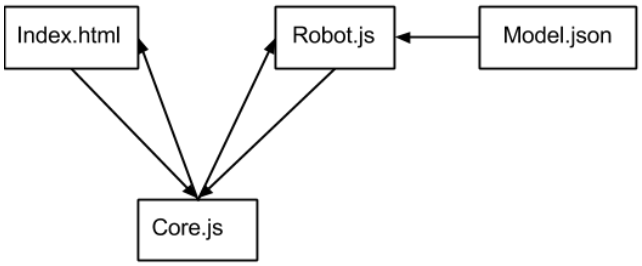
Web-klienten utvikles i JavaScript. WebSockets er tilgjengelig gjennom `WebSocket()`-konstruktøren [**?**], og dette APIet er implementert i alle de mest brukte nettleserne Internet Explorer, Firefox, Chrome, Safari og Opera[14][15]. Det er derfor ikke nødvendig å anskaffe et eget bibliotek for WebSockets.

Denne implementasjonen ble bygget videre på, og innkapslet i klassen `Ws`. Denne tilbyr mer funksjonalitet for debugging av WebSockets, som å logge nettverkstrafikken, og skrive informasjon til konsoll når tilstanden til en WebSocket endrer seg. Denne klassen er til dels basert på eksempelkode inkludert i QtWebSocket.

JSON-RPC-funksjonalitet oppnås ved hjelp av et tredjepartsbibliotek kalt jQuery JsonRpcClient[16], som bygger på det populære jQuery-rammeverket.

3.7 Web-applikasjon

Web-applikasjonen tillater en operatør å fjernstyre robotarmen samtidig som han/hun får tilbakemelding om armens posisjon gjennom en 3D-visualisering av leddene i armen. Når operatøren laster nettsiden kobler applikasjonen automatisk til server-applikasjonen som gir kontroll over, og informasjon om, robotarmen.

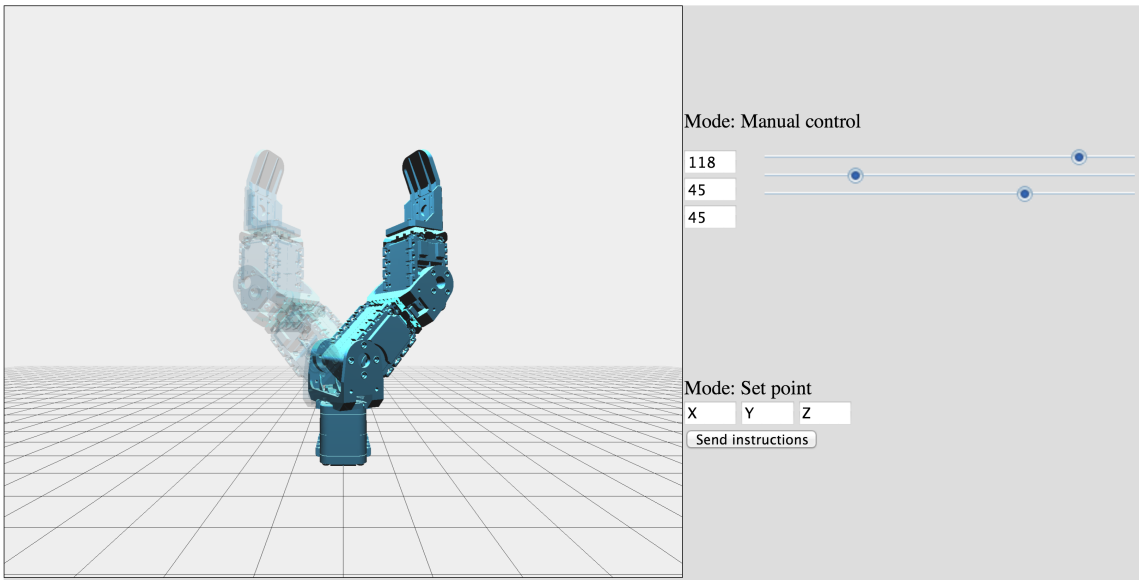


Figur 10: Arkitektur for web-klient.

Web-applikasjonen i seg selv er bygd opp etter en variant av MVC (Model-View-Controller), med filene `Robot.js` (model), `index.html` (view) og `Core.js` (controller). `Robot.js` blir brukt til å generere to modeller, `control_robot` og `real_robot`, som henholdsvis representerer ønsket tilstand og faktisk tilstand på robot-armen. Modellene baserer seg på filen `model.json` som redegjør for antall ledd og maksimum/minimum vinkel for aktuatorene i armen. `Index.html` er nettsiden operatøren forholder seg til for å gi kommandoer og få feedback fra roboten. Input og output blir så håndtert av `Core.js` som er ansvarlig for å holde modellene oppdaterte med informasjon fra operatøren og serveren.

Ved å implementere klienten som en webtjeneste, oppnår systemet med en gang en svært høy grad av multiplattformstøtte. De fleste moderne nettlesere på PCer, nettbrett og mobiltelefoner støtter tjenesten. Dette er en styrke som gjør at det er enklere for bedrifter å adoptere løsningen som et tillegg til allerede eksisterende løsninger.

3.7.1 Fjernstyring av roboten



Figur 11: Det grafiske brukergrensesnittet til web applikasjonen.

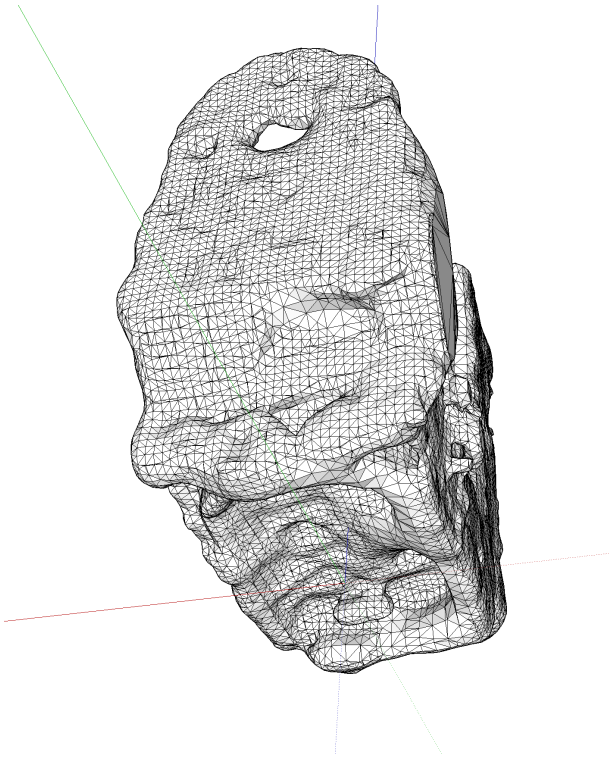
Figur 11 viser grensesnittet operatøren forholder seg til ved bruk av systemet. Gjennom kontrollpanelet har operatøren to valg for styring. *Manual control* tillater manuell styring av hver aktuator i sanntid gjennom bruk av slidere. *Set point* lar operatøren sette koordinater for ett enkelt punkt. Når operatøren så trykker på *Send instructions* vil koordinatene bli sendt til serveren og vinkler på aktuatorene regnet ut på server-siden slik at enden på robot-armen treffer punktet. Operatøren har altså valget mellom *Manual control* for responsiv kontroll i sanntid eller *Set point* for tilfeller hvor det er viktig å nå et nøyaktig punkt.

Til venstre for kontrollpanelet kan operatøren se visualiseringen av robot-armen i 3D. I figuren vises to 3D-modeller av robot-armen. Dette er 3D-representasjoner av de tidligere nevnte `real_robot` og `control_robot`. Den solide modellen viser operatøren hvordan robot-armen faktisk er posisjonert, samtidig som den transparente modellen viser hvordan robot-armen bør være posisjonert basert på det input som er gitt. I eksempelet vist i figuren kan da operatøren klart se at noe hindrer den ene aktuatoren i å fullføre den ønskede rotasjonen, f.eks. en obstruksjon eller en feil ved aktuatoren, og kan da iverksette tiltak. Ved normal drift vil modellene alltid overlappe, siden ønsket posisjon og faktisk posisjon vil være er like.

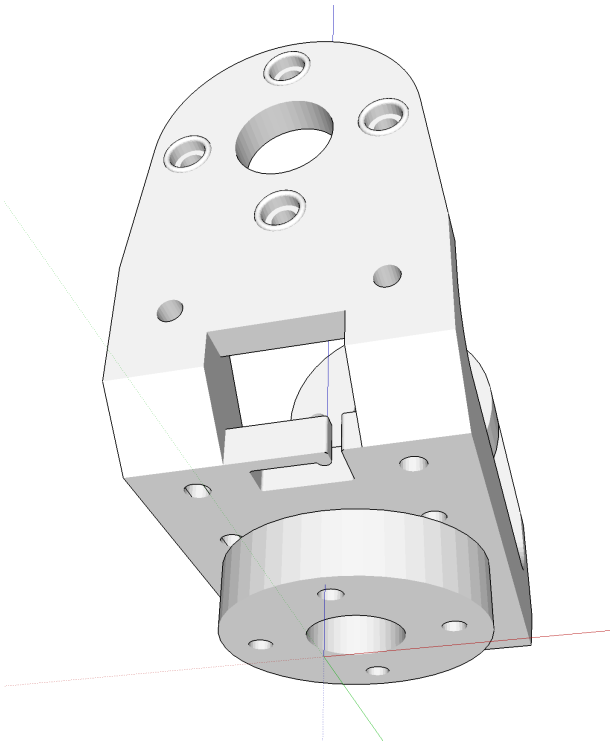
3.7.2 3D-modeller

Brukbarheten til systemet avhenger i stor grad av at operatøren av roboten får et mest mulig korrekt bilde av utstyret sitt. For å oppnå dette var det viktig å skaffe 3D-modeller som på en realistisk måte representerte den fysiske roboten. Dette ble gjort ved å benytte en MakerBot Digitizer 3D-skanner. Hvert enkelt ledd på roboten ble scannet hver for seg, slik at leddene

mellom dem kan manipuleres digitalt i ettertid ved hjelp av Google SketchUp. Ved å finpusse på 3D-modellene fra skanningen for hånd ble sluttproduktet 3D-modeller som var svært like den faktiske roboten. Disse modellene ble konvertert fra `.obj`-format til et JavaScript-basert format som kunne benyttes i webklienten.



Figur 12: Rådataene som de kom fra 3D-skanneren.



Figur 13: Den ferdige modellen slik den framstår i webapplikasjonen.

3.7.3 3D framvisning

Som nevnt tidligere var det ønskelig å implementere systemet som en webapplikasjon, og med dette få en høy grad av multiplattformstøtte. Dette innebar også at det ble nødvendig å jobbe med 3D-grafikk i nettleseren. Dette har heldigvis blitt enklere og enklere de siste årene, med HTML5, JavaScript og WebGL. Disse utvidelsene er nå er tilgjengelig i nesten alle nettlesere. For å gjøre utviklingen av løsningen enklere, ble biblioteket THREE.js benyttet. Det gir en økt grad av abstraksjon fra de WebGL-kallene som nettleserne tilbyr, og lar utvikleren benytte mange former for funksjonalitet som normalt sett er vanskelig å implementere selv. Selve 3D framvisningen er et HTML-lerret (*canvas*) der de to robotmodellene til en hver tid er representert med hver sin 3D-modell.

4 Diskusjon og konklusjon

En videodemonstrasjon av produktet kan ses på `http://youtu.be/avh6Fbj8tRQ`. Sluttproduktet var i stor grad i tråd med målsettingene for prosjektet. Grafikken ble svært detaljert og styringen er responsiv og gir god kontroll, hovedsaklig begrenset av den mekaniske sammenstillingen og servomotorene selv. Innkapslingen til servomotorene er i et plastmateriale, og er ikke veldig

stivt. Dette kombinert med at robotarmen ikke er festet til bordet/gulvet, gjør at armen har en tendens til å flekse og svaie i det man gjør krappe bevegelser med den.

Det var opprinnelig planlagt å implementere serverdelen av prosjektet på en Raspberry Pi, men dette ble ikke gjort. Detaljert diskusjon om dette, samt om andre utvidelser, finnes i avsnittet om videre arbeid.

Servomotorene som er utgangspunktet for prosjektet har en dødsone når den første vinkelen har en verdi rundt 180 grader. Her rapporterer den første sensoren en verdi på null grader. Dette vil forårsake en ukorrekt visualisering av roboten, men programmet forsetter med korrekt oppførsel når man kommer ut av dødsonen. Styringskommandoer i dette området fungerer som forventet. Dette er en uheldig bug, men feilen opptrer sjelden og den er lett å detektere og håndtere. En algoritme for feildeteksjon og beregning kunne blitt implementert for å løse dette problemet. Denne algoritmen ville forkastet alle målinger som innebærer et for stort hopp i målinger og bruke forrige verdi i stedet.

5 Videre arbeid

3D-grafikk er sentralt i produktet, og de fleste relevante utvidelser av prosjektet fordrer kunnskap på denne fronten. Kjennskap til robot-kinematikk vil også være sentralt for de fleste utvidelser.

Integrasjon av hjul

Robotarmen er festet på en bil med fire servomotorer som hjul, og en naturlig utvidelse vil være å inkludere denne i prosjektet. Hjulene på bilen har sensorer som gjør det mulig å inkludere bilen i 3D modellen. Dette åpner også for styring av bilen igjennom det eksisterende brukergrensesnittet.

Banefølging

Da forover- og inverskinematikk allerede er håndtert legger dette til rette for å generere en bane som en ønsker robotarmen skal følge. Man vil da kunne visualisere denne banen på forhånd med den transparente modellen før en gir kommandoen til roboten.

Dette kan kombineres med *Integrasjon av hjul* for å følge mer komplekse kurver i rommet.

Visualisering av omgivelser

For å bruke prosjektet i praksis vil en trenge en visuell referanse, og mulighet til å visualisere objektene som roboten skal arbeide på eller med. Visualisering av omgivelsene vil gi en solid

referanse for styring i 3D-modellen. For å oppnå dette kan en bruke kamera og sensorer, enten separat eller i kombinasjon. En Kinect eller en LIDAR sensor vil gi 3D-informasjon om omgivelsene som kan overføres til klienten og visualiseres sammen med roboten i 3D-visningen. Dersom man har en presis modell av fartøyets bevegelse vil et kamera kunne med god presisjon kartlegge omgivelsene. Et kamera vil også kunne bidra med farging av omgivelsene, og det vil være et godt supplement til 3D-visualiseringen. Dette vil trolig kreve kunnskap om bildebehandling og 3D-grafikk.

Implementasjon av serveren og driver til portabel datamaskin

Ved å overføre serveren og driveren til en mikrokontroller eller Raspberry Pi kan en oppnå trådløshet av robotarmen. Dette antas å være relativt lite tidkrevende, og det vil være praktisk å kombinere med en mobil robot. Det var et mål for prosjektet å implementere dette på en Raspberry Pi, men det ble forkastet da Qt 5.2 ikke var ferdig kompilert til ARM-plattformen til Raspberry Pi.

Implementere sikrere kommunikasjon

Sikkerhet har ikke vært en prioritet for oss i dette prosjektet, men kan være interessant i senere implementasjoner. I nåværende versjon av systemet er det ingen form for kryptering eller autentisering i kommunikasjonen mellom klient og server. Det vil si at det ville vært uproblematisk for utenforstående å benytte seg av systemet og potensielt misbruke det. For å motvirke dette bør det implementeres kryptering og også funksjonalitet som krever at operatører av systemet må logge seg på, for å sikre at kun de som faktisk skal ha tilgang til systemet får det.

Legge til flere ledd

For en mer sofistikert robot kan en legge til flere ledd. En kan utvide biblioteket for å støtte flere ledd uten å fysisk endre på roboten. Inverskinematikken for flere ledd er dog meget eksponentiell i kompleksitet, og i stedet for å utlede denne selv, bør man forsøke å finne et mer generelt bibliotek.

6 Referanser

[1] ROBOTIS. Dynamixel emanual. [Online]. Available: http://support.robotis.com/en/product/dxl_main.htm

[2] ——. Dynamixel eManual: AX-12/ AX-12+/ AX-12A. [Online]. Available: http://support.robotis.com/en/product/dxl_main.htm

[3] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. JOHN WILEY & SONS, INC., 2011. [Online]. Available: http://www.fit.hcmup.edu.vn/~hungnv/teaching/Robotics/0471649902_-_Robot_Modeling_and_Control.pdf

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (1999) RFC2616: Hypertext Transfer Protocol – HTTP/1.1. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html

[5] I. Fette and A. Melnikov. (2011) The Websocket Protocol. [Online]. Available: https://tools.ietf.org/html/rfc6455

[6] JSON-RPC Working Group. (2013) JSON-RPC 2.0 specification. [Online]. Available: http://www.jsonrpc.org/specification

[7] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. (2007) SOAP version 1.2 part 1: Messaging framework (second edition). [Online]. Available: http://www.w3.org/TR/soap12-part1/

[8] A. Lafarge. (2014) Qtwebsocket: A Qt websocket server / client implementation which supports SSL / TLS secured communications. [Online]. Available: https://github.com/antlafarge/QtWebsocket

[9] P. Thorson. (2013) Websocket++. [Online]. Available: http://www.zaphoyd.com/websocketpp

[10] A. Green. (2014) libwebsockets. [Online]. Available: http://libwebsockets.org/trac/libwebsockets

[11] S. Vincent. (2012) Jsonrpc-cpp. [Online]. Available: http://jsonrpc-cpp.sourceforge.net/doxygen/

[12] Devon IT. (2014) qjsonrpc. [Online]. Available: https://bitbucket.org/devonit/qjsonrpc

[13] (2014) Writing websocket client applications. [Online]. Available: https://developer.mozilla.org/en/docs/WebSockets/Writing_WebSocket_client_applications

[14] Awio Web Services LLC. (2014) February 2014 market share. [Online]. Available: http://www.w3counter.com/globalstats.php?year=2014&month=2

[15] StatCounter GlobalStats. (2014) Compatibility table for support of web sockets in desktop and mobile browsers, february 2014. [Online]. Available: http://caniuse.com/websockets

[16] Textalk AB. (2013) jQuery JsonRpcClient, jQuery JSON-RPC 2.0 client for HTTP and WebSocket backends. [Online]. Available: http://plugins.jquery.com/jsonrpcclient/

7 Vedlegg

A Hvordan sette opp serveren

A.1 Bygging av koden

For å bygge koden trengs byggeverktøyet CMake, samt en installasjon av Qt versjon 5.2 eller senere. Qt kan lastes ned fra `http://qt-project.org/downloads` CMake kan lastes ned fra `http://cmake.org/`.

Kildekoden er lagt ved innleveringen av rapporten som en fil (`server.zip`), men kan også lastes ned med Git fra `https://bitbucket.org/eit2014/server`.

A.1.1 Installasjon av Qt

Under installasjon av Qt, velg standard installasjonsvei. Som kompilator, velg MinGW som C++-kompilator. Pass på å krysse av for installere av QtCreator. Det er ikke nødvendig å kompilere med diverse MSVC kompilatorer, disse vil bare ta ekstra tid.

A.1.2 Installasjon av CMake

Programmet installeres på vanlig måte.

A.1.3 Kompilering

- Åpne QtCreator
- Velg **Open Project**
- Bla frem til mappen der kildekoden ligger og velg filen **CMakeLists.txt** og trykk **Next**
- På bygge-plassering velges ønsket lokalisering av de kompilerte filene. Trykk **Next**.
- Om det ikke er gjort tidligere, må lokalisering av *cmake.exe* finnes. Denne ligger der CMake er installert
- Trykk **Run CMake**. På Windows kommer det her noen advarsler, men så lenge byggefilene ble skrevet, er alt OK.
- Prosjektet kan nå kompileres ved å trykke hammeren nederst til venstre.

A.2 Test kommunikasjon med roboten

For å gjøre det enkelt å teste roboten, ble det laget flere små programmer som kjøres uavhengig av konfigurasjon. Det enkleste programmet å teste er **wriggle**.

Roboten er koplet til datamaskinen gjennom en serieport. På Linux vil den få et device-name som `/dev/ttyUSB#` der # er et tall fra og med `0`. Om ingen andre serieporter er tilkoplet, vil dette tallet være 0.

På Windows vil den dukke opp i *Device Manager* som en `COM#`-input der # er et tall fra og med `3`. Om ingen andre serieporter er opptatt, er dette tallet `3`.

- Kople til roboten.
- I QtCreator trykk på **Projects** i venstremenyen.
- Øverst i programmet kan man velge mellom **Build** og **Run**. Trykk på **Run**.
- Under *Run configuration*, velg **wriggle**.
- I tekstboksen **Arguments**, skriv inn **0** for Linux eller **3** for Windows. Dette er det samme nummeret som beskrevet i paragrafen over.
- For linux kan det hende det må endres rettigheter på robot-input-filen. Gå inn i en terminal og skriv inn *"`sudo chmod 777 /dev/ttyUSB0`"*. For å kjøre denne kommandoen trengs passord.
- Trykk på den grønne **Run**-knappen i nedre venstre hjørne for å starte.
- Følg instruksjonene til programmet. Roboten vil begynne å bevege seg, så pass på at den står slik at ingen blir skadet.

A.3 Modellfilen

Modellfilen er en JSON-fil som beskriver hvordan roboten er bygget opp. Denne modellfilen viser hvor de forskjellige servoene er koplet sammen. Hvis det brukes en annen robot enn den som ble brukt i dette prosjektet, må modellfilen konfigureres for den nye roboten.

Hvis modellfilen er korrekt, er neste punkt å teste serverprogrammet.

A.4 Starte serveren

- Kople til roboten og test kommunikasjonen som beskrevet i kap. A.2.
- Bytt program fra **wriggle** til **server**.
- I tekstboksen **Arguments**, skriv inn samme tall som fungerte under **wriggle**testen.

- Trykk på den grønne **Run**-knappen i nedre venstre hjørne for å starte.

Et terminalvindu dukker opp når serveren er oppe å gå. Når klienter kobler seg til og fra serveren vil informasjon om dette dukke opp i terminalvinduet.

A.5 Finn IP

For å gjøre det mulig for andre å kople seg til serveren, må de vite hvilken IP serveren ligger på. IPen kan blant annet finnes ved å gå inn på `http://www.whatsmyip.org/`. Hvis datamaskinen serveren ligger på er bak en ruter, har man to valg. For at serveren skal komme på internett må man bruke *port forwarding* på de to portene `1991` og 5555 på ruteren. Det er disse portene som styringsforbindelsen og tilstandsforbindelsen opprettes mot. Hvis både server og klient kjører på samme lokale nettverk, kan den lokale IP-adressen brukes i stedet.

B Hvordan sette opp web-siden (klienten)

Websiden bør aksesseres gjennom en webserver. For test er det tilstrekkelig å bruke en liten webserver som Mongoose (`https://code.google.com/p/mongoose/`). Det kan være mulig å vise nettsiden i nettleseren rett fra filsystemet, men dette kan kreve en spesiell innstilling i nettleseren for at JavaScript fungerer.

- Pakk ut filen `client.zip` som er vedlagt rapporten, til en mappe som webserveren er konfigurert for å levere.
- Finn filen `rpcserver.js` i mappen *src*.
- Åpne denne filen med et hvilket som helst tekstredigeringsverktøy (feks. notepad/notisblokk).
- På fjerde linje finner du variabelen `ip`. Endre verdien på denne variabelen til å samsvare med IP-adressen som tilhører robot-serveren. Lagre endringene og lukk programmet.
- Brukeren kan nå styre roboten over internett gjennom nettleseren sin ved å aksessere `index.html` som du deler på din web-server.