Høgskulen på Vestlandet

BACHELOROPPGAVE:
BO19E-16
SELVBYGGENDE KONSTRUKSJONER

Henrik Lunde Lyssand
Olav Henrikson Steine

31. mai. 2019

Dokumentkontroll

Rapportens tittel: BO19E-16 Selvbyggende konstruksjoner	*Dato/Versjon* 31. mai. 2019/1.0
	Rapportnummer: B019E-16
Forfatter(e): Henrik Lunde Lyssand Olav Henrikson Steine	*Studieretning:* HEAU16
	Antall sider m/vedlegg 60
Høgskolens veileder: Ingvar Henne	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: Daniel Patel	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): E-post: dapa@norceresearch.no, dpa@hvl.no Telefon: +47 482 15 561	

Revisjon	**Dato**	**Status**	**Utført av**
0.11	29.01.19	Forstudie	Olav og Henrik
0.12	02.05.19	Første utkast – halvferdig rapport	Olav og Henrik
0.13	24.05.19	Revidert utkast – ferdig rapport	Olav og Henrik
1.0	30.05.19	Ferdig rapport	Olav og Henrik

Forord

Denne teksten er skrevet som bacheloroppgave i automatisering ved institutt for elektrofag på Høgskulen på Vestlandet i Bergen våren 2019. Bacheloroppgaven er avsluttende oppgave etter tre år med studie.

Vi har jobbet med programmering av mikrokontrollere og trådløs kommunikasjon, i tillegg til å ha brynt oss på design og konstruksjon av fysisk modell. Arbeidet har vært variert og utfordrende og vi har lært mye av å jobbe med et prosjekt over en lang periode.

Vi vil gjerne takke medstudent Benjamin Skare og professor ved HVL Mathias Christian Mathiesen som har vært behjelpelig med å låne oss diverse utstyr i løpet av oppgaven. De har i flere tilfeller hjulpet oss videre når vi har stått stille. Vi vil også takke Stina Marie Grytten for hjelp med gjennomlesing av teksten.

Til slutt må vi takke veileder Ingvar Henne og ekstern veileder Daniel Patel som har fulgt prosjektet underveis og har gitt oss nyttige tilbakemeldinger.

Bergen, 30.05.2019

Henrik Lunde Lyssand

Olav Henrikson Steine

Sammendrag

Arbeidet i dette prosjektet går ut på å lage et system for selvbyggede konstruksjoner hvor en robot skal kunne endre utformingen til et rom ved hjelp av ulike byggeklosser. Målet er at systemet skal endre rom etter behov, for eksempel ved å sette opp skillevegger, sofaer eller andre typer konstruksjoner. Systemet består av søyler som bærer konstruksjonen og en robot som kjører mellom dem. Roboten frakter med seg klosser til gitte destinasjoner og heiser de opp til konstruksjonen over. For å unngå gliper i konstruksjonen over må søylene kunne flyttes ut av veien slik at store nok klosser kan komme forbi.

Løsningen krever kompetanse fra flere fagfelt, men vi har satt fokus på det som har med kontroll og styring av systemet. Vi laget en løsning hvor roboten navigeres trådløst fra en ekstern enhet over et nettverk. Den eksterne enheten har kontroll på hele systemet og sender kommandoer til både søyler og robot om hvor og når de skal bevege seg. Slik kommer roboten seg til og fra en liste med destinasjoner slik at en valgt konstruksjon til slutt er ferdig. Vi valgte å lage en liten modell av systemet selv om dette ikke var et formelt krav. Dette var for å kunne både teste at logikken vår stemte, men også å se hvordan den fungerte når den ble oversatt til et fysisk system.

1 Innhold

1 Innledning

1.1 Organisering av rapporten

Rapporten starter med et visualiserende eksempel på hva oppgaven går ut på, dette eksempelet kan være nyttig å se tilbake på hvis man ikke forstår noe senere i rapporten. Videre forklares problemstillingen og hvilke krav vi stiller til løsningen av oppgaven. Etter at kravene er satt kommer et metodekapittel som forklarer litt hvordan vi gjorde litteratursøk og samlet inn informasjon. Deretter analyserer vi hvordan vi skal løse utfordringer i prosjektet og diskuterer løsningsalternativer. Vi viser deretter hvordan vi realiserte oppgaven, testet de viktigste elementene og presenterer til slutt en diskusjonsdel som tar for seg de videre utfordringene.

1.2 Oppdragsgiver

Oppgavens oppdragsgivere er HVL og NORCE Research.

Høgskulen på Vestlandet (HVL), Inndalsveien 28, 5063 Bergen, 55 58 58 00, post@hvl.no

NORCE Norwegian Research Centre AS, Nygårdsgaten 112, 5008 Bergen, 56 10 70 00, post@norceresearch.no

Oppgavens veiledere er:

Ingvar Henne:
E-post: ingvar.henne@hvl.no
Telefon: +47 55 58 75 87

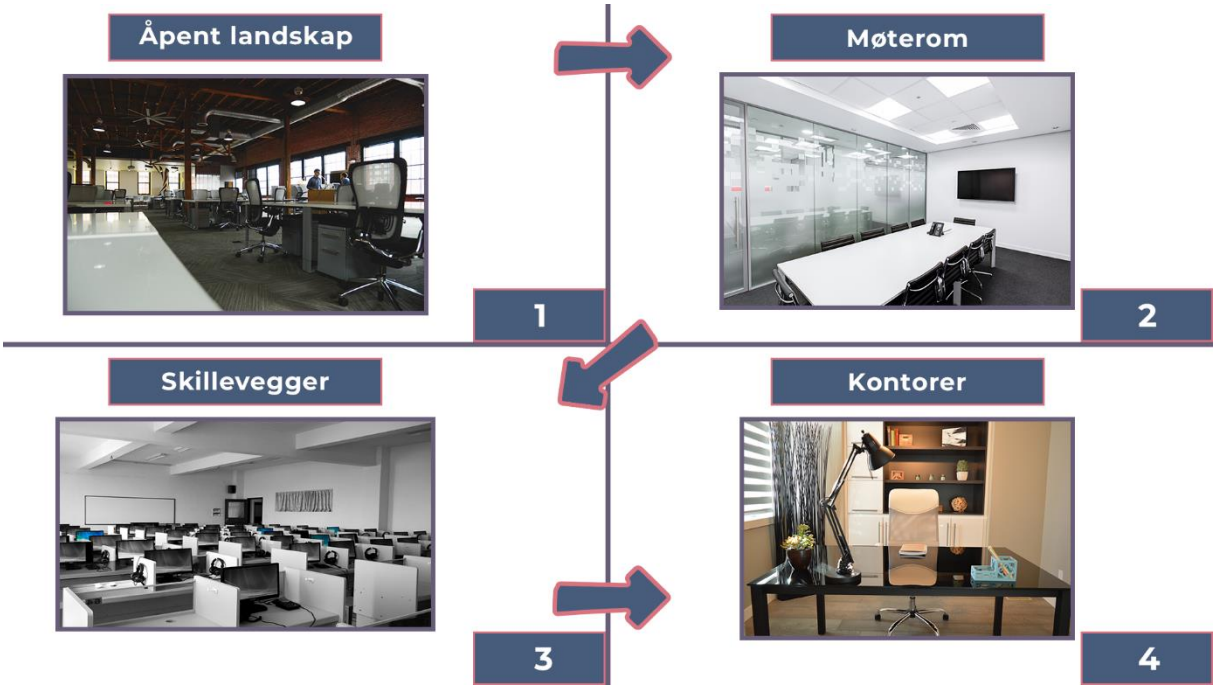
Daniel Patel:
E-post: dapa@norceresearch.no , dpa@hvl.no
Telefon: +47 482 15 561

Gruppen som har løst oppgaven har hatt arbeidssted på HVL, men har også hatt mulighet til å være på NORCE.

1.3 Oppgavens hensikt

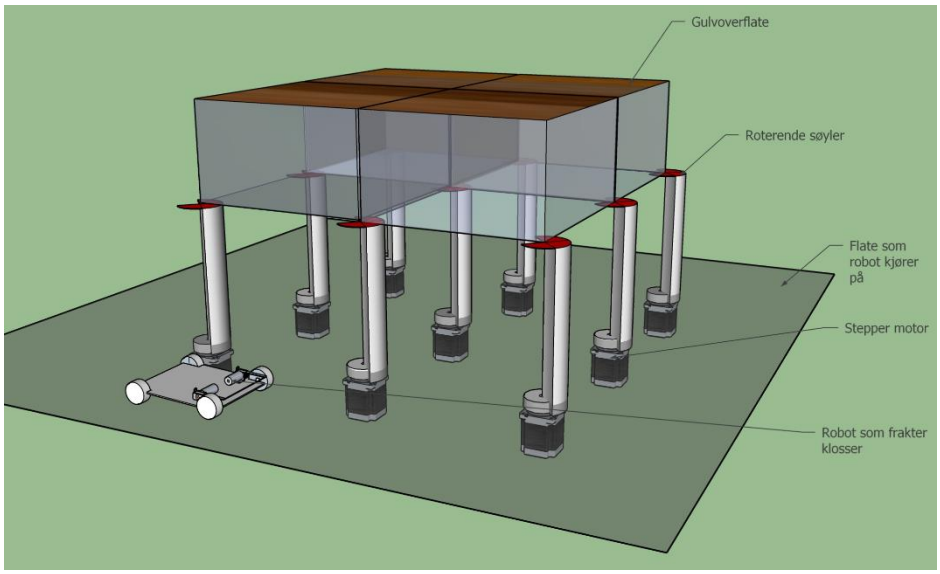
Tradisjonelle byggemetoder gir statiske rominndelinger, og endringer i bygget krever kostbart og tidkrevende arbeid av håndverkere. Byggene blir derfor ikke optimale i forhold til øyeblikksbehovene som kan endre seg ganske raskt og ofte. Eksempler kan være åpne kontorlandskaper, hoteller, konferansesentre og skoler der det ofte er varierende behov ut fra bestillinger. Ser man på skolen vi studerer på er plassmangel et problem. En løsning kunne vært at skolen automatisk kunne endret undervisning og grupperom over natten basert på morgendagens timeplan og de ulike bestillingenes plassbehov. Konferansesentre kunne også brukt en tilsvarende løsning til å dele opp konferansesalene sine til antallet mennesker som er forventet noe som kunne ført til flere rom og høyere inntekter.

På figur 1 har vi et eksempel på en bedrift med varierende behov for lokalet sitt. Bedriften vet ikke sikkert hvilken planløsning de ønsker. De ser for seg at bedriften vil trenge møterom, noen skillevegger og kontorer. En løsning der det settes opp permanente vegger til å skape de forskjellige planløsningene er ikke ønskelig. Målet med prosjektet er å lage en løsning som automatisk skal kunne omorganisere rom slik at et stort åpent område kan rekonfigureres automatisk til for eksempel mange små kontorer, mellomstore møterom eller en blanding.



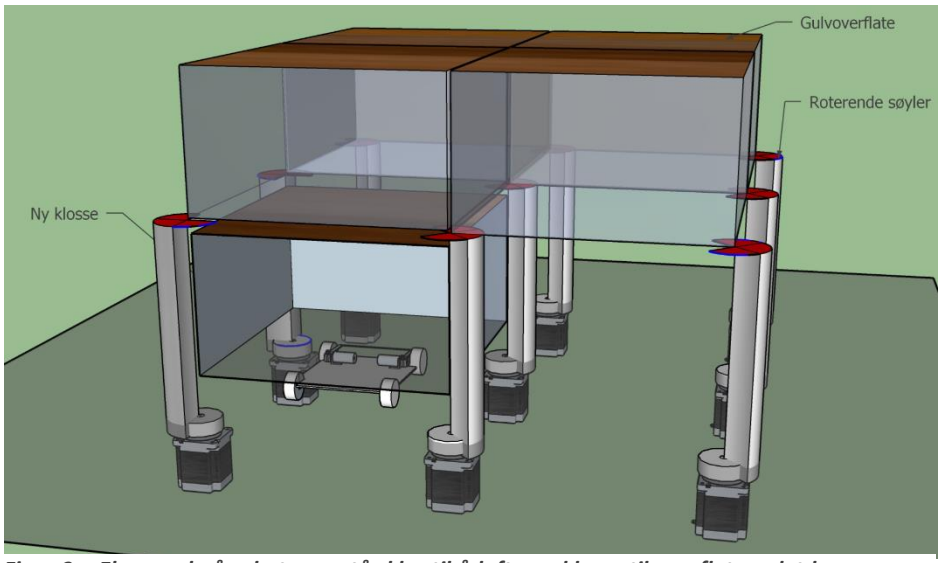
Figur 1 – Å automatisk kunne endre bruken av et rom (1,2,3,4) ved blant annet å sette opp og ned skillevegger er visjonen bak oppgaven. (Selvlaget illustrasjon fra [22] og [32])

Når vi nå skal forklare hvordan prosjektet virker er det lurt å ha eksempelet over i bakhodet. For å løse utfordringen er det meningen at vegger skal kunne bygge seg opp av seg selv og forsvinne uten at noen må gjøre et fysisk arbeid. Idéen er at man har ulike klosser som automatisk skal kunne settes sammen, dette gir mange muligheter som vegger og grove møbler. Distribueringen av klossene skjer under hele konstruksjonen og klossene skubbes opp og fjernes fra undersiden. Da slipper man en robot som beveger seg på det potensielt ujevne terrenget på gulvoverflaten, som må stable kubbene i ulike høyder. Måten dette skal utføres på er ved hjelp av at gulvet består av klosser som hviler på søyler som står under gulvoverflaten. Se figur 2 for å visualisere.



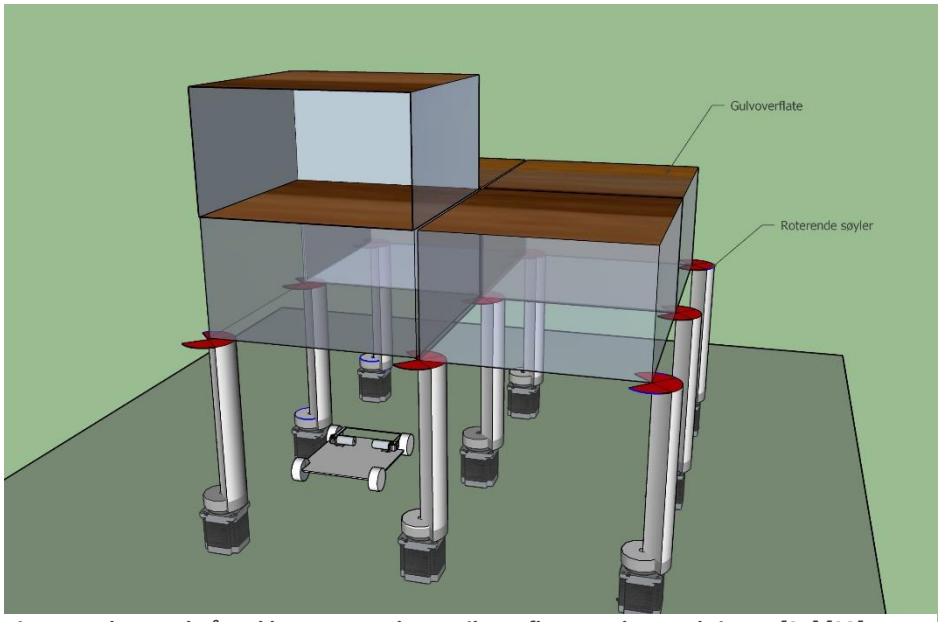
Figur 2 – Eksempel på gulvoverflate, søyler med tilkoblet trinnmotor og selvkjørende robot [25] [32]

På toppen av figuren ser man gulvoverflaten, det er den overflaten man vil kunne se når man står i rommet. Under denne gulvoverflaten er det søyler som har muligheten til å rotere seg for å få tilgang til klossene. Her vil det være en eller flere roboter som kan kjøre mellom søylene. Jobben til roboten vil være å hente og levere klossene som hviler på søylene. Når roboten stiller seg under en av klossene som er der fra før, vil den kunne skyve opp nye klosser eller hente ned klosser og frakte dem bort, se figur 3.



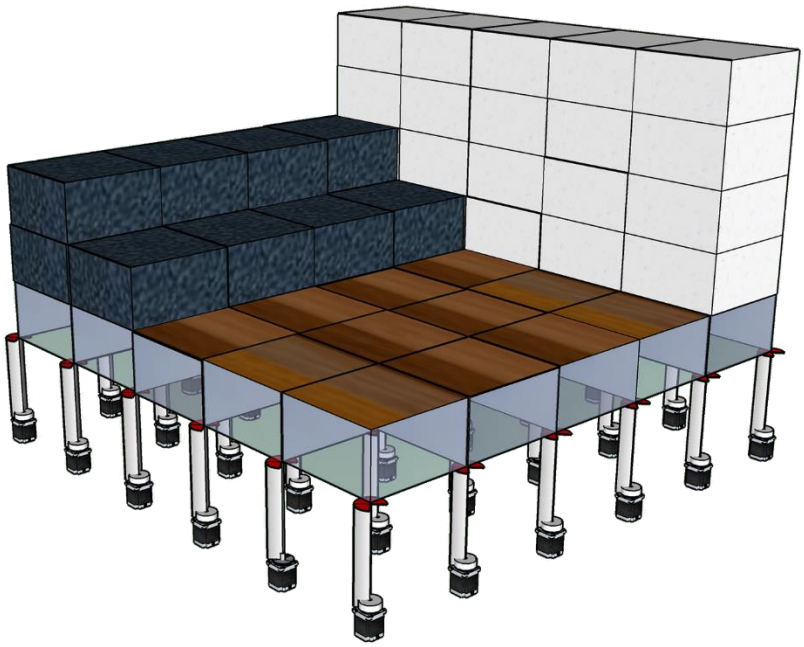
Figur 3 – Eksempel på robot som står klar til å løfte en klosse til overflaten, det kan og være eksempel på en robot som har hentet ned en klosse fra overflaten [25] [32]

Når klossen blir hevet opp vil gulvoverflaten stige og det vil se ut som i figur 4. Ved å gjenta prosessen vil man kunne lage stabler med klosser av valgfri høyde. Flere stabler ved siden av hverandre vil danne en vegg og løse det som er utfordringen i dette eksempelet. For å fjerne veggen igjen gjøres operasjonen i motsatt rekkefølge. Det er utviklet en plan for hvordan klossene vil feste seg i naboklosser, men det er tenkt som et senere prosjekt.



Figur 4 – Eksempel på en klosse som er hevet til overflaten av konstruksjonen [25] [32]

Dette er et eksempel for å forstå hva prosjektet burde kunne løse, men det er ikke satt noen grenser fra oppdragsgiver angående hvilke utfordringer prosjektet skal ha. Derfor er det kun kreativiteten som setter en stopper på hva som kan bygges og hvordan det skal gjøres. Et eksempel på en sofa og en skillevegg er vist i figur 5.



Figur 5 – Eksempel på en sofa og en skillevegg som bygges av klosser som heves opp fra underflaten [25] [32]

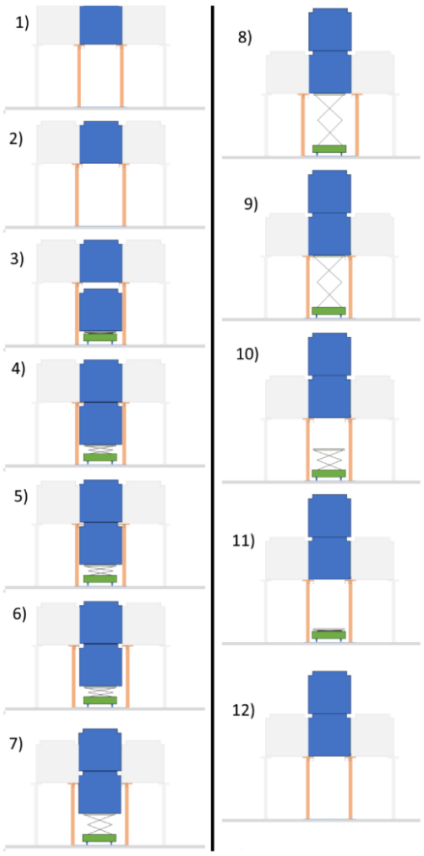
1.4 Problemstilling

I dette underkapittelet forklares prosjektets opprinnelige problemstilling. Prosjektet nedskaleres og dette vil bli forklart i kapittel 1.5.

1.4.1 Beskrivelse av problemstilling

Oppdragsgiver ønsker at vi skal utvikle et «proof of concept» til en selvbyggende konstruksjon i tillegg til en 3-dimensjonal-grafikk av systemet. «Proof of concept» vil si en modell som kan vise en fungerende løsning, men ikke i samme skala som et ferdig produkt. Figur 6 og 7 viser hvordan problemstillingen først ble vist frem. Figur 6 viser hvordan løfting av klosser skal fungere og hvordan søylene flytter seg for at klossene skal komme seg opp på overflaten. Figur 7 er en 3-dimensjonal-visualisering av hvordan et søylesystem ser ut og hvordan klosser er stablet på oppsiden.

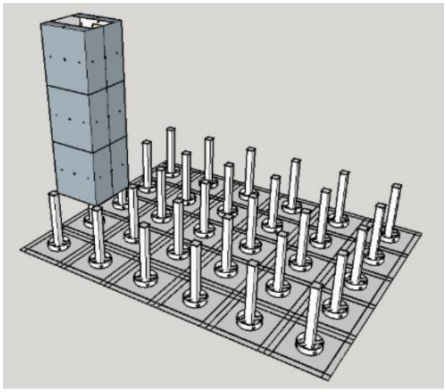
Systemets problemstillinger blir delt inn i tre fagfelt, maskin, elektro og data. Maskin skal løse problemer med tanke på konstruksjon, og skal designe og lage robot og søyler. Deres løsning må ta høyde for utforming i tillegg til krav med tanke på vekt og stabilitet. Elektro skal løse problemer med tanke på styring, og skal designe og lage elektro og styringssystemer. Løsningen deres må være robust og presis slik at roboten kjører dit den skal og søylene gjør de bevegelsene som er nødvendig underveis. Samtidig skal data lage en digital tvilling som skal være en 3-dimensjonal-framstilling av det samme systemet.



Figur 6 – Steg for steg illustrasjon av robot som levere klosse (fra oppgavetekst [15])

1.4.2 Utfordringer

Basert på illustrasjonene som ble vist fra starten kom det tidlig frem utfordringer som måtte tas høyde for når prosjektet begynte. For at klossene skal komme frem i systemet er man avhengig av at søylene flytter seg ut av veien. Både ved transport under overflaten og ved løfting/senkning av klosser. Dette kan føre til tilfeller hvor konstruksjonen ikke har den støtten som den krever. I en endelig modell må det finnes en løsning for å lage åpninger uten at klosser som ikke skal flyttes mister støtten fra søylene.



Figur 7 – Tre klosser som er plassert på søylekonstruksjonen (fra oppgavetekst [15])

1.5 Hovedidé for løsningsforslag

Avgrensningene til oppgaven kommer av at det ikke vil være en maskin eller datadel i oppgaven. Det betyr at design av søyler og robot blir nedprioritert, samt at det ikke lages en grafisk simulasjon av systemet. Hovedidéen fra oppdragsgiver er at vi gjør elektro og styresystems-delen til oppgaven og beviser at styresystemet til prosjektet kan utvikles. Styresystemet vil bestå av en server som sender signaler til resten av systemet om hvordan det skal oppføre seg. Oppdragsgiver har ikke satt som krav at vi utvikler en fysisk prototype eller tar hensyn til løftefunksjonen til roboten, men at det utvikles fjernstyrt navigering av robotens posisjon og en styring av søylene.

2 Kravspesifikasjon

Prosjektet vil være et «proof of concept» av styresystemet. Prosjektet vil fokusere på disse tre punktene:

- Lage (design og) elektronikk til en robot slik at den skal kunne styres trådløst fra en server.
- Lage (design og) elektronikk til en robot slik at den skal kunne navigere mellom alle søylene.
- Lage (design og) elektronikk til søylene slik at de skal kunne flyttes ut av veien når roboten krever det.

Det er ikke et krav for prosjektet å bygge en fysisk modell, men etter hvert har vi valgt vi å bygge en for å visualisere utfordringene til prosjektet. Vi vil gå nærmere inn på hvorfor i kapittel 3. I tillegg skal det ikke designes en løftefunksjon slik at roboten kan løfte byggeklossene.

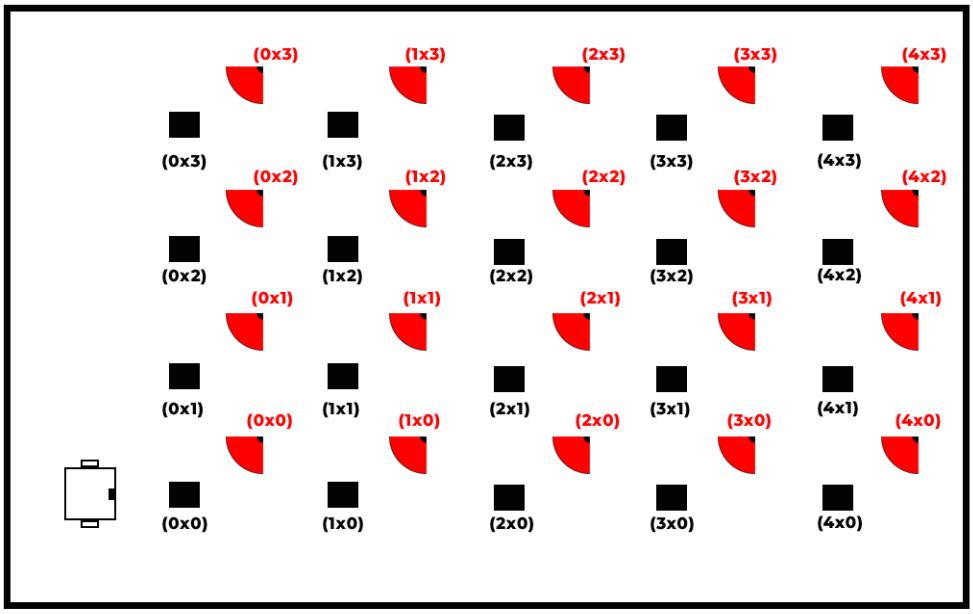
Styringen vil bestå av en robot som skal komme fra en lagringsplass for byggeklosser og derfra kjører den inn i søylekonstruksjonen som forskyver seg unna slik at roboten kan passere. Roboten skal nå en gitt posisjon i systemet og deretter returnere samme vei.

2.1 Trådløs styring

Et krav for prosjektet er at kommunikasjonen med roboten skal være trådløs. Systemet skal kunne styres og overvåkes fra en server som ikke er fysisk tilkoblet roboten. Fra serveren skal en bruker kunne bestemme utformingen til rommet.

2.2 Navigasjon blant søyler

Roboten skal kunne klare å finne frem til riktig posisjon blant søylene. Ved å nummerere søylene i forhold til rekke og kolonne får vi et 2-dimensjonelt koordinatsystem/rutenett. Rutenettet vil være utformet av søylene. Søylene bærer klossene og er plassert i hjørnet av hver kloss. Det vil være avgjørende at roboten klarer å bevege seg presist siden det ikke vil være mye plass mellom søylene og klossene som skal flyttes. Dersom det er upresist vil roboten fort sette seg fast. Roboten skal flytte seg fra en startposisjon utenfor rutenettet, videre til en valgt posisjon og så tilbake til startposisjonen. Rutenettet vises i figur 8.



Figur 8 – Oversikt over rutenett som brukes i logikken til robot og søylene [32]. De sorte merkene representerer stoppunktene til roboten og de røde merkene er søylene.

2.3 Forskyvning av søyler

Søylene som bærer klossene skal kunne forskyves eller flyttes slik at roboten kan frakte en klosse mellom dem. I tillegg skal det være mulig å endre posisjonen til bæreflaten på søylene slik at det blir plass til å løfte en klosse opp gjennom en åpning. Dette må skje på en slik måte at det ikke får konsekvenser for konstruksjonen som ligger på oversiden. Bevegelsene må ikke føre til at noen av klossene sklir til siden eller faller ned.

2.4 Integrering i fult prosjekt

Løsningen som utformes i løpet av prosjektet har som formål å fungere i en fullstendig modell. Selv om ikke alle delene er på plass, skal det tas hensyn til alle komponenter som ikke er til stede. Den trådløse tilkoblingen må for eksempel fungere i et miljø som det ferdige produktet skal være i. Selv om konstruksjonen av søylen ikke er fokus, er det viktig at løsningen fungerer selv med belasting på søylen.

3 Metode

I dette kapittelet forteller vi om hvordan vi har jobbet med problemstillingen og prosessen vi har gått gjennom ved innsamling av informasjon. Elementene som blir nevnt i dette kapittelet vil bli nærmere gått gjennom i kapittel 4, analyse av problemet.

Konklusjonen fra oppgaveteksten og dialogen med veilederne er at en robot skal kunne styres trådløst fra en server. En server vil sende signaler til roboten når den vet at søylene er i riktig posisjon. Roboten tolker signalene og kjører da videre gjennom konstruksjonen. Søylene trenger ikke være trådløse og kan tilkobles serverens porter med fysiske kabler.

For å finne den beste løsningen til trådløs kommunikasjon mellom server og roboten ble det utført datainnsamling fra flere kilder. Gjennom HVL-biblioteket sine sider var det mulig å lese seg opp på artikler med lignende problemstillinger. Artikkelsøk på «self driving robot» førte oss til en artikkel [1]. Metoden brukt i artikkelen gikk ut på at roboten strømmet video til en laptop, som da tok beslutninger ut fra sensorer som er montert på roboten og sender dataene tilbake til en mikrokontroller [33] på roboten, som deretter kontrollerer roboten ved hjelp av en motordriver. Selv om denne metoden var interessant valgte vi å søke videre ettersom denne metoden brukte kamerastrømming som ikke var relevant for prosjektet. Denne metoden var likevel inne på noe av de samme grunnprinsippene vi ønsket. Et annet søk på «Arduino robot driving» (Arduino [19] er en type mikrokontroller) førte til en artikkel [2]. Artikkelen hadde en generell forklaring på en robot som fulgte en linje ved hjelp av et system som består av Arduino med trådløs tilkobling, en infrarød sensor [14] og et separat styresystem (dataprogram som styrer enhetene i systemet). Artikkelen gikk ikke veldig i dybden på hvordan systemet var laget, men hadde mer teknisk data på de forskjellige komponentene. Artikkelen viste oss at et system styrt fra en separat kilde ved hjelp av WiFi og med respons fra en infrarød sensor var mulig å bruke. Vi valgte å utforske videre på hvordan en mikrokontroller kommuniserer trådløst med andre kilder.

Etter forskjellige søk på Arduino mikrokontrollere med WiFi ble vi oppmerksom på Arduino Uno WiFi rev2 [3] som var en ny modell. Arduino viste til en side som het «Getting started with Arduino WiFi» [4]. Der var det vist hvordan man kunne koble seg direkte til en Arduino Uno WiFi og deretter velge å laste opp kode til mikrokontrolleren ved å bruke nettverkstilkoblingen som port istedenfor å bruke direkte kabel. Arduino sine sider viste også til biblioteket WiFiNINA [5] som var laget for at mikrokontrolleren kunne sende og motta data fra en server. Med denne informasjonen konkluderte vi med at en Arduino Uno WiFi rev2 ville kunne bli programmert eksternt via trådløs tilkobling, kunne sende data til en server og kunne motta data fra en server. Forventningene til Arduino Uno WiFi rev2 viste seg i ettertid å være for høye. Da vi mottok mikrokontrolleren var det ikke mulig å koble til den trådløst, ved undersøkelse av problemet viste det seg at denne funksjonen var borte på den nyeste modellen. Videre ble det også oppdaget at hvis man ville både sende og motta data fra mikrokontrolleren, så hengte den seg opp i flere sekunder før den startet igjen, noe som førte til store problemer med presisjonen på roboten. Vi tok en beslutning om at Arduino WiFi rev2 ikke kunne sende og motta trådløse signaler samtidig, og at den ikke kunne programmeres uten kablet tilkobling. Påliteligheten i datainnsamlingen vår var ikke så høy som vi hadde forventet, selv om den var direkte fra produsenten sine sider. Lærdommen vi fikk var at en veldig ny modell (høst 2018) som ikke hadde mye informasjon fra andre forbrukere kunne være veldig annerledes i praksis enn på papiret.

Ved et vanlig google søk på «Arduino robot car» finner vi at det er svært mange som velger å bruke en mikrokontroller som kalles ESP8266 [6] på roboter som styres fra en server. Eksemplene som vises [7] [8] er enkle og viser til kode på hvordan det virker. Påliteligheten til disse eksemplene er stor med tanke på hvor mye detalj som vises av prosessen, samt kode og dokumentasjon som ligger ved. Selv om ingen av eksemplene har like utfordringer som vårt prosjekt, viser undersøkelsen gode eksempler som kan brukes for å løse utfordringer i prosjektet. Etter problemene som oppstod med Arduino Uno WiFi rev2 skaffet vi oss fire stykker ESP8266 D1-mini [9], som var svært gunstig i pris. Konklusjonen var at det kunne hjelpe oppgaven å komme seg videre med noe som virket, istedenfor å bruke mye tid på noe som ikke virket som planlagt. Da vi satte inn ESP-er som mottakere og sendere hver for seg, virket system slik som intensjonen var fra starten av. Det gjorde også at Arduino Uno WiFi kun trengte å sende signal uten å motta, som gjorde at den virket uten at den hengte seg opp. Metoden vi brukte ved å se eksempler av lignende utfordringer gjorde at vi kom oss videre med oppgaven mye raskere, istedenfor å bruke mye tid på å løse hver utfordring på egenhånd.

Prosjektets andre store utfordring var søylekonstruksjonene og hvordan man kunne kontrollere bevegelse av dem på en kontrollert måte. Vi hadde tidligere i studiet brukt servomotorer [11] med eget bibliotek i programmeringsprogrammet Arduino integrated development environment (IDE) [20] til å rotere med høy presisjon. Vi trengte nå en servomotor som roterte 360-grader for å løse rotasjonen av søylene begge veier uten at konstruksjonen faller sammen. Litteratursøk vi gjorde var for å finne en servomotor som kunne rotere 360-grader istedenfor de 180-gradene vi hadde kjennskap til med samme presisjon. Google søk på servoer med 360-rotasjon viste raskt billige servoer [10] [11] som virket å ha de samme egenskapene med mulighet for full rotasjon. Ved mottagelse av servoene oppdaget vi raskt at kode-biblioteket som tidligere var brukt ikke virket likt på denne typen servomotor. Det viste seg at servoer med 360-graders rotasjon virket mer som en vanlig motor, bare at du kunne sette farten og hvilken retning den skulle kjøre. Vi prøvde å bruke tidsinnstillinger på servoene for å rotere dem, men etter mange forsøk ble det konkludert til å være for upresist. Ved ny datainnsamling på internettforum [12] ble det gjort en beslutning mot det vi tidligere hadde sett for oss angående rotasjon av 360-graders servo. Vi måtte konkludere med at datainnsamlingen vår hadde vært for svak og at 360-graders servomotoren ikke virket på samme måte som 180-graders servomotoren. Det var også undersøkt muligheter ved bruk av trinnmotor [13] under datainnsamlingen, men denne ble avskrevet siden vi var mer kjent med servomotorer fra studieløpet, og trodde denne ville fungere godt nok til ønsket løsning. Vi trodde også at trinnmotoren var dyrere ved første datainnsamling, men med nærmere undersøkelser viser det seg at begge motorene har forskjellige versjoner som varierer mye i pris, de motorene som vi trengte var i noenlunde samme prisklasse. Ved ny og grundigere datainnsamling ble det funnet trinnmotorer som passet godt til prosjektet til nær samme pris [13].

Vår metode for datainnsamling og undersøkelse har bestått av artikler, websaker, forum, opplæringsvideoer på internett, datablad, arbeidserfaring, erfaring fra studiet og kontakt med rådgivere og medstudenter.

4 Analyse av problemet

I dette kapittelet går vi gjennom hvilke løsninger som er mulig å bruke, hvilke alternativer vi har og til slutt valget vi tar. Metodedelen har vist hvordan prosessen har vært og i dette kapittelet vil vi ta en mer systematisk gjennomgang av problemstillingen.

4.1 Utforming av mulige løsninger

I dette underkapittelet tar vi for oss hvilke mulige løsninger vi har og hva vi ser på som positivt og negativt med de forskjellige løsningene.

4.1.1 Mikrokontrollere

Til styringen av robot og søyler ønsker vi og oppdragsgiver at det er kommunikasjon med en server som sørger for samhandling ved levering og henting av byggeklosser. Det er fordi det ikke er ønskelig med en fysisk kabel koblet til en kjørende robot. Til denne styringen av systemet er det mulig å bruke Arduino [3] med trådløs tilkobling. Det vil da være mulig å kommunisere med en felles server som styrer samhandlingen av motorer og sensorer. Det er også en mulighet å bruke Raspberry Pi [28] til denne problemstillingen. Raspberry Pi er en mikrokontroller som virker mer som en datamaskin og kan brukes med tastatur, mus og skjerm, mens Arduino er en mikrokontroller som bare kan kjøre et program om gangen. Arduino er også kjent som enklere å bruke [16] enn Raspberry Pi ettersom Arduino bygger på en tidlig versjon av C++ [17] kode, som vi har kjennskap til fra studiet. Bruksområde til de to forskjellige mikrokontrollerne kan beskrives som at Raspberry Pi er best der man trenger å kontrollere svært kompliserte roboter med mye beregninger som gjør flere forskjellige oppgaver. Arduino er best til enklere oppgaver som gjerne gjentas flere ganger og bruker sensordata til rapportering. Under i figur 9 og 10 er eksempler på to mikrokontrollere som bruker Arduino sitt programmeringsspråk og virkemåte, og i figur 11 er Raspberry Pi. Figur 10 viser en ESP8266 D1-mini som er en del mindre enn en Arduino med færre porter, men har ellers like funksjoner. Valget av mikrokontroller blir skrevet om i kapittel 4.2.6.

Figur 9 – Arduino Uno WiFi [3]

Figur 10 – ESP8266 D1-mini [9]

Figur 11 – Raspberry Pi [28]

4.1.2 Motorer

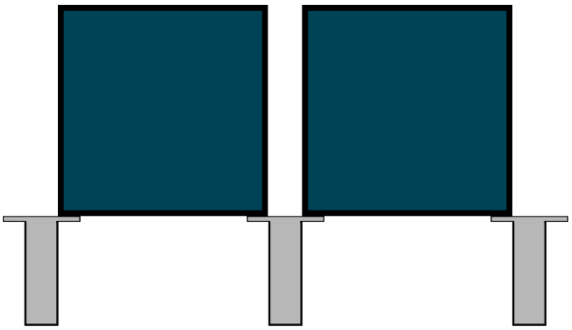
Fremdriften til roboten vil trenge en eller flere elektriske motorer, det er mange forskjellige motorer på markedet og det er stor variasjon i hvilke motorer som er aktuelle. Det viktigste ved valget av motor til roboten er nøyaktighet/slark og dreiemoment. Motor til søylene må som nevnt tidligere kunne roteres eller forskyves. I starten gikk vi ut ifra at en servomotor [11] ville kunne utføre denne rotasjonen presist, men etter testing viste det seg at den ikke hadde den presisjonen vi ønsket. For å oppnå presis rotasjon bruker vi trinnmotor [29] som tillater presis kontroll av posisjon. Trinnmotoren teller antall steg motoren roterer og er derfor mulig å brukes til presis rotasjon, vi går nærmere inn på virkemåten til trinnmotoren i kapittel 5.1.4.

4.1.3 Sensorer

For at systemet skal kunne kjøre feilfritt må det være mulig å få noen tilbakemeldinger ved hjelp av sensorer. Det vil alltid være nyttig å vite at roboten treffer de målene den skal og vite posisjonen i systemet underveis. Planen for å løse dette vil være å bruke en infrarød sensor [1] som skal registrere merker i gulvet under roboten. En robot som skal hente eller plassere en klosse må stoppe når den har nådd målet. Dette oppnår vi ved å ha markører i bakken som blir registrert av sensoren når roboten passerer punktene. Denne responsen vil kunne brukes videre i et program som har kontroll på hendelsesforløpet i systemet. Et annet alternativ vil være å sette antall rotasjoner roboten skal kjøre og velge fastsatte rotasjoner for hvor den beveger seg. Men dette vil bli unøyaktig og svært vanskelig å overvåke ettersom man ikke får noen input. GPS er også et alternativ, men vil også være unøyaktig i forhold til hvor presist dette prosjektet må være.

4.1.4 Utforming av søyler

Utformingen av søylene er avgjørende siden det også avgjør hvordan styringen blir. Det er ikke ønskelig at tykkelsen på søylene som bærer konstruksjonen viser igjen som gliper i konstruksjonen på gulvoverflaten, se figur 12. Dersom konstruksjonen ikke skal ha slike gliper når den er ferdig er det viktig at det blir nok plass til klossen på undersiden. Derfor må søylene kunne forskyves eller roteres bort fra banen til roboten og klossen. Denne bevegelsen må samtidig ikke gjøre at bokser som hviler på konstruksjonen faller ned. Når klossen har kommet frem til riktig posisjon, skal klossen kunne løftes opp. Den skal først løftes opp slik at den fungerer som støtte for en eventuell klosse som ligger over. Deretter må søylene kunne flyttes bort slik at det dannes en åpning som er stor nok slik at klossen kommer gjennom. Når den er kommet gjennom skal søylene flyttes tilbake slik at de blir støtter igjen.



Figur 12 – Uønsket glipe mellom klosser [32]

4.1.5 Programvare

For å få et fungerende styresystem må det lages et dataprogram som kan styre prosessen. I en fullstendig versjon av systemet skal programmet kunne ta inn en beskrivelse av hvordan rommet skal se ut og deretter selv finne en plan på hvordan roboten og søylene skal flytte seg stegvis for å nå målkonfigurasjonen. I tillegg skal det også kunne være flere roboter som jobber i systemet samtidig for å redusere konstruksjonstiden. Da blir også samspill mellom flere roboter noe som må løses for å hindre kollisjon og unngå at byggeklosser mister støtte. Dette er ikke problemstillinger som tas hensyn til i dette prosjektet.

Programmet som lages i dette prosjektet skal ta seg av kjøring av en robot og flytting av søyler, og må ha kontroll på at utførelsen skjer i riktig rekkefølge. Det er viktig at de gjeldene søylene flytter seg ut av veien før roboten kjører. Ikke alle søylene kan flytte seg samtidig siden det alltid må være nok støtte til konstruksjonen over. I programmet skal det være mulig å velge hvor en klosse skal plasseres, så skal programmet lage ruten til roboten selv og hvilke søyler som roteres på veien.

4.2 Løsningsalternativer

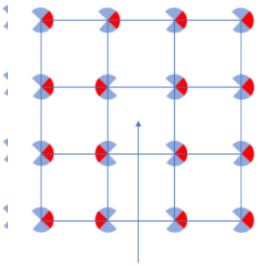
Dette underkapittelet vil være forklaring på forskjellige løsningsalternativer vi har og valgene vi tar før vi begynner realiseringen av løsningen.

4.2.1 Løsningsalternativ for søyler 1

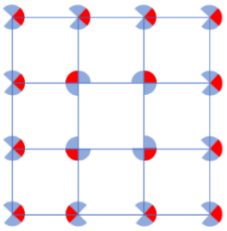
Det ene alternativet for design av søylene vil være å ha søylen formet som en 1/4 sirkel, se figur 13. Søylen vil være festet i ene hjørnet på en rund plate i bunn. Når platen da roterer vil det være mulig å flytte søylen ut av veien uten å ha noen forskyvning av hele søylen. I toppen vil det bli festet en plate som er formet som en 3/4 sirkel. Denne åpningen vil gjøre det mulig å rotere søylen til en posisjon der det ikke blir problem å få klossen løftet opp til gulvoverflaten. Samtidig kan denne utformingen støtte fire klosser.

Figur 13 – Utforming av Søylealternativ 1 [15]

En viktig fordel med denne løsningen er at søylene kan flytte seg nok til at store nok klosser kommer gjennom. Dette fører til at det ikke vil være noen gliper mellom hver klosse når konstruksjonen er ferdig. Hver søyle trenger heller ikke så stor plass for å få dette til. Se figur 14 og 15.

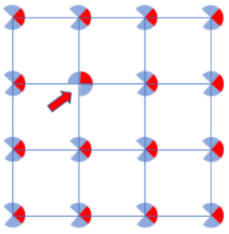


Figur 14 - Søyler når klossen flyttes [15]



Figur 15 - Søyler når klossen løftes [15]

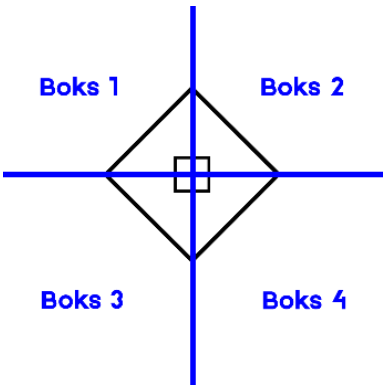
Problemene med denne løsningen er at den krever mer samordning mellom naboliggende søyler. Som vist på figur 16 vil rotasjon føre til at en byggekloss mangler støtte i et av hjørnene. Konstruksjonen skal kunne støttes om tre søyler holder vekten, men vi må unngå at flere søyler mister støtte samtidig. Løsningen vil kreve at rotasjonsposisjonen til søyler i nærheten ikke lager en konfigurasjon som medfører at klosser faller ned.



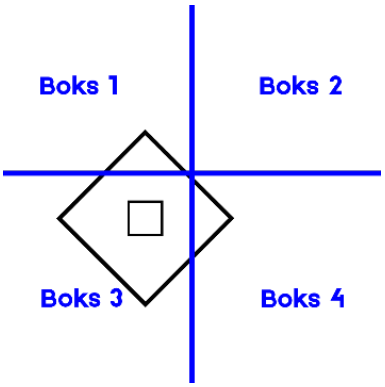
Figur 16 - Åpning som kommer når søylen roterer [15]

4.2.2 Løsningsalternativ for søyler 2

En annen løsning på problemet med å frakte byggeklosser under søylene er å kunne skyve søylene lineært langs diagonalen. Ser man på figur 17 og 18 under kan man tenke seg at de blå firkantene er byggeklosser som hviler på den største sorte firkanten. Den største sorte firkanten er topplaten av søylen. Selve søylen er den minste firkanten. Med fire byggeklosser på søylen vil søylen kunne støtte de tre andre byggeklossene når søylen beveger seg bort for å gjøre en byggekloss tilgjengelig.

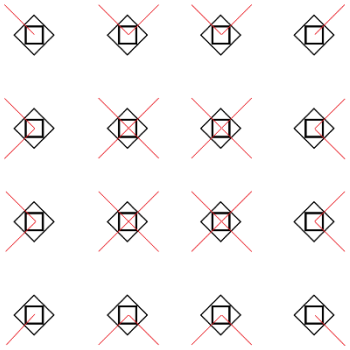


Figur 17 – Standard posisjon av søylealternativ 2 [32]



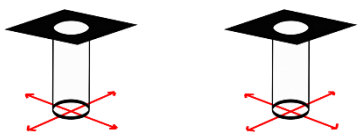
Figur 18 – Froskyvning av søylen [32]

I figur 19 viser de røde linjene hvilken retning søylene må kunne bevege seg. Her ser man at alle i midten må bevege seg i fire retninger, skulle man utvidet systemet i figuren måtte enda flere kunne bevege seg i fire retninger.

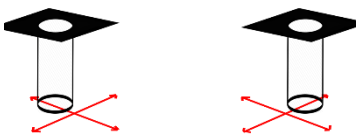


Figur 19 – Oversiktsbilde over søylesystem [32]

Figur 20 og 21 viser en 3-dimensjonal-tegning på hvordan det vil åpne seg for roboten.



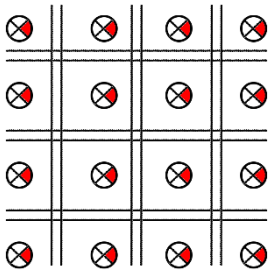
Figur 20 – 3D tegning standard posisjon [32]



Figur 21 – 3D tegning av forskjøvet posisjon [32]

4.2.3 Løsningsalternativ for styring av robot 1

Det er flere måter å bygge en robot som skal levere klosser. Et alternativ til fremdriften til roboten er skinner. På figur 22 er de sorte linjene ment som skinner. Fordelene med skinner vil være presisjonen i leveringspunktet til byggeklossene i forhold til vanlig flate og hjul. Roboten vil kun trenge å bevege seg i to retninger uten feilmargin ved rotasjon eller lignende. Det negative med denne løsningen er en mer utfordrende utforming.



Figur 22 – Oversikt over søylesystem med skinner [32]

4.2.4 Løsningsalternativ for styring av robot 2

Det er flere løsninger med robot som kjører med hjul. En type er en robot som er firkantet og har to drivende hjul i hver retning, noe som gjør at den ikke trenger å rotere for å kjøre i en annen retning. Fordelen med dette er at man unngår at en upresis rotasjon gjør at den kjører feil. Ulempen blir at man må løfte bort de hjulene som ikke brukes, slik at roboten ikke holdes igjen. Dette kan løses ved å skyve det ene settet med hjul høyere enn det andre. Roboten vil vite posisjonen sin ved å ha tilkoblet en sensor som leser av punkter etter hvert som roboten beveger seg.

4.2.5 Løsningsalternativ for styring av robot 3

Det kan også være en løsning å ha tre hjul. To hjul som kjører og et støttehjul. Dersom de drivende hjulene står midt under roboten vil det være mulig å rotere uten å kollidere i noen søyler. Ulempen med dette er at dersom rotasjonen er litt upresis kan det hende den ikke kommer seg mellom de neste søylene. Dette kan hjelpes med å ha noen spor i bakken som roboten kan kjøre inn i slik at den kommer rett vei. Slik kan man kalibrere retningen underveis. Denne løsningen vil bruke samme løsning for å finne posisjonen som alternativ 2.

4.2.6 Løsningsalternativ for styring av robot 4

Vi hadde kunne valgt andre løsninger for levering av byggeklosser til søylekonstruksjonen. Et eksempel er samlebånd som kunne beveget seg under konstruksjonen i stedet for robot. Vi synes det ble litt for store utfordringer med løftefunksjon på et samlebånd og ønsket også å ha en mer fleksibel løsning. Kravet om trådløs kommunikasjon førte også til at vi ville bruke en robotløsning, samt det at vi har erfaring fra robotikk i studieløpet.

4.2.7 Løsningsalternativ for mikrokontroller

Fra før har vi erfaring med Arduino UNO som en del av studieløpet. Erfaringen innebærer både styring av motor/servo og kommunikasjon mellom flere Arduinoer. Dette kommer godt med for å løse problemstillingene. Vi har derimot ikke erfaring med Raspberry Pi og denne løsningen vil derfor kreve opplæring. Vurderingen vi gjør er også at Arduino vil være bedre å bruke siden roboten skal gjøre en gjentagende syklus. I tillegg vil Arduino sine kode-biblioteker til serverkommunikasjon, motorstyringer, sensordata og eksempler på andre løsninger gjøre det enklere å løse våre problemstillinger.

4.3 Konklusjon

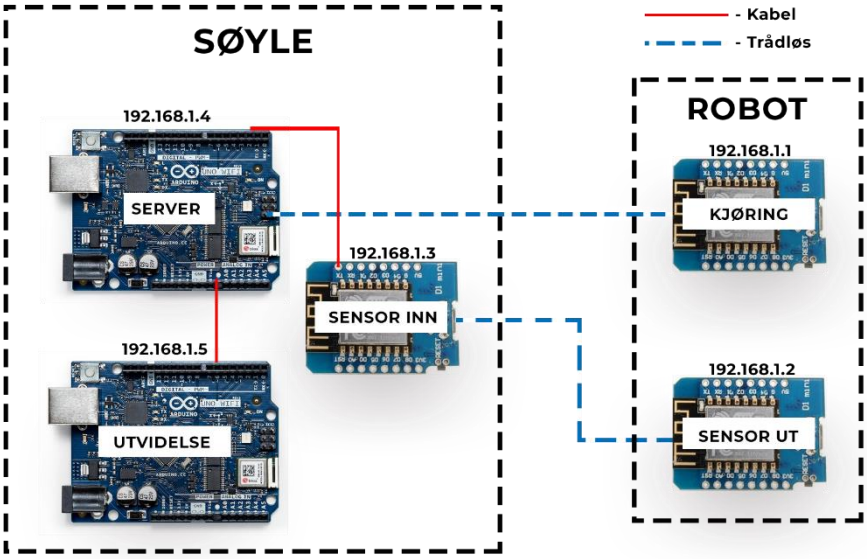
Konklusjonen er at vi velger løsningsalternativ 1 for søyle, som er den roterende søylen med forskjellig topp og bunn. Grunnen til det er at løsningen er enklere å utføre ved hjelp av rotasjon av søyler i forhold til fysisk forskyvning av søyler, som er løsning 2. Løsning 1 er litt mer utfordrende når det kommer til programmeringen, men skal være overkommelig. Med tanke på styringen tenker vi at den beste løsningen vil være løsning 3. Den krever minst å implementere, samtidig som den vil kunne gjennomføre jobben. Utfordringen med løsning 3 er presisjon ved levering av byggekloss, men det vil kunne brukes løsninger som for eksempel et spor roboten må kjøre i ved leveranse og henting av byggekloss. Vi ønsker også å bruke Arduino som mikrokontroller siden den er bedre å bruke til gjentagende systemer og konkluderer med at den vil gjøre jobben like bra som en Raspberry Pi.

5 Realisering av valgt løsning

I dette kapittelet vil vi ta for oss hvilke løsninger vi har kommet frem til i forhold til det som var målet i kravspesifikasjonene, vi vil forklare hva vi har gjort, hva hensikten med det vi har gjort er og hvordan det virker. I dette kapittelet vil språket bli mer teknisk og det kan være lurt å bruke referansene hvis man ikke skjønner hva som er ment.

5.1 Hardware

Figur 23 under viser hvordan mikrokontrollerne i vårt oppsett snakker sammen. Hvis vi ser på det vi har kalt for server så er den fysisk tilkoblet utvidelsen for å utvide antall søyler og en ESP D1-mini for å motta sensordata fra roboten. Serveren sender også signal til roboten som bestemmer om den skal kjøre fremover, venstre, høyre, bakover eller stoppe. De signalene mottas i det som kalles kjøring og det sendes sensordata tilbake ved bruk av den som heter sensor ut.

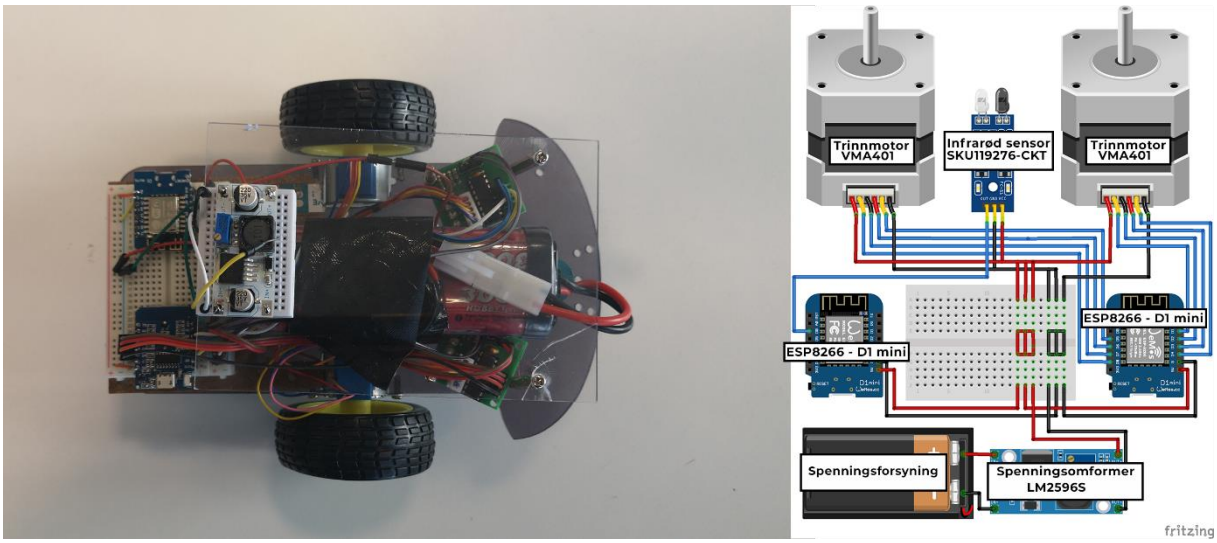


Figur 23 – Oversikt over kommunikasjonen systemet har ved hjelp av kabler og trådløse signaler [32]

5.1.1 Robot

I kravspesifikasjonene var det ikke krav til å bygge en fysisk robot til prosjektet sitt styresystemet. Men ettersom vi trengte motorer til testing av systemet, og vi valgte et løsningsalternativ for robot som bare trengte to hjul og et styrehjul, ble det bygget en fysisk modell, se figur 24 og 25. Roboten vi kjøpte var fra et byggesett [34]. DC-motorene [18] ble koblet opp mot en ESP [6] og det ble koblet en infrarød sensor til en annen ESP, som blir forklart grundigere i kapittel 5.1.3. Roboten ble bygget på kort tid og påvirket ikke fremdriftsplanen noe særlig. Det tok ikke lang tid før presisjon ble et problem med DC-motorer og vi måtte endre motorene på roboten til trinnmotorer. Valget å bygge en fysisk modell lønnet seg for prosjektet ettersom trinnmotorene bruker et annet kode-bibliotek enn servo og DC-motorer og krevde endringer i koden. Roboten består nå av to trinnmotorer, to ESP-er, batteri med spenningsomformer og en infrarød sensor. Virkemåten til roboten er at begge ESP-ene er koblet opp til en server. Fra denne serveren mottar den ene ESP-en et signal til sin IP-adresse som sier enten fremover, bakover, venstre, høyre eller stopp. Når ESP-en mottar og leser signalet sender den så ut

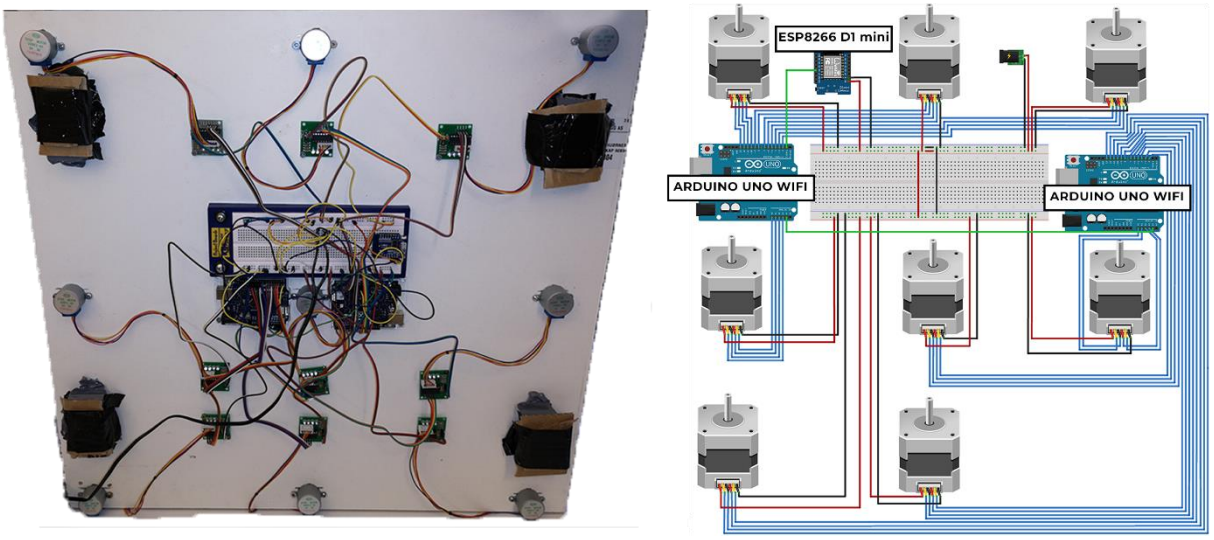
strøm til de utgangene som får motorene til å kjøre i den retningen som er ønsket. Den andre ESP-en er koblet til en infrarød sensor og sender et signal til en annen IP-adresse. De forskjellige komponentene til roboten forklares i mer detalj senere i kapittelet.



Figur 24 og 25 – Bilde av robot og koblingsskjema som viser oppkoblingen til roboten [23] [32]

5.1.2 Søylekonstruksjon

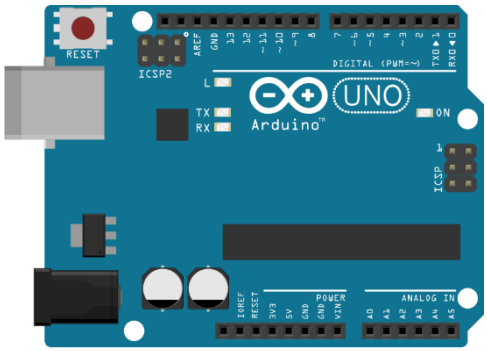
Kravspesifikasjonene til prosjektet krever at søylene til styresystemet skal rotere seg i forhold til roboten. For å enklere løse kravet bygget vi en konstruksjon som gjorde det enklere å visualisere kildekoden i et praktisk eksempel. På eksempelet under kan man se at overgangen til trinnmotorer førte til at motorene i søylekonstruksjonen nå trengte fire utganger fra Arduino UNO WiFi kortet. Som var mer enn servoene som bruker en utgang. Det betyr nå at Arduinoen vil kunne sende signal til maks fire forskjellige trinnmotorer. Utvidelse av dette antallet blir tatt opp i diskusjonskapittelet. Men måten vi har løst det på er ved å utvide ved hjelp av å koble sammen flere Arduinoer som vist ved den grønne linjen i figur 27. På figur 26 og 27 ser vi undersiden til søylekonstruksjonen.



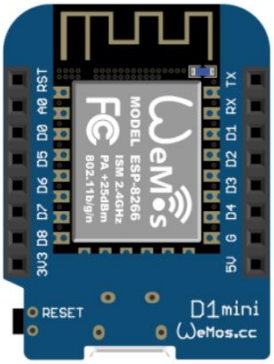
Figur 26 og 27 – Søylekonstruksjon med koblingsskjema som viser sammenkoblingen av de forskjellige enhetene [23] [32]

5.1.3 Mikrokontrollere

Vi har brukt to typer mikrokontrollere til systemet. Arduino Uno WiFi rev2 [3] og ESP8266 D1-mini [6], se figur 28 og 29. Arduino Uno WiFi har 14 digitale og 6 analoge innganger/utganger, en USB-tilkobling, en strømkontakt og en nullstillknapp. Den programmeres i programmeringsprogrammet Arduino IDE [20] på datamaskin og koden overføres til mikrokontrolleren gjennom USB-tilkobling. ESP8266 D1-mini har 11 digitale og 1 analog inngang-/utganger og programmeres også gjennom Arduino IDE, men har en mikro-USB-tilkobling. En forskjell på de to komponentene er at de bruker forskjellige biblioteker i Arduino IDE. Vi bruker Arduino Uno WiFi opp mot søylene ettersom den har flere porter (inngang-/utganger) enn ESP-en, men vi bruker ESP-en på roboten siden den er mindre og roboten trenger færre porter.



Figur 28 – Arduino Uno [23]

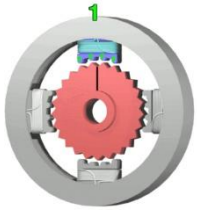


Figur 29 – ESP8266 D1mini [23]

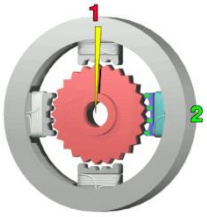
5.1.4 Trinnmotor

Etter problemene vi møtte ved 360-graders servomotor ble valget av søylemotor endret til trinnmotor [29], se figur 30. Trinnmotor er en børsteløs likestrømsmotor som har full rotasjon ved hjelp av en rekke like trinn. Motorens posisjon kan da beordres til å bevege seg og holde seg ved ett av disse trinnene uten noen posisjonsføler for tilbakemelding (åpen loop-kontroller). Trinnmotoren består av magneter som er plassert rundt et tannhjul. For å kjøre brukes en og en magnet til å flytte tannhjulet ett og ett steg rundt. Utførelsen kan gjøres i begge retninger. Denne motoren er svært presis. Virkemåte vises i figurene 31-34.

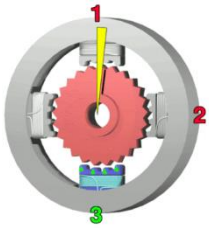
Figur 30 – Trinnmotor [13]



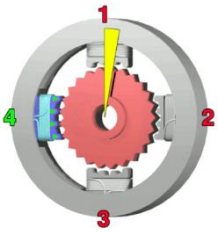
Figur 31 – Trinnmotor steg 1 [29]



Figur 32 – Trinnmotor steg 2 [29]



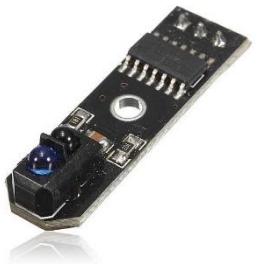
Figur 33 – Trinnmotor steg 3 [29]



Figur 34 – Trinnmotor steg 4 [29]

5.1.5 Sensorikk

For å vite posisjonen til roboten valgte vi å bruke en infrarød sensor [14], se figur 35. Den infrarøde sensoren vi valgte virker med signalene 0/1 ved å reagere på refleksjonen til det infrarøde signalet den sender ut. Det vil si at hvis sensoren måler mot en hvit flate vil sensoren være aktiv høy (1) og vil deretter endre til aktiv lav (0) når den måler mot en helt sort flate som ikke reflekterer. Vi bruker sensoren til å reagere når roboten beveger seg over en liten sort firkantet flate. Serveren mottar signalene og teller deretter hvor i systemet roboten er, samtidig som den sender signaler på hvilken retning roboten skal bevege seg.



Figur 35 – Infrarød sensor [14]

5.1.6 Spenningsforsyning

Styresystemets spenningsforsyning er en svært viktig ting å tenke på skal løsningen virke som forventet. Hvis man ser tilbake på figur 26 er søylekonstruksjonen tilkoblet en separat strømkilde, som vil være en stikkontakt. I leddet mellom koblingsbrettet og stikkontakten vil all strøm som søylekonstruksjonen bruker gå over to kabler, det er da veldig viktig at de kablene har en dimensjon som er høy nok til å tåle strømmen som trekkes av søylekonstruksjonen [26]. Hvis man som et eksempel hadde brukt to svært tynne kabler der, og ikke tenkt på dette, så ville fordelingen ha fått et spenningsfall, som hadde ført til at Arduinoene og trinnmotorene blir tildelt for lav spenning. Det ville da kunne oppstå feil i både motorene og Arduinoene. Det er også viktig med riktig dimensjon på kablene for sikkerhetsgrunner som at for høy strøm (last) på tynne kabler kan skape varmgang.

5.2 Software

5.2.1 Arduino IDE

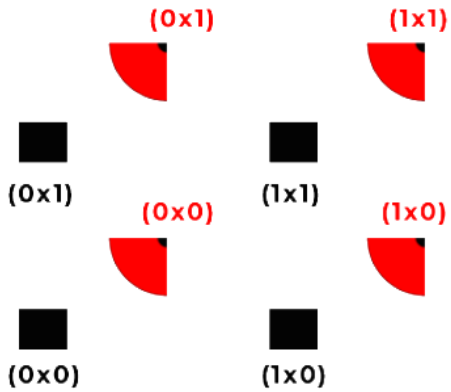
Arduino IDE er en programvare som brukes til å skrive og laste opp kode til Arduino kompatible mikrokontrollere. Koden skrives på datamaskinen først og så lastes den over til mikrokontrolleren og kjøres derfra. Det finnes flere alternativer, men Arduino IDE er det programmet som Arduino selv har gitt ut. Programmet støtter programmeringsspråkene Java, C og C++ [31]. Vi har erfaring med både programmet og C++ fra før gjennom utdanningen vår. Et typisk program i Arduino IDE består av to deler. En setup og en loop. I loopen skriver man den koden som man skal kjøre kontinuerlig så lenge mikrokontrolleren er aktiv. Denne kjører om og om igjen. I setup setter man den koden som man vil bare skal kjøres en gang før programmet starter og gjentas. Når programmet er ferdig kan man laste det opp på mikrokontrolleren ved å koble den til datamaskinen. Et program som er lastet opp vil kjøre på mikrokontrolleren så lenge den er aktiv. Dersom man kutter strømmen eller starter mikrokontrolleren på nytt vil også programmet starte på nytt. Mikrokontrolleren vil ikke huske hvor langt den var kommet.

5.2.2 Arduino biblioteker

Arduino IDE inneholder en rekke biblioteker som kan utvide funksjonaliteten til programmet. Biblioteker som legges til inneholder ofte koder som gjør det mulig å kommunisere med forskjellige typer hardware eller koder som kan manipulere data. I vår løsning trengte vi å legge til biblioteker for å kunne styre trinnmotorene våre og for serverkommunikasjon. Serverkommunikasjonen krevde to forskjellige biblioteker basert på hvilke brett koden skulle lastes opp på. ESP og Arduino brett brukte to ulike bibliotek, men veldig lik kode.

5.2.3 Valg av algoritme

I vår oppgave er det relevant å lage en algoritme for hvordan roboten skal bevege seg i systemet og hvilke søyler som skal rotere til enhver tid. For å gjøre systemet oversiktlig valgte vi å sette opp systemet som to rutenett. Ett rutenett for søylene og ett for posisjonene til klossene. Slik ble det lett å oversette systemene til en matrise i datakoden. Punkt i systemet kunne da adresseres ved hjelp av x- og y-koordinater. Figur 36 viser søylene i rødt og klosseposisjoner i svart og hvordan de er nummerert i forhold til hverandre.



Figur 36 – Koordinatsystemet med tilhørende koordinater, der de sorte firkantene er stoppunkt og de røde figurene er søyler [32]

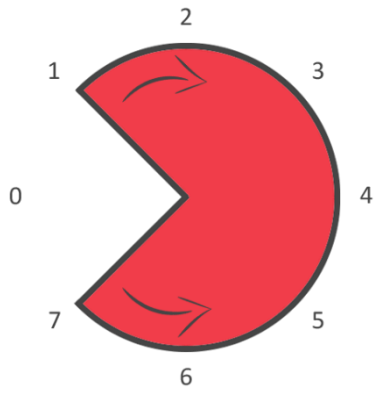
For styring av roboten kan vi nå bruke posisjonene til klossene til å navigere. Koden for kjøringen jobber nå slik at den får oppgitt en startposisjon og en destinasjon, og teller seg frem til om koordinatene stemmer med destinasjonen. Hver klosseposisjon i systemet har et merke i bakken som roboten fanger opp med den infrarøde sensoren. Merket er en svart firkant som gjør at den infrarøde sensoren under roboten returnerer spenningssignal 0. Dette tolkes av koden som at den har nådd en ny posisjon og legges til i enten x- eller i y-koordinaten. La oss si at roboten starter

i posisjon (0, 0) og destinasjonen dens er (3, 2). Den vil da begynne å kjøre i x-retning. Når den passerer merket som ligger på (1, 0) vil koden vite dette og la roboten fortsette å kjøre. Den vil passere merker helt til den når (3, 0), der vil den stoppe. Nå har den nådd en posisjon slik at x-koordinaten stemmer og vil derfor snu seg 90° til venstre. Roboten starter så å kjøre igjen, men nå langs y-aksen. Tilsvarende her kjører den helt til den har passert nok punkter slik at den kommer til rett posisjon i forhold til y-koordinaten. Slik når roboten rett posisjon og får levert klossen til rett plass. Etter at den har levert klossen skal roboten tilbake til startposisjonen. Dette gjør koden med å kjøre stegene i motsatt rekkefølge. Roboten rygger tilbake slik at posisjonen blir (3, 0), snur og rygger så langs x-aksen til den når (0, 0). Koden tolker også dersom startposisjonen har en høyere y-verdi enn destinasjonen. Den vil da svinge til høyre og telle seg nedover.

Innimellom kjøringen til roboten skal søylene også roteres. Dette skjer før roboten begynner å flytte seg langs en akse. Koden sjekker hvor langt roboten skal kjøre, og flytter alle søylene som den skal passere langs x-aksen pluss de som er rundt punktet der roboten snur seg. Rutenettet til søylene er en halv koordinat til høyre og opp som vist på figur 36. Slik kan koden bruke y- koordinaten til roboten til å vite hvilke søyler som skal roteres og i hvilken retning de skal roteres. Søyler som har lik y-koordinat skal roteres oppover, og de som har en lavere skal roteres nedover. Etter at roboten er kommet halvveis og har snudd skjer dette igjen, men nå langs y-aksen. Koden roterer alle søylene den skal passere og de rundt endepunktet ut av veien. Søyler med lik x-koordinat snur seg mot høyre og de med en lavere snur seg mot venstre. Dette skjer på lik måte når roboten kjører tilbake.

Koden inneholder også en egen syklus i destinasjonspunktet. Der er det meningen at en klosse skal bli levert eller hentet. Modellen vår inneholder ingen løftefunksjon, men den tar fortsatt høyde for hvordan søylene må flytte seg dersom den hadde det. Dersom roboten har nådd destinasjonen vil derfor søylene snu seg slik at platene på toppen lager en åpning. Slik kan en klosse heises opp eller ned gjennom åpningen. Etterpå flytter søylene seg tilbake igjen slik at åpningen lukkes.

For å ha kontroll på hvilken retning søylene står i har vi valgt å nummerere de åtte ulike posisjonene som er nødvendige å ha, se figur 37. Hver posisjon har et nummer fra 0 til 7. Koden er avhengig av at søylene står i riktig startposisjon når den starter. Dette er fordi den begynner med å lagre posisjonene til alle søylene som posisjon 0. Dette kunne vært bedre dersom søylene selv kunne vite i hvilken posisjon de sto i og så rapportere dette når programmet starter. Løsningen nå er at søylene flyttes tilbake til startposisjonen når hele programmet er ferdig å kjøre. Nummereringene blir brukt til å vite hvor langt søylene skal rotere. Vi vet hvor langt det er mellom to posisjoner og bruker differansen mellom start og destinasjon til å multiplisere denne avstanden. Dersom en søyle skal flytte seg fra 2 til 4, multipliserer vi avstanden med 4 – 2 = 2. Det finnes også logikk i koden som gjør at ikke søylene roterer unødvendig langt. Dersom den skal fra 1 til 7 velger den heller å gå 2 posisjoner mot klokken i stedet for 6 posisjoner med klokken.



Figur 37 – Toppen av en søyle [32]

Hele syklusen fra startposisjonen til roboten og til den er tilbake til start igjen utgjør mesteparten av logikken og styringen av systemet. Denne skal kunne kjøres så mange ganger som det er nødvendig for å kunne konstruere den ønskelige utformingen. På forhånd må det derfor lages en liste over alle

destinasjonene som roboten skal besøke. Når koden begynner å kjøre vil den kjøre syklusen en gang for hver destinasjon helt til den har kommet seg gjennom hele listen. Det har ikke vært en del av prosjektet å lage denne listen på en logisk måte basert på hvordan man ønsker romkonfigurasjonen.

5.2.4 Server kommunikasjon

For at logikken skal fungere må det være kommunikasjon mellom roboten og serveren. Vi vil ikke at utformingen til konstruksjonen skal ligge på roboten slik at det ikke skal være nødvendig å hente roboten hver gang listen over destinasjoner endrer seg. Målet er at roboten skal kunne kjøre av seg selv uansett hva som endres hos serveren. Derfor var det viktig å ha trådløs kommunikasjon. Alle mikrokontrollene utenom utvidelsen er koblet opp mot et nettverk. Når informasjon skal sendes fra server til robot blir det sendt en tekststreng til IP-adressen til mikrokontrolleneren kalt «kjøring» på roboten (se figur 23). Deretter kan mikrokontolleren lese det som den har mottatt. Andre veien sender «sensor ut» på roboten tilbake sensordata fra den infrarøde sensoren til «sensor inn» på serversiden.

Den infrarøde sensoren returnerer kontinuerlig data og all informasjonen er ikke nyttig for oss. Sensoren gir spenningssignalet 0 dersom overflaten foran seg ikke reflekterer noe lys og 1 ellers. Derfor bruker vi svarte merker i systemet slik at vi kan tolke signalet 0 som at roboten er over et slikt merke. Istedenfor at roboten kontinuerlig sender sensordata til server så reduserer vi kommunikasjon ved å la roboten ta seg av noe av dataprosesseringen lokalt. Roboten sender bare signal til master når det er endring fra 0 til 1 i det infrarøde signalet. Slik kan serveren tolke alle signal den får inn som at roboten har nådd ett nytt punkt.

Roboten må under drift motta informasjon fra server slik at den kan vite når den skal kjøre, svinge eller stoppe. Når styringsprogrammet vil at roboten skal kjøre fremover vil den sende et signal til roboten. Dette tolker roboten som at den skal kjøre begge hjulene slik at den beveger seg fremover og gjør dette helt til den mottar et annet signal. Signaler om å kjøre framover eller bakover må lagres på roboten fordi koden for å drive en trinnmotor er blokkerende og basert på hvor mange trinn den skal flyttes seg. Det vil si at man først oppgir antall trinn og så mens koden utfører det kan ikke noe annet gjøres. Derfor må roboten ta et steg av gangen slik at den innimellom kan få beskjed om å gjøre noe annet. Dette betyr også at bare et hjul kan kjøres av gangen, men alt skjer så fort at det ikke merkes på kjøringen. Ved svingning til en av sidene er dette litt annerledes. Siden alle svinger er på 90° kan dette gjøres i en omgang. Signalene som blir sendt er i form av ordene «forward», «backward», «left», «right» og «stop». Slike ord blir kalt tekstrenger i koden og blir sendt over IP-en. Når «left» blir sendt starter roboten det høyre hjulet fremover og det andre bakover, som gjør at den roterer til venstre mens den står i ro, og motsatt når «right» blir sendt. Stopp blir sendt når roboten skal stå helt rolig.

5.2.5 Utvidelse med flere søyler

På grunn av begrenset antall utganger og at hver av trinnmotorene krever fire porter, kan ikke en Arduino UNO WiFi styre mer enn fire søyler. Vår løsning for å fikse dette ble å legge til flere Arduinoer. Disse Arduinoene kan vi kalle utvidelser og de skal ikke være tilkoblet nettverket eller kommunisere med robot, men bare styre sine søyler. Derfor blir det en Arduino som er serveren og har hovedstyring, mens de andre mottar signaler fra server for å rotere søylene de styrer. Samtidig må alle Arduinoene ha kontroll på hvor stort systemet er og hvordan utformingen skal bli. Dette er fordi alle må bruke

samme koordinatsystem slik at alle vet når deres del av systemet er aktivt. Det vil også si at dersom systemet eller konstruksjonen som skal bygges endres må alle Arduinoene lastes opp med oppdatert informasjon.

```
Stepper steppermotor[matriseX][matriseY] =
{
  {Stepper(64, 8, 6, 7, 5), Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1)},
  {Stepper(64, 12, 10, 11, 9), Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1)},
  {Stepper(64, A0, A2, A1, A3), Stepper(64, 0, 3, 2, 4), Stepper(0, 1, 1, 1, 1)}
};

Stepper steppermotor[matriseX][matriseY] =
{
  {Stepper(0, 1, 1, 1, 1), Stepper(64, 12, 10, 11, 9), Stepper(64, 4, 2, 3, 0)},
  {Stepper(0, 1, 1, 1, 1), Stepper(64, A5, A3, A4, 13), Stepper(64, 8, 6, 7, 5)},
  {Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1)}
};
```

Figur 35 – Kodesnutter av Arduinokode som viser hvordan trinnmotorer blir definert hos to ulike Arduinoer [19]

Alle søyler som er koblet til en Arduino må legges til på riktig plass i systemet. I figur 35 viser to kodesnutter for to ulike Arduinoer som har fire søyler hver i et system på ni søyler, i dette eksempelet er ikke den niende koblet til. Det kodesnuttene utfører er å oppgi for koden at det eksisterer ni trinnmotorer (steppere). De er plassert inn i en matrise slik at det senere skal være lettere å finne igjen til bestemte motorer. Matrisen representerer samtidig hvordan trinnmotorene er plassert i det fysiske systemet. MatriseX og matriseY er konstanter som er gitt før denne kodesnutten, men som sier noe om hvor stor matrisen er. I dette tilfellet er begge 3. MatriseX representerer x-aksen og sier noe om hvor mange rekker det er. MatriseY er y-aksen og antall kolonner. Når man skal velge ut en trinnmotor skriver man steppermotor etterfulgt av x- og y-koordinater. Tellingen begynner fra 0 og derfor finner vi da steppermotor[0][0] oppe i venstre hjørne, og steppermotor[2][2] nede til høyre. Hver av trinnmotorene må oppgis med fem variabler. De fire siste er hvilket av de nummererte portene på Arduinobrettet motoren er fysisk koblet til. Der det står A0-A5 betyr det at porten også kan tolke analoge signal, men det er ingen forskjell på disse når det gjelder digitale signal fra motorene. På grunn av måten stepper-funksjonen er satt opp må man dersom man bruker portene fra 12-9 skrive de opp i rekkefølgen 12, 10, 11, 9 [35]. Det første variabelen er et tall som kan variere fra en motor modell til en annen, og er et tall på hvor mange steg det er i en rotasjon til motoren. I dette tilfellet er den 64. Alle trinnmotorer som den gitte Arduinoen ikke styrer får oppgitt porter som ikke er koblet til og 0 som antall steg. Når logikken kjører og søylen i posisjon [1, 2] skal roteres vil alle Aduinoene i systemet prøve å aktivere, men det er bare en av de som faktisk har noe styring på den valgte søylen.

Måten alle utvidelsene som driver søylene kjører samtidig er at serveren sender ut et spenningssignal til disse når det er på tide å rotere søyler. Dette signalet blir satt lavt igjen når søylene er ferdig å rotere og serveren skal sende kjørekommandoer til roboten. Da står utvidelsene bare og venter til søylene skal brukes igjen og når den tid kommer sender serveren ut et nytt spenningssignal. Siden alle Arduinoene har samme informasjon om systemet og destinasjonene den skal til, og syklusen er lik fra en destinasjon til en annen, kan alle alltid være enige om hvor i systemet roboten er og hvor langt i syklusen den er.

5.3 Kildekode

5.3.1 Programflyt

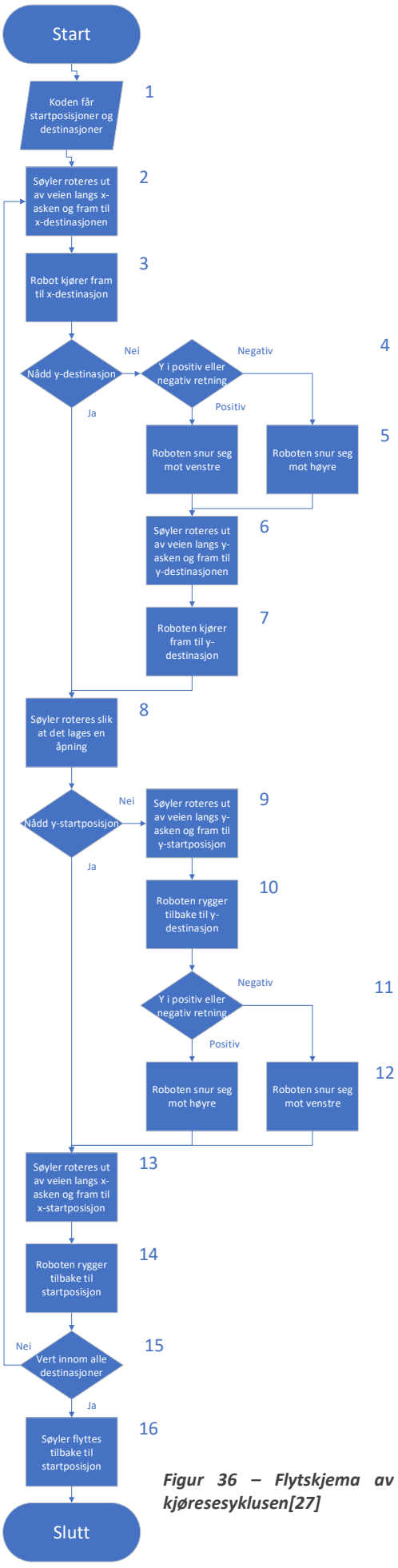
Figur 36 viser et flytskjema over styringsprosessen til roboten og søylene, og gjenspeiler koden som ligger på serveren. De rektangulære rutene viser kode som blir utført, mens de roterte rutene representerer problemstillinger som programmet må ta høyde for.

Programmet starter med en setup del der den setter en del konstanter som brukes i løpet av programmet (1). Dette innebærer blant annet startposisjoner, liste med destinasjoner og andre variabler som programmet bruker på å holde orden på at prosessen kjører rett. Alle disse verdiene må skrives inn i koden på datamaskinen først før det lastes over til mikrokontroller. Dersom det skal gjøres endringer i koden er det stort sett her, og etter endringer må koden lastes over på nytt. Logikken som kommer etterpå bruker verdiene i denne delen og kjører ut fra dem.

Etter dette kommer selve kjørekoden. Koden plukker første punkt i en liste av destinasjoner og plukker ut x- og y-verdiene til et punkt om gangen. Kjørekoden kjører syklusen vist på figuren for hvert av de punktene og slutter når den har kommet gjennom hele listen. Etter det vil roboten være ute av systemet og søylene flyttes tilbake til startposisjonen (16).

Syklusen (2-15) består av at roboten kjører frem og tilbake til destinasjonen, men underveis må koden ta høyde for en del variabler slik at den ender opp der den skal. Roboten starter utenfor søylesystemet og koden antar at retningen roboten peker når programmet starter er langs x-aksen. Koden begynner med å rotere søylene fra startposisjonen og helt bort til x-koordinaten til destinasjonen (2). Deretter begynner roboten å kjøre (3). Koden holder orden på roboten sine koordinater i systemet og bruker den infrarøde sensoren og merkene i bakken til å registrere og oppdatere posisjonen. Når x-posisjonen til roboten er lik som x-koordinaten til destinasjonen stopper den. Her må koden gjøre to undersøkelser (4). Hadde startposisjonen lik y-koordinat som destinasjonen? I så fall har den nådd destinasjonen. Dersom ikke, er y-koordinaten til destinasjonen et lavere eller høyere tall en y-koordinat til roboten sin posisjon? Basert på dette velger koden hvilken vei roboten skal snu (5). Deretter roteres søylene langs y-aksen ut av veien og roboten kjører frem til destinasjonen (6 -7).

Når roboten har nådd destinasjonen, stopper den og søylene rundt roteres slik at topplatene lager en firkantet åpning over den (8). Dersom systemet skal utvides slik at roboten også kan løfte klosser



Figur 36 – Flytskjema av kjøresesyklusen[27]

er det ikke så mye logikk som må legges til koden. Det vil være to ulike senarioer. Enten skal roboten hente en klosse eller så skal den levere en klosse. Dersom den skal levere en klosse, må klossen som leveres først løftes opp under den som ligger over slik at den kan hvile på den. Etter det må søylene flyttes bort slik at klossen kan løftes over søylene. Deretter må åpningen lukkes og roboten kan senke ned igjen. Dersom en klosse skal hentes må det samme skje i motsatt rekkefølge. Roboten må først få kontakt med klossen som ligger oppå søylene før søylene kan flyttes bort. Deretter går roboten ned til en posisjon slik at en klosse som eventuelt ligger over kan få støtte når søylene så roteres igjen. Til slutt løftes klossen helt ned. Denne prosessen vises godt i figur 6 i kapittel 1.4.1.

Etter dette skal roboten tilbake til startposisjonen. Dersom den ikke allerede er på riktig y-koordinat må søylene langs y-aksen roteres ut av veien (9). De aller fleste søylene langs denne aksen er allerede ute av veien fra tidligere, så denne delen lukker bare åpningen som ble laget. Roboten rygger så til riktig y-koordinat (10). Basert på hvor den kommer fra snur den seg slik at den står med bakenden mot startposisjonen (11-12). Siste innspurt blir så å rotere søylene langs x-aksen og så rygge tilbake til start (13-14). Til slutt i syklusen sjekker koden om den er ferdig med listen eller om den skal fortsette med neste destinasjon (15). I appendiks E.1 er en bildeserie av modellen som er laget som viser et par deler av syklusen.

Utenfor systemet må det også finnes et system hvor roboten kan hente eller få levert en ny klosse, eller levere fra seg en den har hentet. Dette er ikke med i koden nå, men bare markert med en pause i startposisjonen.

5.3.2 Mikrokontrolllerkode

All koden ligger ved som vedlegg, men her går vi gjennom noen deler som har vært viktig for løsningen vår og hvordan de fungerer.

I figur 37 har vi dratt ut de delene i styringskoden til Arduinoen som gjør at den kan sende kommandoer til roboten om hva den skal gjøre. For å kunne bruke koden må man først inkludere bibliotek som støtter den. I kode som skal på Arduino må man bruke biblioteket WiFiNINA [5], mens på ESP-ene bruker man biblioteket ESP8266WiFi [6]. Utenom dette er koden for sending av strenger like på begge. I setup-delen av koden blir kode som gjør klar sekvensen i loop-delen skrevet. Før dette må konstanter som skal brukes defineres og få en verdi. Med tanke på kommunikasjonen må SSID (nettverksnavn) og passord oppgis i tillegg til IP-adressen til mottaker. Deretter i setup prøver Arduinoen å koble seg opp til nettverket. Så lenge den ikke blir tilkoblet vil den være i while-loopen og ikke kunne fortsette dersom en tilkobling ikke blir opprettet. Videre i loop-delen ser man der den aktive styringen pågår. Først defineres en klient som skal brukes. Denne blir tømt og fjernet på slutten av loopen og defineres på nytt hver gang loopen starter på nytt igjen. Det er fordi vi oppdaget

```
#include <WiFiNINA.h>

char ssid[] = "";
char pass[] = "";

IPAddress server (192,168,1,1);

void setup() {
  // put your setup code here, to run once:
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
     delay(500);
  }
}

void loop() {
  // put your main code here, to run repeatedly:
  WiFiClient client;

  if (client.connect(server, 80))
  {
    client.println("GET /forward");
    delay(100);
  }

  client.flush();
  client.stop();
}
```

Figur 37 – Arduinokode som viser hvordan mikrokontrollerne sender tekstrenger [19]

at dersom vi brukte den samme over lengre tid uten å resette den, ble kommunikasjonen dårligere over tid. Selve kommunikasjonsdelen skjer i if-setningen. Når klienten er koblet opp mot IP-adressen til serveren sender den en tekst. I dette tilfellet sender den «forward», men teksten kan endres basert på hva mottaker kan tolke. Dette blir det samme som å sende http://192.168.1.1/forward over nettverket.

Figur 38 viser hvordan en ESP kan motta og tolke teksten som kommer fra eksempelet over. En del er likt, men noen deler er gjort litt annerledes. ESP-en bruker som sagt et annet bibliotek, men måten den kobler seg opp mot nettet er lik. I tillegg til klient defineres det også en server. Dette er en server som er knyttet til IP-adressen til ESP-en. Den blir definert i starten av koden og server.begin() gjør at serveren begynner å overvåke om den mottar informasjon. Ulikt her er at klienten blir definert helt i starten av koden. Dette er fordi i motsetning til å være den som sender informasjon, brukes den her til å sjekke når det er kommet inn noe til IP-adressen ved hjelp av server.available(). Dersom det er kommet inn noe ny informasjon hopper koden inn i funksjonen checkClient(). Denne funksjonen endrer den informasjonen som kommer inn slik at man bare står igjen med «forward». Denne blir så lagret i strengvariabelen «data». Slik leser denne koden informasjonen som kommer inn slik at den så kan brukes videre til å kjøre roboten.

```
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);

const char* ssid = "";
const char* pass = "";

String  data ="";

void setup() {
  // put your setup code here, to run once:
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
     delay(500);
  }

  server.begin();
}

void loop() {
  // put your main code here, to run repeatedly:
  client = server.available();
  if (!client) return;
  data = checkClient ();
}

String checkClient (void)
{
  while(!client.available()) delay(1);
  String request = client.readStringUntil('\r');
  request.remove(0, 5);
  request.remove(request.length()-9,9);
  return request;
}
```

Figur 38 – Arduinokode som viser hvordan mikrokontrollerne mottar tekstrenger [19]

6 Testing

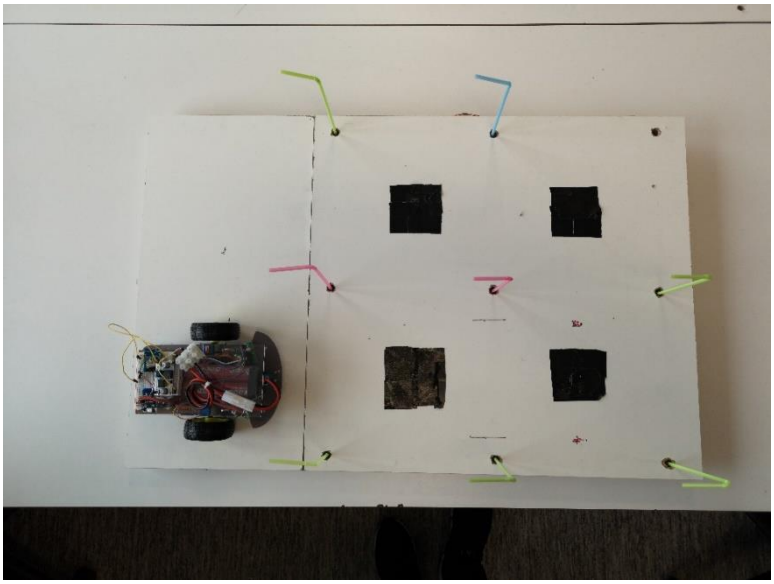
I dette kapittelet forklarer vi hvordan det forskjellige utstyret ble testet.

6.1 Trådløs kommunikasjon med Arduino UNO WiFi og ESP8266 D1 Mini

Vi ønsket at Arduinoen skulle sende trådløse signaler til roboten. Vi ønsket også at Arduinoen skulle motta trådløse signaler fra sensoren på roboten. Ved testing av at Arduinoen både sendte og tok mot signaler observerte vi at den hengte seg opp i rundt 10 sekunder til tilfeldige tidspunkt. Ofte så begynte Arduinoen fint i starten for deretter å henge seg opp, og andre ganger skjedde det tidligere. Vi brukte mye tid på feilsøking i koden og undersøkelser på internettforum. For å ikke bruke for mye tid på dette problemet begynte vi å undersøke andre løsninger. Vi skaffet oss fire ESP-er [9] og valgte å bruke en av dem til å motta signalet fra roboten. Da ble oppgaven til Arduinoen kun å sende signal til roboten, siden den fikk robotsignalet fysisk fra en ESP. Se gjerne figurene 23-27 for å visualisere løsningen. Denne metoden virket mye bedre og problemene med at det hengte seg opp forsvant. Systemet fungerte bedre når hver mikrokontroller konsentrerte seg om en ting. Etter testing konkluderte vi med at det nå var nøyaktig nok, det vil si at roboten stoppet i samme posisjon hver gang den ble kjørt i konstruksjonen. Mer om nøyaktighet tas opp i kapittel 6.6.

6.2 Testing av kode/logikk

Testing av kildekode har skjedd gradvis under hele utviklingen. Før vi begynte å lage den var vi opptatt av å ha en modell som vi kunne prøve den på. Det ville blitt vanskelig å få et bra resultat uten å få noe tilbakemelding på om det faktisk fungerer. Mål med styringskoden var å få gjennomført sekvensen som er vist i figur 36 under kapittel 5.3.1. Vi begynte med å bare kjøre roboten uten søylene og testet at den kunne nå ulike destinasjoner i systemet ved hjelp av kommandoer fra styringen og sensordata. Selve styringssekvensen hadde vi ikke så mange problemer med, men testingen avdekket svakheter med andre deler av systemet. Dette har ført til at vi blant annet har byttet motorer og antall mikrokontrollere vi måtte ha, som forklart i 4.1.2 og 5.1.2. Etter dette testet vi kode for å styre søylene. De rette søylene måtte rotere når det var nødvendig og til riktig posisjon. Neste steg var så å slå sammen de to delene slik at søyler og robot kjørte i riktig rekkefølge. Det siste som ble gjort var å utvide med flere søyler styrt av en ekstra Arduino. Denne måtte kunne styre søylene like bra som hovedstyringen og sekvensen i systemet måtte holde seg lik. Figur 39 viser den ferdige modellen som ble brukt for å teste det totale systemet.



Figur 39 – Modell som ble laget for å teste systemet

6.3 Testing av servomotorer

Vi har tidligere forklart hvordan vi ville bruke servomotorer i kapittel 4 og 5.1.4. Testing av servomotoren gikk ut på å koble den opp til Arduinoen og deretter bruke servokode-biblioteket til Arduino. Det ble da klart at 360-graders servomotorer ikke kan roteres til valgte vinkler, men kun kjøres ved bruk av tidsinnstillinger. Valget ble da å gå over til trinnmotorer.

6.4 Testing av trinnmotorer

For å teste trinnmotorene brukte vi Arduino sitt stepper-bibliotek. Ved å bruke det fikk man muligheten til å sette farten og antall trinn trinnmotoren skulle utføre. Dette gjorde robot løsningen mye mer presis og gjorde til at søylene alltid ble rotert likt antall trinn. Nøyaktigheten til trinnmotoren er gitt som 2048 trinn. Det vil si at når en søyle skal rotere seg 45 grader rundt kan man dele 2048 på 8, da blir en rotasjon satt til 256 trinn. Denne informasjonen ble tatt fra databladet til trinnmotoren og det ble ikke utført nærmere tester ettersom vi kunne se at det virket.

6.5 Slutt-test av system

Vi gjorde til slutt en test av hele det ferdige systemet der vi så på gjennomkjøringen til et par ulike destinasjoner i modellen vår. Flyten i systemet fungerte bra og roboten nådde riktige punkt. Sekvensene gikk i riktig rekkefølge og søylene roterte til riktige posisjoner. Testen ble gjort uten en klosse oppå roboten og hadde heller fokus på at styringen og kommunikasjonen i systemet virket.

Vi valgte også å gjøre en test for å se om bevegelsene til roboten er forutsigbare og konstante. Vi satte roboten på en startposisjon 11cm fra et stoppunkt den skulle kjøre mot. Når roboten traff stoppunktet ville den infrarøde sensoren reagere og deretter vil styringen stoppe roboten. Det vi så på var hvor mye avvik det ble når vi gjorde dette flere ganger. Det ble gjennomført 10 ganger og vi valgte den første gjennomkjøringen som et nullpunkt. I tabell 1 kan man se resultatene vi fikk. Resultatene er individuelle avvik fra det første resultatet som vi setter som 0. Resultatene er påvirket av at akslingene hjulene henger på er litt for små, noe som fører til at de står litt skeivt og er mulig å bevege på.

Test	Resultat fra første punkt
1	0
2	0,2
3	0,3
4	-0,2
5	-0,35
6	0
7	0,1
8	0,1
9	-0,2
10	0,2
	Gitt i cm

Tabell 1 – Resultat fra test av presisjonen til roboten

Testene ga avvik som er lave nok til at et system virker hvis søylene tar høyde for det. Resultatene vil også kunne bli enda bedre med en aksling som passer robotens hjul. Det vil uansett være viktig med korrigering av feilmarginen etter hvert så den ikke blir større for hver levering. Dette kunne blitt utført ved markerte områder som nullstiller robotens posisjon etter hver klosseopperasjon, eller ved en av de løsningene som diskuteres i kapittel 7.3.

7 Diskusjon

I dette kapittelet vil vi diskutere oppgavens utvidelser og mangler. Vi vil diskutere hva som kunne blitt gjort annerledes og hva som kan hjelpe videreutviklingen av prosjektet.

7.1 Begrenset omfang

Det opprinnelige prosjektet var ment å være et samarbeid mellom maskin- og marinfag, institutt for elektrofag og institutt for data- og realfag. Planen var at maskinstudenter skulle lage søyler og robot, elektrostudenter skulle lage elektro og styresystem, og datastudenter skulle lage en 3-dimensjonal-datagrafikk av systemet. Ettersom prosjektet kun har to studenter fra institutt for elektrofag ved studiet automatisering, måtte prosjektet nedskaleres og ha sitt hovedfokus på elektro og styresystem. Vi har fulgt kravspesifikasjonene og brukt prosjektperioden på å løse dem. Ønsket har vært å utvikle et styresystem som kan brukes dersom prosjektet videreføres. Selv om hovedfokuset har vært på styresystemet har vi måttet utfordre oss utenfor det som er vårt fagfelt og brukt en del tid på søyleløsninger og mekaniske utforminger. Noe av det vi ser på som viktigst ved en fortsettelse på prosjektet vil være å undersøke søylestrukturene og deres vekt begrensinger. Grunnen til at vi ikke har gjort dette er for å bruke tiden på det som er vår oppgave.

7.2 Utvidelse av systemet

For at en eventuell videreføring av prosjektet er det viktig at det kan utvides på en god måte. Da vi startet oppgaven var det som nevnt tidligere planen å bruke servomotorer til søylene, det ville ha ført til at hvert Arduino-kort kunne styre opp til ni søyler. Når vi byttet til trinnmotorer krevde de flere utganger, noe som har ført til at et Arduino-kort nå kan maks kontrollere fire søyler. Hvis man ser på søylekonstruksjonen, figur 26 og 27, ser man fire søyler per Arduino-kort. Vi referer til 5.2.5 om utvidelse metoden som vi har brukt. Slik vår løsning er nå er flere Arduinoer koblet til serveren ved hjelp av et enkelt spenningssignal. Dette signalet settes høyt eller lavt basert på om det er på tide å rotere søyler i systemet. I dette underkapittelet ønsker vi å diskutere andre løsninger på utvidelse av systemet.

Et alternativ er Arduino Mega [21]. Arduino Mega har 54 digitale inngang-/utganger og 16 analoge innganger/utganger. Det vil si at et Arduino Mega kort vil kunne ha 17 trinnmotorer koblet til. Det antallet kan bli litt mindre siden man vil trenge noen porter til en separat trådløs tilkobling, som ikke er innebygd. Man vil også trenge en port til å utvide videre til neste Arduino Mega, på samme måte som vi har utvidet i oppgaven vår. Arduino Mega ligger til noenlunde samme pris som Arduino Uno WiFi på Arduino sine hjemmesider. Dette alternativet vil være mer gunstig enn det som vi har brukt ettersom det er enklere å utvide med flere søyler uten å bruke like mange Arduinoer. Men det må legges til kostnadene til en ESP D1-mini ettersom Arduino Mega ikke har trådløs tilkobling. Grunnen til at vi ikke valgte denne løsningen fra starten av var at med servomotorer istedenfor trinnmotorer følte vi antallet inngang-/utganger på en Arduino Uno WiFi var tilstrekkelig. Arduino Uno WiFi hadde også innebygd trådløs kommunikasjon som vi ønsket.

Et annet alternativ på hvordan søylekonstruksjonene kunne blitt styrt er programmerbar logisk styring [30]. Bruk av PLS-er kan være veldig nyttig i systemer som har store fordelinger til mange motorer og sensorer. Programmerbar logisk styring (PLS) sin virkemåte er ofte at man har en stasjon som

inneholder moduler med mange inngang-/utganger, både digitale og analoge. De modulene er koblet videre til en prosessor som kan programmeres ved hjelp av leverandørens dataprogram. I dette prosjektet ville man kunne brukt PLS-er som fordelere til alle søylene, som igjen ble styrt av PLS-en sin prosessor og programvare. PLS-er er også svært stabile over lengre perioder, et typisk eksempel på noen som bruker PLS er samlebåndene hos produksjonsselskap som kjører gjennom flere steg ved hjelp av motorer og sensorer. Grunnen til at vi ikke ønsket å bruke PLS var blant annet server kommunikasjonen mellom de forskjellige enhetene i systemet. Alle PLS-programmer som vi har erfaring med fra jobb og studieløpet har vært koblet fysisk sammen uten noen trådløse signaler. Det var vurdert at en PLS ville kunne fått inn det trådløse signalet ved å ha PLS-en fysisk tilkoblet en ekstern trådløs kilde. Det ville da vært mulig å programmere styresystemet ved hjelp av PLS-programmet, samtidig som det snakket med serveren ved hjelp av f. eks ESP-er [6]. Konklusjonen ble at systemet ville bli det samme men ville koste mye mer ettersom man trengte PLS-systemet i tillegg. Logikken vi ønsket å lage ville også kunne ganske enkelt overføres til et PLS-program dersom det var ønskelig ved en senere anledning. Vi valgte derfor å utvikle logikken til Arduino og forklare den godt nok til at den kunne videreføres til f. eks PLS ved en senere anledning.

7.3 Presisjon

Siden roboten skal kunne komme seg frem i et system som egentlig ikke har så mye plass til overs, er vi avhengige av presisjon. Det er mange deler av systemet som kan ha variabler som ikke er så lette å kontrollere, men som man må forebygge best mulig.

Hver gang roboten når et av merkene i systemet som den skal stoppe ved, skjer ikke dette øyeblikkelig etter den infrarøde sensoren registrer det. Signalet skal først sendes til serveren og deretter sender serveren signal til roboten om at den skal stoppe. På den tiden det tar å gjøre dette har roboten flyttet seg videre. Vi er avhengige av at roboten faktisk stopper så presist i midten av søylene som mulig. Litt testing på dette problemet viste at avstanden som roboten flytter seg var ganske konstant under de samme betingelsene. Denne avstanden kunne vi bruke til å vite hvor store merkene i bakken måtte være. Avstanden fra kanten til merket og inn til sentrum gjorde vi like store som avstanden som roboten flyttet seg. Denne løsningen er ikke gunstig siden den er veldig avhengig av hvor raskt roboten kjører og hvor fort kommunikasjonen skjer. Vi har vurdert andre løsninger, men det er en begrensing på at det ikke alltid er roboten som skal stoppe i punktene den når.

Vi kan redusere tiden det tar før roboten får stoppsignal ved å stoppe roboten og kutte spenningen til hjulene direkte på roboten. Vi vil ikke at dette skal skje for hvert merke den når så da må det også legges til noe logikk på roboten som gjør at den vet at neste punkt er der den skal stoppe. Slik vi har det vet ikke roboten selv at den nærmer seg et stopp. Det er bare serveren som vet hvor langt roboten har kommet og roboten gjør bare det den blir fortalt om å gjøre. En løsning ville for eksempel være å ha et merke før hovedmerket som gir serveren en «heads-up» og deretter blir stoppet lokalt på roboten.

For å redusere avstanden roboten flytter seg etter at sensoren har mottatt signal kan vi redusere farten den kjører med når den nærmer seg. Etter den har passert det nest siste merket kan vi redusere farten på hjulene slik at den når målet i lav hastighet. Høyere fart fører til større avvik dersom kommunikasjonsfarten ikke er konstant.

En løsning kan også være å ha sensorer i hvert punkt som kan sjekke at roboten står i riktig posisjon. Styringen vet fra før hvor roboten må stoppe og kan ut fra det aktivere de sensorene det gjelder. Sensoren skal da være aktiv når roboten står riktig. Dersom den ikke gjør det må den heller korrigere posisjonen. En aktuell sensor vil kunne være en magnetoresistiv sensor [24] som registrerer magnetfelt. Den kan oppdage en magnet som kan henge under roboten. Denne løsningen vil kreve at man implementerer logikk som tar hensyn til om roboten enten har kjørt for langt eller for kort, og korrigerer posisjonen sin basert på det, men det vil da bli mer elektronikk og styringslogikk.

En annen ting som kan skape problem for presisjonen er om roboten ikke kjører bent. Den bør alltid kjøre rette linjer siden det ikke skal så mye til før klossen roboten frakter treffer noe. Trinnmotorene som driver hjulene kan bare kjøres en om gangen på grunn av at funksjonen i programmet er blokkerende. Dette påvirker ikke vanlig kjøring av roboten. Problemet kommer dersom spenningsforsyningen ikke er god nok til å drive begge motorene. Da kan det oppstå forskjeller som gjør at roboten ikke kjører bent lengre. Samme kan skje dersom det blir ulik slitasje på de to motorene. Dette er ting som kan forebygges med vedlikehold og regelmessig ladning av roboten.

Et lignende problem kan også oppstå når roboten skal rotere. I tillegg til at ulikheter fra hver motor er det også et problem med å nå en nøyaktig 90° vinkel. Trinnmotorene styres ved å si hvor mange trinn motoren skal ta. Antall trinn avhenger da av hvor store hjul som er på roboten og hvor langt det er mellom dem. I vår modell har vi nådd 90° etter litt prøving og finjustering.

En løsning på unøyaktig kjøring vil være å ha spor langs bakken som roboten kan kjøre i. På samme måte som en trakt kan de rette opp litt unøyaktig kjøring og få roboten på rett kjør. For denne løsningen må det monteres støttehjul fremme og bak på roboten som kan kjøre inn i disse sporene. Sporene må bare ligge midt i overgangen mellom to punkt og ikke ligge i kjørebanen til roboten når den ikke skal bruke de.

7.4 Løftemotor

En videreutvikling vil trenge en løftemotor som monteres på roboten. Den vil brukes til å levere og hente klosser i konstruksjonen. Vi valgte å ikke bruke tid på dette i vår løsning ettersom vi i samarbeid med veiledere følte det ble en stor nok utfordring å lage styresystemet. Prosjektet har allikevel tatt høyde for at det skal være en løfte-og-senke-funksjon på roboten. Kildekoden vil ikke trenge for store endringer til denne funksjonen. Der roboten i dag stopper i leveringspunktet må man legge inn en ny funksjon hos serveren som ber roboten løfte eller senke klossen. Denne løftesyklusen er forklart nærmere i kapittel 5.3.1.

7.5 Ladestasjon/Klosselager

Ettersom roboten i oppgaven er batteridrevet vil det være lurt å ha en fast ladestasjon når roboten står i ventemodus. Den beste plassen for dette vil være der klosselageret plasseres. En måte å gjøre dette på vil være å ha en kontakt som stikker ut i hente-og-leveringsposisjonen til roboten, denne kontakten vil da koble seg på roboten når den kommer i posisjon og lade batteriet. Det må da være et støpsel på roboten som passer i kontakten. Det burde også monteres en overvåking på roboten som gjør at den kjører til denne posisjonen når den begynner å bli tom for strøm. Da vil prosessen settes i

pause, derfor burde man bruke hurtigladende batteri og lader. Vi har valgt å ikke prioritere hvordan et klosselager må designes eller hvilken ladekontakt som må lages. Grunnen til det er igjen at vi ønsket å holde oss til styresystemet og kravspesifikasjonene. Når vi har laget logikk har vi likevel sett for oss at det finnes en posisjon der roboten vil hente og levere klosser.

7.6 Stabilitet

Det er viktig at prosjektet har et stabilt nettverk som er så robust som mulig, sånn at de forskjellige elementene ikke mister tilkobling underveis i en prosess. Det positive med nettverksløsningen til prosjektet er at nettverket ikke må være koblet til internett. Det at nettverket ikke er tilkoblet internett gjør at det er mye mindre utsatt for uforutsette hendelser. En annen faktor er strømbrudd, dette diskuteres nærmere i kapittel 7.7. Ved strømbrudd vil nå styringsprogrammet tro at roboten står i startposisjon, selv om den står en annen plass i systemet. Arduino Uno har ingen lagringsmetode ved et strømbrudd, som fører til denne feilen. En mulig løsning er å koble opp batterier som ligger klar ved et eventuelt strømbrudd. Da vil ikke systemet bli frakoblet ved et strømbrudd.

7.7 Sikkerhetstiltak

I denne oppgaven finnes det forskjellige sikkerhetstiltak som må vurderes. Den ene typen er at det uføres kalkulasjoner på kabeldimensjonene og strømfordelingen ved et ferdig system for å forsikre seg om at det ikke blir varmgang eller spenningstap i systemet. Varmgang oppstår når utstyr trekker mer strøm enn det er egnet for. Hvis man ikke regner ut kabeldimensjon for kablene i systemet kan det føre til stort spenningstap og i verste fall at noe blir så varm at det begynner å brenne. Det finnes formler som brukes for å sjekke dimensjonene på kabler [26] der man tar hensyn til hvor mye spenningstap man kan ha, hvilken spenning som brukes, hvor mye strøm systemet trekker og kabellengde. Da finner man ut hvilken dimensjon kabelen skal være. Vi har regnet på kablene og tatt høyde for at det er innenfor kravet, og vi er klar over de mest utfordrende punktene i systemet. Ved utvidelse til flere søyler og motorer vil det kreve nye beregninger.

En annen type sikkerhetstiltak oppgaven trenger er hva som gjøres ved feil i systemet, som at roboten ikke stopper opp eller andre ytre påvirkninger. For å løse dette kreves mye testing og at det legges til flere gjentagende funksjoner ved feil som oppstår, det fører da til at hvis noe ikke virker som det skal så har systemet en løsning på hva det skal gjøre istedenfor at alt stopper opp. Hvis man tenker seg at roboten får inn et signal om å kjøre fremover, for deretter at det oppstår nettverksproblemer som gjør at roboten ikke får signal om å stoppe. Så må det legges inn i koden til roboten at dersom roboten er frakoblet nettverket må den stoppe. Vi har prøvd å gjøre kode så robust som mulig etter kravspesifikasjonene til oppgaven var nådd. Noe vi har implementert er eksempelet som er nevnt over ved nettverksbrudd. Ved videre jobbing med prosjektet bør det være et mål at driften er forutsigbar og at systemet handler uforutsette hendelser bra.

8 Konklusjon

Ønsket fra oppdragsgiver var et «proof of concept» til et system. Kravene vi valgte å fokusere på var trådløs kommunikasjon mellom robot og styring, roterende søyler og robot navigasjon. Etter mange timer med utforming av logikk, testing ved hjelp av modell og feilsøking føler vi at målene i kravspesifikasjonene er nådd. Selv om det ikke er mangler ut fra kravene, ser vi flere positive og negative sider med prosjektet. De positive sidene er at styresystemets logikk virker som ønsket og kan videreføres, i tillegg er utformingen av søyler også en fungerende løsning. De negative sidene ved prosjektet er at selv ved relativ god presisjon vil det oppstå utfordringer og at vekten på overflaten kan påvirke systemets søyler negativt.

Det er også mange forskjellige måter å løse dette problemet på, og det å finne gode kombinasjoner med tanke på robusthet, pris og tilgjengelighet som krever mye testing. Vi har sikkert ikke det beste systemet, men vi har et system med enkelt design, billige og tilgjengelige komponenter og oversiktlig kode som virker, som er et utmerket utgangspunkt for videre optimalisering. Så til konklusjon ser prosjektet i sin helhet gjennomførbart ut, men en videreføring burde bruke enda flere sensorer og en mer presis leveringsmetode, gjerne skinner som også kan bruke vår logikk. Det burde også fokuseres på selve konstruksjonen og hvilken vekt denne er tiltenkt i et ferdig system.

Appendiks A Litteraturliste

[1] A. Mogaveera, R. Giri, M. Mahadik og A. Patil, «https://ieeexplore-ieee-org.galanga.hvl.no,» https://ieeexplore-ieee-org.galanga.hvl.no/document/8533870/. [Internett]. [Funnet 25 Januar 2019].

[2] S. Mandal, S. K. Saw, S. Maji, V. Das, S. K. Ramakuri og S. Kumar, «https://ieeexplore-ieee-org.galanga.hvl.no,» https://ieeexplore-ieee-org.galanga.hvl.no/document/7518916. [Internett]. [Funnet 25 Januar 2019].

[3] «Arduino Uno Wifi rev2,» https://store.arduino.cc/arduino-uno-wifi-rev2. [Internett]. [Funnet 28 1 2019].

[4] «Getting started with Arduino WiFi,» https://www.arduino.cc/en/Guide/ArduinoUnoWiFi?fbclid=IwAR35xi1Iv0ngrDVDg8y0ftc3_D-g9fr6oQ2Sb0e-UR8b39RAePNn3Xz5sPI. [Internett]. [Funnet 2019 1 28].

[5] «WiFiNINA,» https://www.arduino.cc/en/Reference/WiFiNINA. [Internett]. [Funnet 28 1 2019].

[6] «ESP informasjon,» https://arduino-esp8266.readthedocs.io/en/latest/. [Internett]. [Funnet 4 2 2019].

[7] «ESP Robot Eksempel 1,» https://www.youtube.com/watch?v=2AL7HfiRlp4. [Internett]. [Funnet 4 2 2019].

[8] «ESP Robot Eksempel 2,» https://www.youtube.com/watch?v=UsIB8lTm3Ik. [Internett]. [Funnet 4 2 2019].

[9] «ESP D1-mini,» https://www.banggood.com/no/WeMos-D1-mini-V2_2_0-WIFI-Internet-Development-Board-Based-ESP8266-4MB-FLASH-ESP-12S-Chip-p-1143874.html?gmcCountry=NO&currency=NOK&createTmp=1&utm_source=googleshopping&utm_medium=cpc_bgs&utm_content=xibei&utm_campaign=pla-mix. [Internett]. [Funnet 6 2 2019].

[10] «Servo eksempel 1,» https://www.ebay.co.uk/itm/192416274596?ul_noapp=true. [Internett]. [Funnet 28 1 2019].

[11] «Servo eksempel 2,» https://www.kjell.com/no/produkter/elektro-og-verktoy/elektronikk/rc-tilbehor/tilbehor-servo-og-mottaker/luxorparts-servo-360-5-5-kg-p90770. [Internett]. [Funnet 28 1 2019].

[12] «Forum Arduino,» https://forum.arduino.cc/index.php?topic=91508.0. [Internett]. [Funnet 10 2 2019].

[13] «Trinnmotor,» https://www.kjell.com/no/produkter/elektro-og-verktoy/utviklerkit/arduino/tilbehor/trinnmotor-med-girkasse-og-driver-p87062. [Internett]. [Funnet 15 3 2019].

[14] «IR Sensor,» https://www.banggood.com/5V-Infrared-Line-Track-Tracking-Tracker-Sensor-Module-For-Arduino-p-920304.html?cur_warehouse=CN. [Internett]. [Funnet 5 2 2019].

[15] D. Patel, E-post: dapa@norceresearch.no , dpa@hvl.no. [Internett].

[16] «Arduino vs Raspberry Pi,» https://www.educba.com/raspberry-pi-3-vs-arduino/. [Internett]. [Funnet 23 3 2019].

[17] «C++ forklaring,» https://en.wikipedia.org/wiki/C%2B%2B. [Internett]. [Funnet 21 4 2019].

[18] «DC-motor,» https://en.wikipedia.org/wiki/DC_motor. [Internett]. [Funnet 21 5 2019].

[19] «Arduino,» https://www.arduino.cc/. [Internett]. [Funnet 21 1 2019].

[20] «Arduino IDE,» https://www.arduino.cc/en/Main/Software. [Internett]. [Funnet 10 2 2019].

[21] «Arduino Mega,» https://store.arduino.cc/mega-2560-r3. [Internett]. [Funnet 22 12 2019].

[22] «Bilder er ikke beskyttet,» https://www.pexels.com/. [Internett]. [Funnet 8 5 2019].

[23] «Designprogram,» http://fritzing.org/home/. [Internett]. [Funnet 10 3 2019].

[24] «Digi-Key electronics Tech Forum,» https://forum.digikey.com/t/magnetoresistive-vs-hall-effect-in-sensor-applications/1185. [Internett]. [Funnet 20 5 2019].

[25] «Designprogram 2,» https://www.sketchup.com/. [Internett]. [Funnet 21 5 2019].

[26] «Kabeldimensjonering,» https://www.solar-wind.co.uk/info/dc-cable-sizing-tool. [Internett]. [Funnet 29 05 2019].

[27] «Microsoft Visio,» https://en.wikipedia.org/wiki/Microsoft_Visio. [Internett]. [Funnet 21 5 2019].

[28] «Raspberry Pi,» https://www.raspberrypi.org/. [Internett]. [Funnet 4 2 2019].

[29] «Stepper motor,» https://en.wikipedia.org/wiki/Stepper_motor. [Internett]. [Funnet 11 2 2019].

[30] «Wikipedia,» https://en.wikipedia.org/wiki/Programmable_logic_controller. [Internett]. [Funnet 24 5 2019].

[31] «Wikipedia,» https://en.wikipedia.org/wiki/Arduino_IDE. [Internett]. [Funnet 22 5 2019].

[32] O. Steine, «Selvlaget illustrasjon,» olav.henrikson.steine@gmail.com. [Internett].

[33] Wikipedia, «Mikrokontroller,» https://en.wikipedia.org/wiki/Microcontroller. [Internett]. [Funnet 20 1 2019].

[34] «Byggesett,» https://www.kjell.com/no/produkter/elektro-og-verktoy/utviklerkit/arduino/tilbehor/robotbyggesett-med-hjul-og-motor-p87065. [Internett].

[35] A. Raj, «CircuitDigest,» [Internett]. Available: https://circuitdigest.com/microcontroller-projects/arduino-stepper-motor-control-tutorial. [Funnet 26 5 2019].

Appendiks B Forkortelser og ordforklaringer

Arduino IDE	Arduino integrated development environment
C++	Et populært programmeringsspråk, som bygger på programmeringsspråket C.
DC	Direct Current
IP	Internet Protocol
Java	Et populært objektorientert programmeringsspråk.
Mikrokontroller	En programmerbar prosessor som samtidig inneholder periferifunksjoner
PLS	Programmerbar logisk styring
Spenningsomformer	Elektrisk utstyr som kan regulere spenningen den tar inn.
SSID	Service Set Identifier
USB	Universal Serial Bus

Appendiks C Prosjektledelse og styring

C.1 Gruppemedlemmer

- Henrik Lunde Lyssand
- Olav Henrikson Steine

C.2 Prosjektorganisasjon

Gruppen ønsker å samarbeide mest mulig med prosjektet. På grunn av prosjektets utfordringer er det mest effektivt for gjennomføringen å være to stykker på alle kravspesifikasjonene som skal utføres. Gruppen planlegger faste møtetider fortløpende for hver uke og holder seg til milepælsplanen for den overordnede oversikten. Hvis gruppen ønsker å jobbe individuelt med noen deler av prosjektet avtales det fordeling av hva som skal jobbes med.

C.3 Prosjektform

Prosjektmetoden kan beskrives best som eXtreme Programming (XP) med tanke på at gruppen har vært fleksible underveis. Ikke med tanke på kravspesifikasjonene siden de har vært de samme hele veien, men heller endring av løsning underveis. Dersom problem har oppstått, eller andre alternativer har vist seg å være bedre, har gruppen ikke hatt noen problem med å gå videre. Forstudiet gav oss en pekepinn på hva vi kunne forvente, men ting har endret seg etter det.

C.4 Fremdriftsplan

Initial fremdriftsplan:

					Uke	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18	w19	w20	w21	w22	w23	w24	w25	
					Man	21/1	28/1	4/2	11/2	18/2	25/2	4/3	11/3	18/3	25/3	1/4	8/4	15/4	22/4	29/4	6/5	13/5	20/5	27/5	3/6	10/6	17/6	
					Fre	25/1	1/2	8/2	15/2	22/2	1/3	8/3	15/3	22/3	29/3	5/4	12/4	19/4	26/4	3/5	10/5	17/5	24/5	31/5	7/6	14/6	21/6	
#	Aktivitet	Start Dato	Slutt Dato	Fram drift	Ansvarlig						SU?							Påske										
1	Første møte med veileder		18/1	100 %	alle																							
2	Arbeid med forstudie	18/1	31/1	100 %	Gruppen																							
3	Tilbakemelding forstudie	1/2	7/2	0 %	Institutt			◆																				
4	Bestille komponenter	1/2	15/2	0 %	Gruppen				◆																			
5	Trådløs kommunikasjon	1/2	20/2	0 %	Gruppen					◆																		
6	Styring av robot	20/2	20/3	0 %	Gruppen									◆														
7	Styring søyler	20/3	10/4	0 %	Gruppen												◆											
8	Midtveis presentasjon	20/3	1/4	0 %	Gruppen											◆												avtales med veileder
9	Bachelor oppgave arbeid	1/4	31/5	0 %	Gruppen																			◆				
10	Bachelor oppgave seminar		10/5	0 %	Gruppen																◆							10 min/gruppe
11	Bachelor oppgave innlevering		31/5	0 %	Gruppen																			◆				absolutt frist
12	Bachelor oppgave presentasjon	5/6	12/6	0 %	Gruppen																				◆			
13	EXPO / Avslutningsfest	13/6		0 %	Gruppen																					◆		alle må stille

Figur 40 – Initial fremdriftsplan

Endelig fremdriftsplan:

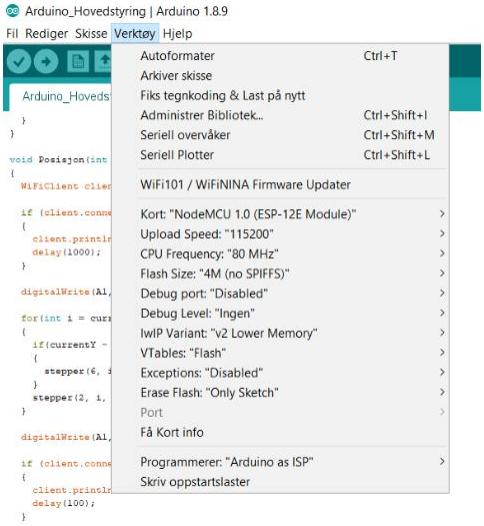
Figur 41 – Endelig fremdriftsplan [32]

Appendiks D Brukerdokumentasjon

En kort beskrivelse av hvordan man bruker den fysiske modellen til prosjektet skrives her.

Opplasting av kode til ESP:

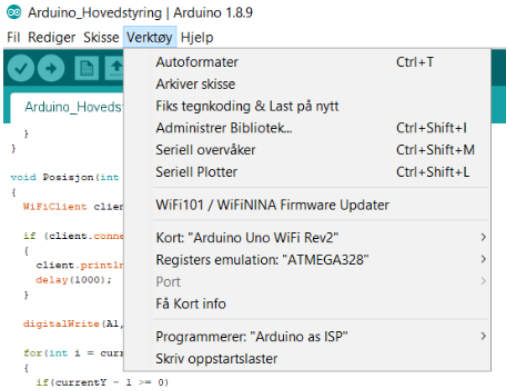
For å laste kode opp til ESP starter man førts opp Arduino sitt hovedprogram for kodeskriving. Dette programmet kan lastes ned på Arduino sine hjemmesider. Når man har ferdig skrevet kode og er klar til å laste opp til ESP-en er det viktig å velge de riktige innstillingene inne på programmet. Gå til verktøy og se deretter at «kort:» er satt til NodeMCU 1.0. Hvis man ikke har dette alternativet mangler man biblioteker på Arduino programmet og må laste det ned, et google søk på ESP8266 NodeMCU vil gi flere linker til nedlasting av biblioteket. Etter man har valgt NodeMCU 1.0 velger man «port:» og velger USB-porten ESP-en er koblet til. Man trykker deretter på last opp og koden bli lastet opp til mikrokontrolleren.



Figur 42 – Valg av kort til ESP.

Opplasting av kode til Arduino UNO WiFi Rev2:

For å laste kode opp til Arduino UNO WiFi Rev 2 starter man førts opp Arduino sitt hovedprogram for kodeskriving. Dette programmet kan lastes ned på Arduino sine hjemmesider. Når man har ferdig skrevet kode og er klar til å laste opp til Arduinoen er det viktig å velge de riktige innstillingene inne på programmet. Gå til verktøy og se deretter at «kort:» er satt til Arduino Uno WiFi Rev2. Hvis man ikke har dette alternativet mangler man biblioteker på Arduino programmet og må laste det ned, et google søk på Arduino Uno WiFi Rev2 library eller en oppdatering av Arduino-programmet kan løse dette. Etter man har valgt kort velger man «port:» og velger USB-porten Arduinoen er koblet til. Man trykker deretter på last opp og koden bli lastet opp til mikrokontrolleren.



Figur 43 – Valg av kort til Arduino.

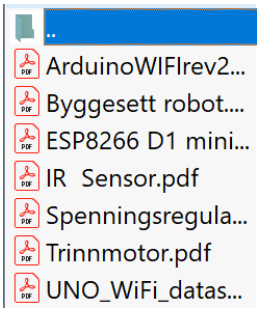
Viktige endringer som må gjøres før kjøring av kode:

Når man skal starte systemet er det et par ting man må sjekke at stemmer.

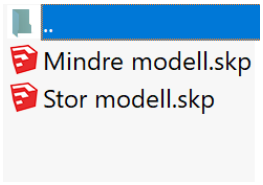
- Nytt nettverk vil og si ny SSID og passord. Dette må endres på alle komponenter som kobler seg opp mot nettverket. Ved oppkobling til nytt nettverk vil hver av komponentene også få tildelt nye IP-adresser. Serveren som sender kjørekommandoer til roboten, må få oppgitt riktig IP til den ESP-en som skal mota disse. Samme gjelder for ESP-en som sender sensordata.
- Dersom man endrer hvilke porter på mikrokontrollerene som brukes må dette også føres i koden. Alle porter som blir bruk må deklareres i starten av koden om man skal kunne bruke dem senere.
- Alle utvidelser må ha samme informasjon om systemet som server. Dette gjelder startposisjon, søylesystem og liste av destinasjoner. Dersom det ikke stemmer må korrigert kode lastes opp på hver mikrokontroller.

Appendiks E Vedlegg

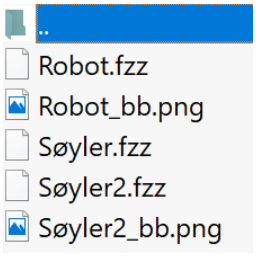
Datablader legges ved i en zip-fil.



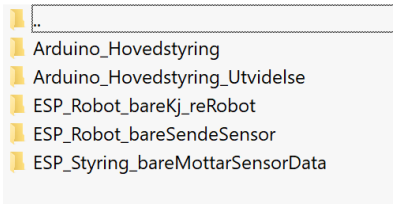
Modellene som er laget med programmet SketchUp legges ved i en zip-fil.



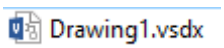
Modellene som er laget med Fritzing legges ved i en zip-fil.



Kildekode legges ved i en zip-fil.

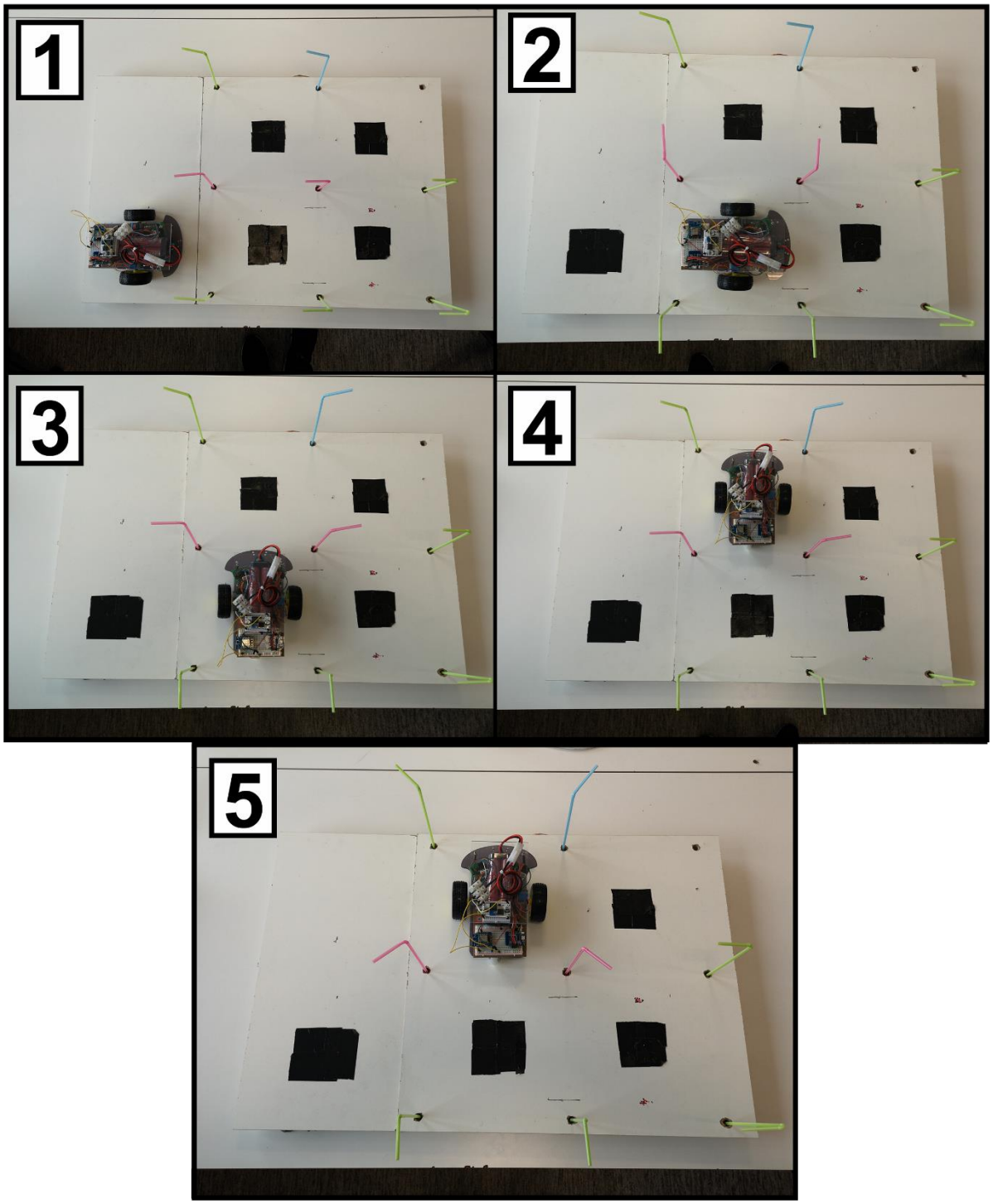


Flytskjema er lagt ved.



Figur, Kildekode og utstyrsliste

E.1 Figur av testkjøring av system



Figur 44 – Simulering av robot som kjører til leveringspunkt [32]. Se andre veien for å se hvordan roboten kjører tilbake.

E.2 Kildekode

Arduino_Hovedstyring 1

```
#include <WiFiNINA.h>
#include <Stepper.h>

char ssid[] = "NETGEAR30";
char pass[] = "freelotus929";

IPAddress server (192,168,1,2);

const int matriseX = 3;
const int matriseY = 3;

Stepper steppermotor[matriseX][matriseY] =
{
  {Stepper(0, 1, 1, 1, 1), Stepper(64, 12, 10, 11, 9), Stepper(64, 4, 2, 3, 0)},
  {Stepper(0, 1, 1, 1, 1), Stepper(64, A5, A3, A4, 13), Stepper(64, 8, 6, 7, 5)},
  {Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1)}
};

int posStepper[3][3] =
{
  {0, 0, 0},
  {0, 0, 0},
  {0, 0, 0}
};

bool kjor = true;

int destinasjonX = 0;
int destinasjonY = 0;
int startX = 0;
int startY = 1;
int currentX = startX;
int currentY = startY;
bool siste = LOW;

int steps = 0;
int stepsPerRunde = 2048;
int stepsPerSteg = stepsPerRunde / 8;

const int arrayX = 2;

int Array[arrayX][2] =
{
  {1,2},
  {2,2},
};

void setup()
{
  pinMode (A2, INPUT);
  pinMode (A1, OUTPUT);

  digitalWrite(A1, LOW);

  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
     delay(500);
  }
}

void loop()
{
  while (kjor == true)
  {
    for (int i = 0; i < arrayX; i++)
    {
      destinasjonX = Array[i][0];
      destinasjonY = Array[i][1];
      Posisjon(destinasjonX, destinasjonY);
    }
    kjor = false;

    digitalWrite(A1, HIGH);

    for(int i = 0; i < matriseX; i++)
    {
      for(int j = 0; j < matriseY; j++)
      {
        stepper(0, i, j);
      }
    }

    digitalWrite(A1, LOW);
  }
}
```

2

```
void Posisjon(int destinasjonX, int destinasjonY)
{
  WiFiClient client;

  if (client.connect(server, 80))
  {
    client.println("GET /stop");
    delay(1000);
  }

  digitalWrite(A1, HIGH);

  for(int i = currentX; i <= destinasjonX; i++)
  {
    if(currentY - 1 >= 0)
    {
      stepper(6, i, currentY - 1);
    }
    stepper(2, i, currentY);
  }

  digitalWrite(A1, LOW);

  if (client.connect(server, 80))
  {
    client.println("GET /forward");
    delay(100);
  }

  while (currentX < destinasjonX)
  {
    if (digitalRead(A2) == LOW && siste == HIGH)
    {
      currentX = currentX + 1;
    }
    siste = digitalRead(A2);
    delay(100);
  }

  if (currentY != destinasjonY)
  {
    if (client.connect(server, 80))
    {
      client.println("GET /stop");
      delay(1000);
    }

  if (destinasjonY > currentY)
  {
    if (client.connect(server, 80))
    {
      client.println("GET /left");
      delay(2000);
    }

    if (client.connect(server, 80))
    {
      client.println("GET /stop");
      delay(1000);
    }

    digitalWrite(A1, HIGH);

    for(int i = currentY; i <= destinasjonY; i++)
    {
      if(currentX - 1 >= 0)
      {
        stepper(0, currentX - 1, i);
      }
      stepper(4, currentX, i);
    }

    digitalWrite(A1, LOW);

    if (client.connect(server, 80))
    {
      client.println("GET /forward");
      delay(100);
    }

    while (currentY < destinasjonY)
    {
      if (digitalRead(A2) == LOW && siste == HIGH)
      {
        currentY = currentY + 1;
      }
      siste = digitalRead(A2);
      delay(100);
    }
  }
}
```

3

```
else
{
  if (client.connect(server, 80))
  {
    client.println("GET /right");
    delay(2000);
  }

  if (client.connect(server, 80))
  {
    client.println("GET /stop");
    delay(1000);
  }

  digitalWrite(A1, HIGH);

  for(int i = currentY; i >= destinasjonY; i--)
  {
    if(currentX - 1 >= 0)
    {
      stepper(0, currentX - 1, i);
    }
    stepper(4, currentX, i);
  }

  digitalWrite(A1, LOW);

  if (client.connect(server, 80))
  {
    client.println("GET /forward");
    delay(100);
  }

  while (currentY > destinasjonY)
  {
    if (digitalRead(A2) == LOW && siste == HIGH)
    {
      currentY = currentY - 1;
    }
    siste = digitalRead(A2);
    delay(100);
  }
}
}

if (client.connect(server, 80))
{
  client.println("GET /stop");
  delay(5000);
}

digitalWrite(A1, HIGH);

if(destinasjonX - 1 >= 0 && destinasjonY - 1 >= 0)
{
  stepper(7, destinasjonX - 1, destinasjonY - 1);
}
if(destinasjonY - 1 >= 0)
{
  stepper(5, destinasjonX, destinasjonY - 1);
}
if(destinasjonX - 1 >= 0)
{
  stepper(1, destinasjonX - 1, destinasjonY);
}
stepper(3, destinasjonX, destinasjonY);

digitalWrite(A1, LOW);

if (client.connect(server, 80))
{
  client.println("GET /stop");
  delay(5000);
}

if (currentY != startY)
{
  digitalWrite(A1, HIGH);

  if (currentY > startY)
  {
    for(int i = currentY; i >= startY; i--)
    {
      if(currentX - 1 >= 0)
      {
        stepper(0, currentX - 1, i);
      }
      stepper(4, currentX, i);
    }

    digitalWrite(A1, LOW);

    if (client.connect(server, 80))
    {
      client.println("GET /backward");
      delay(100);
    }

    while (currentY > startY)
    {
      if (digitalRead(A2) == LOW && siste == HIGH)
      {
        currentY = currentY - 1;
      }
      siste = digitalRead(A2);
      delay(100);
    }

    if (client.connect(server, 80))
    {
      client.println("GET /stop");
      delay(1000);
    }

    if (client.connect(server, 80))
    {
      client.println("GET /right");
      delay(2000);
    }
  }
  else
  {
    for(int i = currentY; i <= startY; i++)
    {
      if(currentX - 1 >= 0)
      {
        stepper(0, currentX - 1, i);
      }
      stepper(4, currentX, i);
    }

    digitalWrite(A1, LOW);

    if (client.connect(server, 80))
    {
      client.println("GET /backward");
      delay(100);
    }

    while (currentY < startY)
    {
      if (digitalRead(A2) == LOW && siste == HIGH)
      {
        currentY = currentY + 1;
      }
      siste = digitalRead(A2);
      delay(100);
    }

    if (client.connect(server, 80))
    {
      client.println("GET /stop");
      delay(1000);
    }

    if (client.connect(server, 80))
    {
      client.println("GET /left");
      delay(2000);
    }
  }

  if (client.connect(server, 80))
  {
    client.println("GET /stop");
    delay(1000);
  }
}

digitalWrite(A1, HIGH);

for(int i = currentX; i > startX; i--)
{
  if(currentY - 1 >= 0)
  {
    stepper(6, i, currentY - 1);
  }
  stepper(2, i, currentY);
}

digitalWrite(A1, LOW);

if (client.connect(server, 80))
{
  client.println("GET /backward");
  delay(100);
}

while (currentX > startX)
{
  if (digitalRead(A2) == LOW && siste == HIGH)
  {
    currentX = currentX - 1;
```

4

5

```
    }
    siste = digitalRead(A2);
    delay(100);
  }

  if (client.connect(server, 80))
  {
    client.println("GET /stop");
    delay(5000);
  }

  client.flush();
  client.stop();
}

void stepper(int nypos, int x, int y)
{
  steppermotor[x][y].setSpeed(400);

  if(nypos - posStepper[x][y] >= 0 && nypos - posStepper[x][y] <= 4)
  {
    steps = ((nypos - posStepper[x][y]) * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }
  else if(nypos - posStepper[x][y] >= 0)
  {
    steps = ((8 - (nypos - posStepper[x][y])) * -1 * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }
  else if(nypos - posStepper[x][y] < 0 && nypos - posStepper[x][y] >= -4)
  {
    steps = ((nypos - posStepper[x][y]) * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }
  else
  {
    steps = ((8 + (nypos - posStepper[x][y])) * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }

  posStepper[x][y] = nypos;
}
```

Arduino_Utvidelse

1

```
#include <Stepper.h>

const int matriseX = 3;
const int matriseY = 3;

Stepper steppermotor[matriseX][matriseY] =
{
  {Stepper(64, 8, 6, 7, 5), Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1)},
  {Stepper(64, 12, 10, 11, 9), Stepper(0, 1, 1, 1, 1), Stepper(0, 1, 1, 1, 1)},
  {Stepper(64, A0, A2, A1, A3), Stepper(64, 0, 3, 2, 4), Stepper(0, 1, 1, 1, 1)}
};

int posStepper[matriseX][matriseY] =
{
  {0, 0, 0},
  {0, 0, 0},
  {0, 0, 0}
};

bool kjor = true;

int destinasjonX = 0;
int destinasjonY = 0;
int startX = 0;
int startY = 1;
int currentX = startX;
int currentY = startY;
bool siste = LOW;

int steps = 0;
int stepsPerRunde = 2048;
int stepsPerSteg = stepsPerRunde / 8;

const int arrayX = 2;

int Array[arrayX][2] =
{
  {1,2},
  {2,2}
};

void setup()
{
  pinMode (A5, INPUT);
}

void loop()
{
  while (kjor == true)
  {
    for (int i = 0; i < arrayX; i++)
    {
      destinasjonX = Array[i][0];
      destinasjonY = Array[i][1];
      Posisjon(destinasjonX, destinasjonY);
    }
    kjor = false;

    while(digitalRead(A5) == LOW)
    {
      delay(10);
    }

    for(int i = 0; i < matriseX; i++)
    {
      for(int j = 0; j < matriseY; j++)
      {
        stepper(0, i, j);
      }
    }
  }
}

void Posisjon(int destinasjonX, int destinasjonY)
{
  while(digitalRead(A5) == LOW)
  {
    delay(10);
  }

  for(int i = currentX; i <= destinasjonX; i++)
  {
    if(currentY - 1 >= 0)
    {
      stepper(6, i, currentY - 1);
    }
    stepper(2, i, currentY);
  }

  while(digitalRead(A5) == HIGH)
  {
    delay(10);
  }

  currentX = destinasjonX;

  if (currentY != destinasjonY)
  {
```

2

```
  {
    for (int i = 0; i < arrayX; i++)
    {
      destinasjonX = Array[i][0];
      destinasjonY = Array[i][1];
      Posisjon(destinasjonX, destinasjonY);
    }
    kjor = false;

    while(digitalRead(A5) == LOW)
    {
      delay(10);
    }

    for(int i = 0; i < matriseX; i++)
    {
      for(int j = 0; j < matriseY; j++)
      {
        stepper(0, i, j);
      }
    }
  }
}

void Posisjon(int destinasjonX, int destinasjonY)
{
  while(digitalRead(A5) == LOW)
  {
    delay(10);
  }

  for(int i = currentX; i <= destinasjonX; i++)
  {
    if(currentY - 1 >= 0)
    {
      stepper(6, i, currentY - 1);
    }
    stepper(2, i, currentY);
  }

  while(digitalRead(A5) == HIGH)
  {
    delay(10);
  }

  currentX = destinasjonX;

  if (currentY != destinasjonY)
  {
    stepper(1, destinasjonX - 1, destinasjonY);
  }
  stepper(3, destinasjonX, destinasjonY);

  while(digitalRead(A5) == HIGH)
  {
    delay(10);
  }

  if (currentY != startY)
  {
    while(digitalRead(A5) == LOW)
    {
      delay(10);
    }

    if (currentY > startY)
    {
      for(int i = currentY; i >= startY; i--)
      {
        if(currentX - 1 >= 0)
        {
          stepper(0, currentX - 1, i);
        }
        stepper(4, currentX, i);
      }
    }
    else
    {
      for(int i = currentY; i <= startY; i++)
      {
        if(currentX - 1 >= 0)
        {
          stepper(0, currentX - 1, i);
        }
        stepper(4, currentX, i);
      }
    }

    while(digitalRead(A5) == HIGH)
    {
      delay(10);
    }

    currentY = startY;
  }

  while(digitalRead(A5) == LOW)
```

3

```
  {
    delay(10);
  }

  for(int i = currentX; i > startX; i--)
  {
    if(currentY - 1 >= 0)
    {
      stepper(6, i, currentY - 1);
    }
    stepper(2, i, currentY);
  }

  while(digitalRead(A5) == HIGH)
  {
    delay(10);
  }

  currentX = startX;
}

void stepper(int nypos, int x, int y)
{
  steppermotor[x][y].setSpeed(400);

  if(nypos - posStepper[x][y] >= 0 && nypos - posStepper[x][y] <= 4)
  {
    steps = ((nypos - posStepper[x][y]) * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }
  else if(nypos - posStepper[x][y] >= 0)
  {
    steps = ((8 - (nypos - posStepper[x][y])) * -1 * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }
  else if(nypos - posStepper[x][y] < 0 && nypos - posStepper[x][y] >= -4)
  {
    steps = ((nypos - posStepper[x][y]) * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }
  else
  {
    steps = ((8 + (nypos - posStepper[x][y])) * stepsPerSteg);
    steppermotor[x][y].step(steps);
  }

  posStepper[x][y] = nypos;
}
```

ESP_ROBOT_KJØRING 1

```
#include <ESP8266WiFi.h>
#include <Stepper.h>

Stepper steppermotorV(64, 2, 4, 0, 5);
Stepper steppermotorH(64, 16, 12, 13, 14);

int steps = 2048;
int stepsPerStegKjor = steps/32;
int stepsPerStegSving = steps/2;

WiFiClient client;
WiFiServer server(80);

const char* ssid = "NETGEAR30";
const char* pass = "freelotus929";

String  data ="";

void setup()
{
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
     delay(500);
  }

  server.begin();
}

void loop()
{
  client = server.available();
  if (WiFi.status() != WL_CONNECTED)
  {
    data = "stop";
  }
  else if (client)
  {
    data = checkClient ();
  }
  else if (!client && (data != "forward" && data != "backward"))
  {
    return;
  }

  steppermotorV.setSpeed(400);
  steppermotorH.setSpeed(400);

  if (data == "forward") MotorForward();
  else if (data == "backward") MotorBackward();
  else if (data == "left") TurnLeft();
  else if (data == "right") TurnRight();
  else if (data == "stop") MotorStop();
}

void MotorForward(void)
{
  for(int i = 0; i < stepsPerStegKjor; i++)
  {
    steppermotorV.step(1);
    steppermotorH.step(-1);
  }
}
```

2

```
void MotorBackward(void)
{
  for(int i = 0; i < stepsPerStegKjor; i++)
  {
    steppermotorV.step(-1);
    steppermotorH.step(1);
  }
}

void TurnLeft(void)
{
  for(int i = 0; i < stepsPerStegSving; i++)
  {
    delay(1);
    steppermotorV.step(1);
    steppermotorH.step(1);
  }
}

void TurnRight(void)
{
  for(int i = 0; i < stepsPerStegSving; i++)
  {
    delay(1);
    steppermotorV.step(-1);
    steppermotorH.step(-1);
  }
}

void MotorStop(void){}

String checkClient (void)
{
  while(!client.available()) delay(1);
  String request = client.readStringUntil('\r');
  request.remove(0, 5);
  request.remove(request.length()-9,9);
  return request;
}
```

ESP_SENDE_SENSOR 1

2

```
#include <ESP8266WiFi.h>

char ssid[] = "NETGEAR30";
char pass[] = "freelotus929";

IPAddress server(192,168,1,7);

void setup() {
  pinMode (12, INPUT);

  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
     delay(500);
  }
}

void loop()
{
  WiFiClient client;

  int sensorReading = digitalRead (12);
  if (client.connect(server, 80))
  {
    if(sensorReading == 1)
    {
      client.println("GET /one");
    }
    else
    {
      client.println("GET /zero");
    }
  }
  delay(10);

  client.flush();
  client.stop();
}
```

ESP_MOTTA_SENSOR 1

```
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);

const char* ssid = "NETGEAR30";
const char* password = "freelotus929";

String  data ="";

int sensorData = 2;

void setup()
{
  pinMode(sensorData, OUTPUT);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
     delay(500);
  }

  server.begin();
}

void loop()
{
    client = server.available();
    if (!client) return;
    data = checkClient ();

    if (data == "one") SensorHigh();
    else if (data == "zero") SensorLow();
}

void SensorHigh(void)
{
  delay(1);
  digitalWrite(sensorData,HIGH);
}

void SensorLow(void)
{
  delay(1);
  digitalWrite(sensorData,LOW);
}


String checkClient (void)
{
  while(!client.available()) delay(1);
  String request = client.readStringUntil('\r');
  request.remove(0, 5);
  request.remove(request.length()-9,9);
  return request;
}
```

2

E.3 Utstyrsliste

Navn	Merke	Prod. nummer
2WD MOTOR CHASSIS ROBOTICS KIT	Velleman	VMA500
ESP8266 D1 mini	Wemos	
Line Tracking Sensor	DFRobot	RB-Dfr-40
LM2596S	Vellemann	VMA404
5 VDC STEPPER MOTOR	Vellemann	VMA401
Arduino UNO WiFi rev2	Arduino	ABX00021
Fordelingskabler	Luxorparts	Art. 90793