University College Nordjylland

4. Semester Projekt

UvU-Bot Projekt

*Forfatter:*

Dennis Rasmussen

*Vejledere:*

Mogens Holm Iversen

Ib Helmer Nielsen

Antal anslag:  
xxxxx

17. december 2018

**University College Nordjylland  
Teknologi og Business**Datamatiker  
Dmaa0217  
UVU-Bot projekt

**Projektdeltager:**Dennis Rasmussen

**Vejledere:**Ib Helmer Nielsen  
Mogens Holm Iversen

**Afleveringsdato:**17. december 2018

**Repository placering:**<https://github.com/Derasm/4Semester>

# Abstract:

Skabelsen af robotter er gået fra at være en utroligt krævende affære kun mulig for virksomheder, til at blive noget der er muligt på hobbyniveau. Med et hav af muligheder indenfor valg af computer, programmeringssprog og komponenter kan det til gengæld blive svært at vælge. Rapporten her undersøger to af de mest populære computere til robotter og hvordan sammenkoblingen mellem hardware og software kan udføres.  
Problemformuleringen for rapporten lægger grundlag for spørgsmålene om design og implementation af robotten, og hvordan en arkitektur kan opbygges så den bliver så robust og nem at vedligeholde og udvide, som muligt.   
Rapporten undersøger, beskriver og løser problemformuleringen ved at kigge på mulighederne indenfor to af de microcontrollers der eksisterer, med simple udbygning- og tilkoblingsmuligheder, arkitekturen som skal understøtte softwaren og algoritmen der vil ligge til grund for ruteplanlægningen i systemet. Som noget af det sidste består rapporten af en perspektiveringssektion hvor eventuelle problemer og udfordringer bliver gennemgået og diskuteret. Konklusionen viser at Arduino var den mest passende microcontroller til styring, en modificeret tre-lags arkitektur var den mest hensigtsmæssige arkitektur at bruge til at sikre lav kobling og høj samhørighed, samt at pathing-algoritmen A\* gav de bedste resultater til en autonomisk ruteplanlægning.

Til den fremtidige perspektivering i udviklingen af robotter kunne det undersøges hvordan skabelsen af robotter kunne gøres mere modulært og tilgængeligt udenom programmering, så det ikke bliver kodning og den abstrakte forståelse der sættes grænser men derimod fantasien og brugen heraf.

# Forord:

Denne rapport er skrevet af Dennis Bundgaard Rasmussen til 4. semester på Datamatiker uddannelsen på University College Nordjylland (UCN). Projektet kunne ikke have været muligt uden hjælp fra Ib Helmer Nielsen og Steffen Vutborg fra It-teknolog uddannelsen på UCN. Disse to herrer var en uendelig hjælp til opsætning og forståelsen af hardwaren, hvem foruden dette projekt havde været markant sværere, grænsende til umuligt.   
Dernæst skal der rettes speciel tak til Brian Hvarregård og Mogens Holm Iversen for besvarelse af spørgsmål og hjælp til direktion for projektet.

Indholdsfortegnelse

[Abstract: 3](#_Toc532746705)

[Forord: 4](#_Toc532746706)

[Indledning 6](#_Toc532746707)

[Problemområde 7](#_Toc532746708)

[Problemfelt 7](#_Toc532746709)

[Problemformulering 7](#_Toc532746710)

[Teknologi 8](#_Toc532746711)

[C++ 8](#_Toc532746712)

[Microcontroller 10](#_Toc532746713)

[Arduino Uno 15](#_Toc532746714)

[Styring af motorerne. 16](#_Toc532746715)

[Lokalisering og simulering af robottens placering i et miljø. 18](#_Toc532746716)

[Pathing 19](#_Toc532746717)

[Styring af robotten. 21](#_Toc532746718)

[Delkonklusion 22](#_Toc532746719)

[Design 23](#_Toc532746720)

[Usecases 24](#_Toc532746721)

[Arkitektur 26](#_Toc532746722)

[Delkonklusion; 27](#_Toc532746723)

[Implementering 27](#_Toc532746724)

[Motor 27](#_Toc532746725)

[Pathing 29](#_Toc532746726)

[Refleksion 30](#_Toc532746727)

[Undervisningsmateriale 30](#_Toc532746728)

[Fremgangsmåde for projektet: 33](#_Toc532746729)

[Konklusion 35](#_Toc532746730)

[Litteraturliste 36](#_Toc532746731)

[Bilag 38](#_Toc532746732)

# Indledning

Der er få ting i verdenen der fremstår som et bedre eksempel på menneskets nysgerrighed end robotter. Deres funktioner er ikke altid nyttige ej heller er de nødvendige - ofte stammer deres skabelse fra attituden ”gad vide” frem for spørgsmålet ”hvordan løser jeg”. Denne attitude har ført til skabelsen af robotter hvis eneste formål er at hælde sukker i en kop eller sprøjte maling på et lærred.  
I den modsatte ende har det også ført til selvkørende biler, droner og automatisk juridisk rådgivning.

Denne vilje til at undersøge mulighederne for hvad man kan få en maskine til at gøre, ligger til grunde for skabelsen af ufatteligt mange interessante ting. Kulturen omkring hjemmebyggede robotter vokser dag for dag, med ressourcer og hjælp til alt fra at få en diode til at blinke til at kunne køre gamle Nintendo spil. Denne nysgerrighed for skabelsen af robotter har ført til det overgående spørgsmål for dette projekt:

”Hvordan kan man lave en robot der automatisk kan køre fra et punkt til et andet til fragtning af pakker?”

Spørgsmålet blev grebet an ved at analysere microcontrollerne der styrer robotten, den underliggende teknologi der understøtter den samt softwaren der kontrollerer den. Dernæst blev softwaren implementeret og testet for at sikre et produkt der besvarede spørgsmålet.

Af hensyn til opgavens omfang behandler rapporten ikke valg af programmeringssprog, men gennemgår i stedet funktionaliteten i det for udvalgte sprog.

# Problemområde

## Problemfelt

## Problemformulering

Hvordan kan man, ved hjælp af en selvkørende robot, transportere små pakker mellem to lokationer. Herunder med særligt fokus på løsning af følgende:

* hvilken platform skal man vælge for at udvikle en prototype?
* hvordan kan en robust softwarearkitektur designes og implementeres til en selvkørende robot?
* er det hensigtsmæssigt at robotten er autonomisk, eller skal den kontrolleres fra en ekstern enhed (fx en mobiltelefon)?

## Teknologi

Teknologiafsnittet af rapporten vil komme ind på programmeringssproget og hardware valg til kontrol af servoerne. Herunder vil der forekomme en kort beskrivelse af opsætningen af Arduinoens environment. Udover dette vil der i afsnittet blive gennemgået pathing i projektet, lokaliseringssystemet og tankerne bag de individuelle valg. Til sidst vil en delkonklusion følge, der opsummerer afsnittet og de underspørgsmål afsnittet besvarer.

### C++

Der vil i dette afsnit forekomme en gennemgang af C++ som programmeringssprog med begrundelse for valget samt særlig fokus på compilerprocessen, da denne har direkte indflydelse på hvordan koden bliver kompileret og forstået af microcontrolleren.

C++ blev valgt da det ligger meget tæt på hardware laget, og der dermed ikke er meget abstraktion imellem programmerede kode og maskinsprog, hvilket gør det til et meget hurtigt sprog. Udover dette er Arduinos eget programmeringssprog en simplificeret version af C++ med nogle begrænsninger, hvilket betyder C++ koncepter er velkendte i Arduino udviklingsprocessen og det er nemmere at finde ressourcer online til at løse eventuelle problemer der dukker op.

C++ er et objekt orienteret sprog (OOP) bygget ovenpå C af Bjarne Stroustrup (Stroustrup, u.d.) til at være et generelt programmeringssprog. C++ udskiller sig fra andre OOP sprog ved dets adskillelse af interface og implementering som standard kodeskik. Interfaces bliver defineret i header filer, mens implementering bliver defineret i cpp filer. Dette er gjort af samme grund som C gjorde det; nyere sprog bruger ikke fremadrettet deklarationer, hvilket gør headerfiler unødvendige i de nye sprog, og nødvendige i C og C++ da compileren skal vide hvad den kan forvente at der ligger i implementationsfilen.

Nye sprog gør brug af identifiers som er genkendte automatisk fra kildefiler og læst af dynamiske libraries. Dette gør header filerne unødvendige (Wikipedia, u.d.)

Det er standard naming convention der har valgt navnene header og cpp, reelt set kunne de hedde alt, så længe man fortæller compileren hvad filer den skal lede efter. Dette gør C++ anderledes end andre sprog; det er open source, og udviklingen af sproget bliver gjort i samarbejde med et råd der tager imod inputs som bliver stemt på af brugerne af C++. Dette betyder at man kan ændre compileren og sproget så meget man vil, til at udføre de ting man selv har behov for.

Der er en del forskellige open source-compilers der er meget brugt, herunder GCC, GNU og installationspakken MinGW med compilers til bland andet C, C++ og Fortran til brug af Windows. Compilerne adskilles efter hvilke dele af C++ sproget de har implementeret og styresystemet de fungerer bedst på. C++ er lige pt. I C++17, hvilket beskriver udgivelsesåret af versionen, og er den gældende version indtil C++20 kommer i 2020. Det er ikke alle funktionaliteter fra C++17 der er implementeret i de forskellige compilers, hvilket gør sproget anderledes fra closed source sprog, hvor sproget og compilerens udvikling følges ad.

Da C++ er et open source-sprog har sproget også diverse kodestandards og en komité af folk der tager imod ændringer og vælger hvad der skal inkluderes i sproget. Det er her at headers og cpp filer blev valgt for at adskille interface og implementation. Dette kan følges på den officielle C++ side[[1]](#footnote-1), hvor man også kan læse om standard konventionerne.

C++ udmærker sig til generel brug da det ligger lavt på stacken af abstraktioner over maskinsprog og ikke skal igennem virtuelle maskiner for at være læsbart for pc’en.  
Dette gøres i stedet igennem en compiler som fungerer ved et 3-skrids system i form af Pre-processing, Compilation og Linking. (Gregory, n.d.)

* Pre-processing tager source code filerne og håndtere alle pre-processor kommandoer som er annoteret med #. Outputtet af dette er en C++ fil uden pre-processor direktiver. Dette betyder blandt andet at pre-processoren tager indholdet af de filer der bliver included og kopierer dem ind i filerne de er henvist fra.
* Compileren tager pre-processorerens output og laver denne fil om til Object Files med varierende former for optimering af hastighed og læsbarhed, alt efter hvilken indstilling man har sat på compileren. Her får hver metode en unik signatur som bliver brugt af linkeren til at linke de forskellige .obj filer sammen. Hvis man har aktiveret debugging funktionen i compileren vil man få en større fil med mere tekst, da det bliver nemmere at debugge og finde den specifikke fejl end hvis det blot er lavet til minimale instruktioner for maskinen.
* Linkeren tager object filerne som Compileren lavede og producerer en executable fil. I dette stadie bliver alle objektfilerne linket sammen ved at referencer bliver sammenlagt med de rigtige adresser og signaturer bliver læst fra metoder hvor referencer dertil bliver lavet. Det hele bliver lagt i en enkelt fil som kan blive kørt.

De executables der bliver produceret af generelle compilere i C++ er anderledes fra dem der bliver produceret af Arduinoen. Dette skyldes at C++ har en specifik .ino fil som deklarerer sketches, et koncept der bliver brugt når der skrives programmer ved brug af Arduinos egen IDE.

Det er muligt at kompilere og lave programmer der kan køres på Arduinoen uden at bruge Arduinocompileren fra deres IDE. Dette gøres ved at man laver en Makefile, som er en fil med shell-kommandoer til at beskrive hvordan filerne skal compileres og sættes sammen. Dette er specielt nyttigt hvis man har tænkt sig at bruge nogle funktioner der endnu ikke er understøttet af Arduino compileren, såsom C++17 funktionalitet og lignende, da de sagtens kan blive kørt alligevel, da det hele bliver lavet om til machinecode som Arduinoen kører uden problemer.

### Microcontroller

Styringen af robotten bliver gjort af en computer kaldet en microcontroller. Denne består oftest af en simpel opbygning af RAM og CPU og nogle pins til kobling af andet hardware, dog kan den også være opbygget som en normal PC blot i mindre størrelse. I rapporten er der valgt at kigge på de to største udbydere af microcontrollers på markedet, navnlig Arduino Uno skabt af Arduino virksomheden og Raspberry Pi 3B+ skabt af Raspberry Foundation. Der er valgt at kigge på sværhedsgrad af brug, programmeringssprog, styresystem, hastighed og opkoblingsmuligheder med anden hardware.

Sværhedsgraden er valgt at blive defineret som værende en skala fra 1 til 5, hvor 1 er det nemmeste og 5 er det sværeste. Dette er gjort for at holde det simpelt at sammenligne, på trods af at sværhedsgrad er en subjektiv holdning baseret på foregående viden.

|  |  |  |
| --- | --- | --- |
|  | Arduino Uno | Raspberry Pi 3B+ |
| Sværhedsgrad af brug: | 1/5. Simpel upload, intet OS, nemt at bruge pins. | 3/5. Linux OS, svært at bruge onboard pins. |
| Operativsystem | intet | Raspbian Linux. |
| Programmeringssprog | C++ og Python, visuelle sprog og C#. | C++ og Python, JVM sprog. |
| opkoblingsmuligheder | Pins til hardware, usb kabel. | Usb kabel, hdmi, ethernet, SD card, SSH / FTP. |
| Hastighed: | 16 MHz. [[2]](#footnote-2) | 1,2 GHz. |
| Hardware: | CPU / Ram. | CPU, Ram, Hd, Ethernet, HDMI, Audio jack.[[3]](#footnote-3) Onboard GPU. |
| Voltage: | 5V, nem at køre med batterier. | 5V power supply, mere alt efter hvad der er sat til. |
| Understøttende libraries | Nemt C++ og Python library lavet af Arduino selv. | Nemt Python library, lidt mindre nemt C++ library |

Som man kan se på diagrammet er der fordele og ulemper ved begge microcontroller.

**Arduino Uno**

|  |  |
| --- | --- |
| Fordele | Ulemper |
| Intet operativsystem | Langsommere end Raspberry |
| Nemmere pins | Kun usb og strømkabel påsat fra start |
| Skabt til c++ | Kræver elektronikviden |
| Nemt library til pins og strøm | Intet networking uden ekstra modul |
| Længere batteritid på 4 aa batterier | Ingen harddisk |
| Nem overførsel og brug af software |  |

**Raspberry Pi 3B+:**

|  |  |
| --- | --- |
| Fordele | Ulemper |
| Hastighed | Operativsystem, ikke plug’n use. |
| Mere komplet pc | Linux OS og linux kommandoer |
| Opkoblingsmuligheder foruden pins | Højere strømkrav, lavere batteritid |
| Kræver ikke elektronikviden | Sværere opkobling af pins |
| God til internetbaserede løsninger | Mere kompleks |
| Harddisk | Bedst egnet til Python |
|  | Svært library til C++ |

Som man kan se, er begge microcontrollers med deres fordele og ulemper fordelagtige alt efter projektbehov. Raspberrien egner sig bedst til projekter der skal bruge det underliggende operativsystem, eller har brug for netværksopkobling til multitasking. Arduinoen egner sig bedst til prototyping, ikke har behov for internet eller skal køre på batteri i længere perioder. Da projektet handler om at finde den bedste microcontroller til at løse problemformuleringen, specielt med fokus på at robotten er en prototype, var der nogle punkter der var meget vigtige for valget.

Raspberriens underliggende operativsystem talte meget imod brugen af denne, da det betød højere kompleksitet. Manglen på internet uden opkobling var en stærk ulempe ved Arduinoen da det gjorde videreudvikling mere besværligt. Da en beslutning ikke kunne blive taget på baggrund af fordelene og ulemperne, blev der fastsat nogle krav robotten skal opfylde for at kunne besvare problemformuleringen.

Kompleksitetskrav

* Den skal kunne køre ved hjælp af servoer.
* Den behøver ikke en skærm.
* Den skal være batteridreven.
* Den skal kunne videreudvikles til trådløs kommunikation med andre enheder.
* Den skal være simpel at opsætte og programmere.
* Den behøver ikke harddisk.
* Den behøver ikke netværksopsætning.
* Den skal ikke multitaske, og kræver derfor ikke en hurtig processor.

Baseret på kompleksitetskravene, diagrammet over Raspberry og Unoen samt listen over fordele og ulemper, blev der vurderer at Arduino Uno var den bedste kandidat.

Dette valg blev taget på baggrund af kompleksiteten for opsætningen af Raspberrien, kravet om batteri og nem opsætningen af softwaren på microcontrolleren. Da Raspberrien kører Linux som operativsystem kræver det installation af Windows Subsystem for Linux for effektivt at kunne kommunikere imellem PC og Raspberry. Samtidig blev det vurderet at der ikke var behov for internet til prototypen, da denne blot skulle vise sammenhængen mellem software og hardware, samt muligheden for implementation af pathing-algoritme.

Da valget af microcontroller og programmeringssproget var fuldført, var det næste at kigge på hvilke fysiske opkoblinger Arduinoen skulle have. Her blev der refereret til kompleksitetskravene og problemformuleringen med underspørgsmålene, for at se hvad komponenter der bedst ville passe. Der blev valgt to servoer til at bevæge hjulene, en omdrejningsmåler til at måle hvor langt robotten har kørt samt et Bluetooth element til videreudvikling hvis tiden tillader det.

Del-konklusion:

På baggrund af en undersøgelse af de to microcontrollers Raspberry Pi 3B+ og Arduino Uno, blev det vurderet at Arduino ville passe bedst til kravene for projektet. Unoens lave strømforbrug gør den bedre egnet til brug med batteri, dens simple kompleksitet gjorde udviklingen på, og opsætningen af, den markant nemmere.

Unoen opfyldte de kompleksitetskrav der blev vurderet nødvendige for at kunne drive robotten.

### Arduino Uno

Arduino skabte deres første microcontroller i 2005 til brug til IoT løsninger og undervisning. Der er sidenhen blevet udbygget i udvalget af microcontrollers i styrke og størrelse.

Arduino har lavet deres eget IDE med dertilhørende compiler og upload metode til en tilkoblet Arduino microcontroller, hvilket gør udviklingen og testing af koden utroligt simpel. Sproget der er valgt at blive brugt er en modificeret version af C++, med forsimplede elementer til brug i deres eget sprog, og understøttelse for normal C++.

Unoen bruger en modificeret Main metode som indgangspunkt i programmet, hvor main er delt op i Setup() og Loop().   
Dette er gjort for at holde setup af de fysiske pins separat fra loop’et hvor primærkoden bliver kørt. Samtidig er dette gjort for at lave optimering af koden af compileren. Setup er det første der bliver kørt, hvorefter variablerne i setup bliver lavet til constants som ikke skal tjekkes igen.

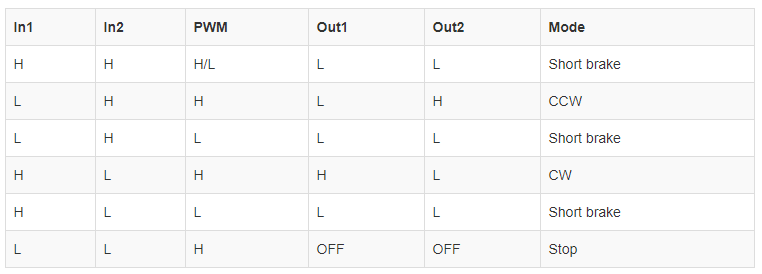
I projektet er der udover Arduinoen gjort brug af nogle forskellige hardwareopkoblinger der giver tilføjet funktionalitet og mulighed for påvirkning af fysiske elementer. Disse vil blive beskrevet med deres funktion og ikke deres virkemåde da dette er udenfor scopet for dette projekt.  
Ovenpå Arduinoen er der påsat et Shield med Servo-kontrol til de to motorer der bliver brugt. Et Shield er en i forvejen opsat hardwareløsning til en specifik funktionalitet.  
Udover dette er der placeret en omdrejningsmåler under Arduinoen, som måler hvor mange omgange et hjul har roteret, og på den måde kan afstandsmålingen af robotten kontrolleres.  
Til sidst er der påsat et bluetooth element, der gør kommunikation mellem Arduinoen og et andet bluetooth-device muligt igennem bluetooth teknologien. Dette er påsat for at gøre videreudvikling muligt, men er ikke gjort brug af i denne iteration af robotten da det var udenfor scope.

De individuelle komponenter kan kontrolleres af softwaren enten igennem Arduino’s eget kodebibliotek eller igennem udvidelser fra producenterne af hardwaren. I hardwaren er der gjort brug af Sparkfun’s Redbot chassis, og deres medfølgende kodebase til kontrollering af omdrejningsmåleren. (Magician\_Encoder, u.d.). Dette blev gjort da den fysiske implementation af hardwaren blev vurderet for kompleks til at kunne blive nået i dette projekt.

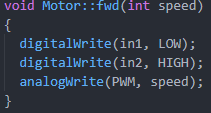
### Styring af motorerne.

Shield’et der er sat ovenpå Unoen har pins til de to motorer hvor hjulene er påsat. Disse pins er sat direkte ovenpå Unoens pins, og styrer hastigheden hjulene drejer rundt og hvilken retning der drejes. Disse kan manipuleres med Arduino’s eget library Arduino.h hvor der kan skrives digitale og analoge værdier til metoderne DigitalWrite() og AnalogWrite().   
Forskellen på de digitale og de analoge værdier er at de analoge værdier ikke er binære, de kan svinge mellem 0 til 255. Dette fungerer ved at Arduinoen sender strøm igennem den valgte pin i intervaller indtil et kald til den samme pin sætter spændingen til noget andet eller stopper spændingen. [[4]](#footnote-4)   
DigitalWrite er derimod binær da den kun kan tage imod enten HIGH eller LOW. Dette har forskellige værdier alt efter om det er en pin der er i Input mode eller Outputmode, hvor Inputmode er en on-off switch, mens Outputmode er enten 3.3V spænding eller 5V spænding.

Styringen af motorerne bliver gjort igennem manipulation af pins’ene på Unoen.   
Disse pins kan sættes som værende Inputmode eller Outputmode med et kald til pinMode(). Dette gør enten pin’en til en binær on-off pin, eller en pin med varierende strømmængde igennem sig. Alt efter hvad setting en pin er sat til, vil servoen der er forbundet til Shield’et ovenpå køre Clockwise, Counterclockwise, bremse midlertidigt eller stoppe.  
Dette kan ses på diagrammet nedenfor.

[[5]](#footnote-5)

Baseret på diagrammet blev et bibliotek lavet til abstrahering over den fysiske manipulation af pins’ene, med implementation af Motor klassen der indeholder funktioner såsom Forward, Backward, Left og Right. Fwd funktionen ser sådan ud.



Det er en privat funktion der bliver kaldt af Cpp filen Motor.cpp til at sætte hjulene til at køre Counterclockwise. CounterClockwise blev sat da der i den fysiske opsætning kom til at blive byttet om på motorerne.

Efter motorerne blev mulige at køre med var det nødvendigt at have en måde at holde tjek på hvor langt robotten havde kørt. Til dette var planen at der skulle bruges en anden komponent i form af en omdrejningsmåler. Denne viste sig dog at kræve et library der ikke var tilgængeligt på producentens hjemmeside grundet opgradering af hardware.   
det blev dermed nødvendigt med en anden løsning til at kontrollere hvor langt robotten havde kørt. Dette blev gjort ved at forsinke robotten med delay() metoden fra Arduino.h. Dette var ikke en optimal løsning, men vil fungere til prototypen af robotten, indtil et andet library kan blive fundet der passer til komponenten, eller en anden komponent kan blive fundet med dertilhørende library.

Det var ikke muligt at lave et library selv inden for tidsrammen, da dette kræver et højere niveau af undervisning i elektronik og spænding end hvad undervisningsmaterialet der er fundet til dette projekt kunne formulere.

### Lokalisering og simulering af robottens placering i et miljø.

I dette projekt er der gjort brug af en abstraktionsmetode hvor robottens verden er simuleret igennem et todimensionelt grid. Her er robottens start- og slutpunkt kendt. Dette gør det muligt at lave en algoritme der kan bestemme en rute fra startpunkt til slutpunkt, hvor nogle kriterier kan blive taget højde for.

Et grid kan efterkomme flere forskellige algoritmer alt efter opsætningen. Der er i dette projekt opsat en simpel fire-retningsbaseret algoritme der kan bestemme den korteste mulige rute for robotten. Denne kan udbygges med forhindringer i form af punkter som robotten ikke må køre på, samt der kan bruges optimeringsalgoritmer til at undersøge den korteste mulige rute fra et punkt til et andet.  
Afstanden imellem hvert punkt i grid’et blev simuleret ved at forsinke robotten med et sekund. Dette havde til en vis grad den samme effekt som omdrejningsmåleren havde, da det blev muligt at fake en afstand imellem hvert punkt.

### Pathing

På baggrund af simuleringen af robottens placering i verdenen ved hjælp af et todimensionelt array, var der behov for en pathing algoritme der kunne finde den mest optimale vej fra UvU’s nuværende placering til den ønskede placering. Dette afsnit vil omhandle valg af algoritme, hvordan algoritmen fungerer og hvordan algoritmen er implementeret. I projektet er der valgt at kigge på henholdsvis Dijkstras algoritme og A\*, samt en forsimplet algoritme der er brugt i prototypen af robotten.

Den mest effektive vej igennem et netværk er et problem der først blev dokumenteret af Leonhard Euler i 1736 med sit Seven Bridges of Köningsberg problem (Euler, 1736), hvor han forsøgte at finde en rute over et sæt af syv broer, ved kun at bevæge sig over hver bro én gang.  
Dette lagde fundamentet for grafteori og topologi i kort, som kan modelleres med grafer med parvise forbindelser mellem hvert punkt.

En graf er opbygget af en serie af forbundne punkter – også kaldet noder - hver med en *edge* hen til den næste node og en bestemmelse af henholdsvis en identifier af sig selv og en reference til den tidligere node. En graf er dermed en datastruktur, ofte en liste, bestående af en serie af parvise noder - Disse grafer kan være både directed og undirected, hvilket henviser til retningen man kan bevæge sig igennem grafen.

Disse parvise noder kan man dermed iterere igennem ved hjælp af søge algoritmer. To eksempler på søgealgoritmer der bruges i grafer er Depth First og Breath First. Hver søger igennem en hel graf indtil alle noder er undersøgt, dog med forskel på hvordan de gør det.   
Depth-first søges ned igennem alle grene i en graf indtil den ikke kan komme længere. Så går den tilbage igen og tager den næste gren osv. indtil alle noder er undersøgt (Se bilag 1; Depth-first Search).   
Breath-first search gennemgår i stedet alle de tilhørende noder fra den første node. Så vælger den en af de tilhørende noder til den første node og besøger alle dens tilhørende node igen, hvorefter den går tilbage og ser om andre tilhørende noder til den første node har nogle tilhørende noder før den går videre igen (se bilag 2; Breath-first Search).

Det der gør en pathing algoritme anderledes fra en søgealgoritme, er tilføjelsen af vægte til hver node i grafen samt deres edge hen til den næste node. Dette gør det muligt at vælge den mest effektive rute igennem grafen, ved at tjekke hver enkelt mulighed fra den nuværende node i grafen, videre til den næste node indtil man er ankommet til det ønskede endpoint baseret på nodernes vægt. Dette blev først gjort af Dijkstra hvis algoritme fungerer ved at give hver node og edge i grafen en vægt, og derefter iterere igennem hele grafen for at finde den mest effektive rute.

Denne algoritme har den matematiske formel F og G er det samme her, da Dijkstra kun tager højde for den reelle pris for at bevæge sig fra en node til den næste indtil man er ankommet til slutningen. Dette er utroligt ineffektivt, da der ikke bliver tjekket om man egentlig går tilnærmelsesvis den rigtige retning hen mod endepunktet.   
Dette problem fik en løsning igennem A\* algoritmen der tilføjer heurestics, til algoritmen. Denne er defineret som afstanden fra den nuværende node til endpointet – hvilket selvfølgelig først kan tjekkes ved at finde ud af hvor endpointet er henne, hvilket kræver en søgealgoritme først, eller et allerede kendt endpoint.

Pathing i projektet gør ikke brug af A\* eller Dijkstra’s, da implementationen af disse var for tidskrævende. I stedet er der brugt en forsimplet version hvor startpunktet og endepunktet har en x og y værdi, og en algoritme så indsætter retninger i en liste indtil det nuværende punkt og endepunktet er det samme.

I senere iterationer af projektet vil der blive gjort brug af A\* algoritmen. Denne algoritme er en overbygning af Dijkstra’s algoritme, og bliver brugt til at finde den mest effektive rute fra startpunktet til slutpunktet, ved at tage højde for vægtningen af hvert punkt og den herestiske værdi, som repræsenterer hvorvidt punktet er i retning af slutpunktet.

### Styring af robotten.

Robotten skal på en eller anden måde kunne modtage et punkt A og B som den skal køre imellem. Dette skal gøre det muligt på et senere tidspunkt at bruge robotten til fragtning af objekter mellem flere forskellige mulige punkter. Dette blev vurderet muligt på nogle forskellige måder, med hver deres fordele og ulemper;  
Der kunne være en forbindelse til en database der indeholdte Task’s med punkt A og B samt en prioritering.  
Der kunne også laves forbindelse til en håndholdt enhed som kunne kontrollere robotten, enten for at styre robotten eller for at give robotten to punkter at køre imellem.

Databaseforbindelse:

|  |  |
| --- | --- |
| Fordele | Ulemper |
| autonomisk | Kræver databaseopsætning |
| nem at skalere til flere brugere | Kræver måde at lægge opgaver ind i databasen |
| nemmere at lave som produkt der kan videreudvikles. | Kræver networkinghardware på robotten |
|  | Højere kompleksitet |
|  | Kræver konstant trådløs forbindelse. |
|  |  |

* Databaseforbindelsen ville være optimal hvis der var tale om flere brugere, eller hvis der var brugt Raspberry Pi som microcontroller, da denne allerede er opsat med trådløs forbindelse og har med sin markant højere hastighed meget bedre muligheder for multitasking til tjek af database og styring af software.

**Trådløs forbindelse til håndholdt enhed:**

|  |  |
| --- | --- |
| Fordele | Ulemper |
| god til prototyping af forbindelse mellem robot og telefon / pc. | Trådløs teknologi med egen teknologistack. |
|  | Kræver hjemmeside / app. |
|  | Passer dårligt til formålet med robotten. |
|  | Sværere at skalere |

* Denne løsning virkede bedre end databaseløsningen, men krævede en hjemmeside / app for at kunne blive brugt, hvilket var udenfor scope for dette projekt. Det ville også fjerne det autonomiske aspekt af robotten, og derfor fjerne formålet.
* Dette kunne også blive gjort med en app på telefonen der havde et grid, hvor der kunne indsættes et A og B punkt. Grid’et skulle så sendes til robotten som kørte den mest optimale rute imellem punkterne. Dette ville dog også kræve en opsætning af en app, samt undersøgelse af bluetooth teknologien og blev derfor vurderet for tidskrævende til dette projekt.

Baseret på de to muligheder blev det vurderet at senere iterationer ville fungere bedst med databaseopsætningen. Dette ville kræve et skift til Raspberry Pi i stedet, for at gøre multitasking og netværksopsætning så simpel som muligt. Denne løsning ville gøre det nemt at skalere til flere robotter på én gang, der hver udfører en Task, og ville gøre det muligt at bruge robotten kommercielt, samt ville gøre det muligt for flere brugere at indsætte tasks der skulle udføres.

### Delkonklusion

Undersøgelsen af microcontrollerne viste at Arduino ville være det bedste valg til prototypen af robotten, mens Raspberrien ville passe bedre til det endelige produkt. På baggrund af analysen af de udviklingsmæssige muligheder blev det også vurderet at Arduino ville passe bedst med programmeringssproget C++, da denne allerede var opsat med C++ i mente. Dette gjorde sig gældende for både compileren og ressourcerne online til de individuelle komponenter.

## Design

Designet af robotten blev gjort på baggrund af de kompleksitetskrav der tidligere blev udformet. For at kunne løse problematikken med en robot der kan fragte objekter fra et punkt til et andet blev handlingen delt op i mindre usecases, hver med sit eget element af det overordnede problem.   
Usecasene blev opdelt efter prioritering for bedre at give en retning for udviklingen af robotten. Dette blev primært gjort for at kunne besvare spørgsmålet ”er det her virkelig vigtigt lige nu”.

Udviklingsmetoden af robotten kom på denne måde til at ligne en agil udvikling. Hver usecase udgør en feature, og hver feature bliver implementeret for sig. Dette viste sig at fungere bedre end en vandfaldsmodel, hvor der bliver arbejdet på projektet samlet set.   
Denne udviklingsmetode blev underbygget af arkitekturens modularitet, hvor der kunne indsættes og fjernes funktionalitet uden at påvirke resten af systemet. Dette var specielt vigtigt til test af librariet for omdrejningsmåleren.

Der er ikke lavet fully dressed usecases da det blev vurderet dette var unødvendigt baggrund af manglen på aktører og stakeholders. I stedet er der lavet barebone usecases med krav, prioritering og forventet tidsbrug vs. aktuel tidsbrug. Denne beslutning blev taget da usecases beskrev kompleksitet og rækkefølge, i stedet for ROI.

### Usecases

**Robotten kører.**

* Prioritering: 1

|  |  |  |  |
| --- | --- | --- | --- |
| Krav | Research tid. | Tid | |
|  |  | Forventet | Aktuel |
| opsætning af hardware | 2-3 timer | 1 time | 1 dag |
| programmering af software. | 1 dage | 2 timer | 4 dage |
| testing | 0 timer. | 1 time | 1 time. |

* Denne usecase lagde grundlag for al videre funktionalitet. Der var dog problemer i form af opsætning af pins. Den guide der blev fulgt til opsætning af pins’ene var ikke korrekt i dens pin numre. Dette var blevet overset, hvilket førte til den ene servo kun fik halv strøm eller slet ikke virkede. Denne fejl gjorde den aktuelle tidsbrug på usecasen til markant længere end den forventede tid.

**Pathing algoritme bliver undersøgt og implementeret**

|  |  |  |  |
| --- | --- | --- | --- |
| Krav | Research tid. | Implementationstid | |
|  |  | Forventet | Aktuel |
| Grafer | 2-3 timer. |  |  |
| søgealgoritmer | 2 timer. |  |  |
| Dijkstra | 1 time. |  |  |
| A\* | 1 time. |  |  |
| Simpel quad-direction | 1 time. | 2 timer. | 2 timer. |
| Test | 0 | 1 time | 1 time. |

* Denne usecase involverede markant mere undersøgelse end reel implementation. Dette skyldes at A\* pathing algoritmen, specifikt heurestic delen af den, viste sig at være markant mere advanceret at implementere end forventet. Samtidig blev dette kigget på for sent i processen, grundet mængden af tid der blev brugt på hhv. Studering af C++ materiale, Raspberry Pi opsætning, skiftet til Arduino samt opsætningen af Arduino.

**Robotten kører fra punkt A til punkt B.**

## Arkitektur

Arkitekturen i robotten har to formål; At interagere med hardwaren uden at skabe for meget kobling og at holde samhørighed høj i de individuelle segmenter af arkitekturen.

Da Sketches i Arduino’en har et overordnet lag hvor loop er, samt der er et hardwarelag hvor interaktion med hardwaren bliver gjort, var det oplagt at bruge en modificeret tre-lags arkitektur, da både modellag og UI / topcontroller-lag allerede var til stede. Arkitekturen blev opbygget nedenfra med Model laget som agerede ansvarlig for hardwaremanipulation. Denne bliver styret af en controller som igen bliver styret af Super. Hver controller har sig eget fokusområde for at efterkomme Separation of Responsibility princippet.

De tre lag med deres segmenter er:

UvU (Super):

* Kender til Controllers.
* Kan kalde på metoder på controllers for at styre funktionaliteten.

Controllers:

* CtrMotor
  + Har kendskab til Motor klassen. Har private Motor objekter på klassen der bliver instantieret i constructoren, da der aldrig skal instantieres andre end 2 motorer.
* CtrPlanning
  + Har grid til lokalisering og start- og endepunkter.
  + Har kendtskab til Node, da denne skal bruges til A\* i senere iterationer.
  + Har ruteplanlægningsdelen, som pt er meget simpel men med mulighed for udbygning til en A\* algoritme i senere iterationer.

Denne opsætning gjorde arkitekturen meget modulær, da hvert segment har sit eget ansvarsområde vil koblingen automatisk forblive lav. Dette gør det nemt at videreudvikle robotten, da der ikke skal bekymres om kobling og samhørighed så længe hver controller har sit eget fokus, samt det er muligt at teste hvert segment for sig.

### Delkonklusion;

Den valgte tre-lags arkitektur har gjort samhørigheden høj mens koblingen forblev lav. Al kommunikation i arkitekturen går igennem controllers som bliver kaldt af Arduino

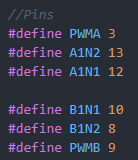
# Implementering

Da fokusset for dette projekt var udviklingen af en robot der kan fragte objekter, er der i denne sektion valgt at blive kigget på Motor klassen, der har opsætningen og manipuleringen af Arduinoens servoer, og pathing algoritmen der er blevet implementeret. Dette blev gjort da de repræsenterer

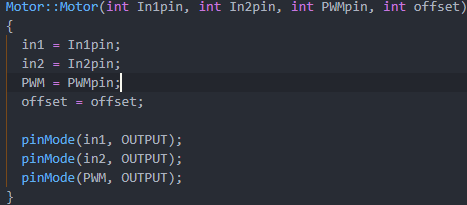
## Motor

Motorklassen har i sin funktionalitet opsætningen af pins’ene som de to servoer er påsat. Dette bliver gjort i constructoren, da der ikke skal opsættes andre servoer på robotten. Hvis dette var nødvendigt ville det være simpelt at lave en setter der kunne gøre det på et senere tidspunkt.

Constructoren bliver kun kaldt af CtrMotor, som opsætter motorerne med pins der er defineret med preprocessormacroer. Dette skyldes at pins’ene kun ændrer sig hvis hardwaren ændrer sig. Ellers er det unødvendigt at ændre på dem.

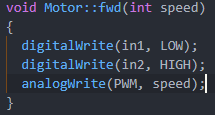


Objekterne er opsat med A1N1, A1N2, PWMA og Offset. A1 repræsenterer de to pins på Arduinoen, PWMA står for PowerMain A og Offset er en int der kan ændre retningen hjulene drejer ved at inverse tallene der bliver brugt ved enten at sætte offset til 1 eller -1. På den måde bliver 255 til   
-255, og hjulene kører omvendt. Dette blev sat op for ikke at skulle bytte om på hardwaren, så det i stedet kunne gøres igennem softwaren.

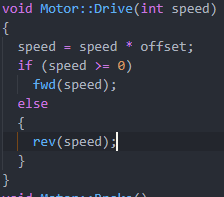
[[6]](#footnote-6)

Som det fremgår på billedet bliver der sat mode på pins’ene in1 og in2. Disse er lokale variabler der bliver instantieret i constructoren med OUTPUT som mode, igennem en metode fundet i Aruino.h Librariet. Denne mode gør det muligt at føre stærk strøm igennem pin’en, da resistoren foran pin’en er slået fra. (Arduino, u.d.)

Motorklassen gør brug af metodern Fwd og Rev, private metoder som hhv sætter hastigheden på en servo positiv eller negativ med en variabel mellem -255 og 255.



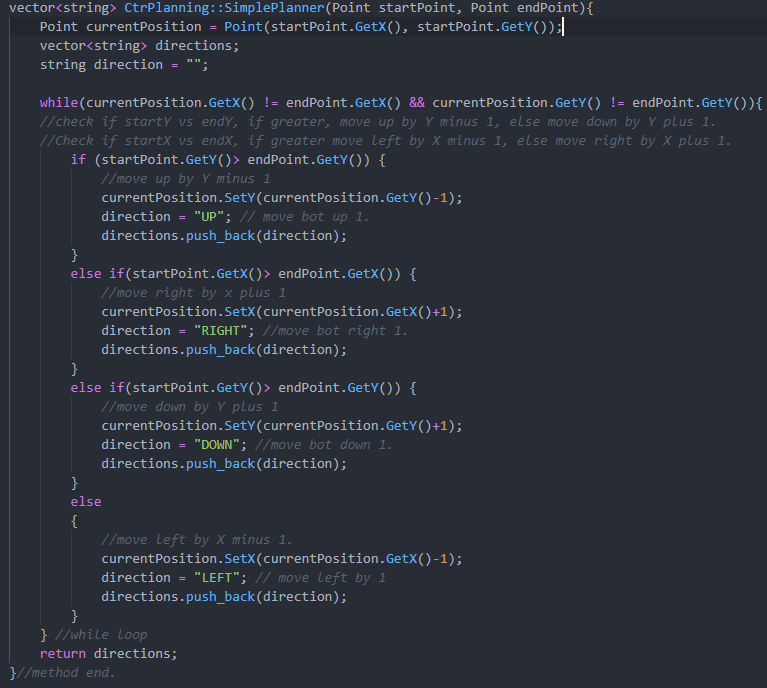
Denne metode bliver kaldt internt på objektet af de andre metoder, hvor den sætter pin1 og pin2 til hhv. LOW og HIGH, hvorefter den sætter outputmængden fra PWM til indholdet af variablen speed. Denne har en grænse på 255 der bliver overholdt af metoden selv.



Drive er metoden der bestemmer om fwd eller rev bliver kaldt baseret på offset. Dette gør det muligt at sætte inden man kalder Drive, for at få baglæns kørsel i stedet.

## Pathing

Pathing i projektet er implementeret med en simpel algoritme der bruger et todimensionelt array bestående af en X- og Y-akse. Den tjekker så værdierne på Start og Slutpunktet og danner en quad-directional rute indtil den når til endpointet. Derefter bliver en liste med retninger returneret som robotten kører.



Metoden SimplePlanner kan ses her. Den tager to objekter Point, som er simple objekter bestående af 2 ints, X og Y. Den returnerer en Vector bestående af String. Vector er C++ versionen af ArrayList; et array der har en størrelse der kan udvides efter dens instantiering. (Cpp-foundation, u.d.).  
Metoden bruger et Point kaldet currentPosition til at repræsentere hvor den er på grid’et lige nu. Denne bliver sat til startpunktet og opdateret i takt med algoritmen danner ruten. Når currentPosition er lig med endPoint bliver vectoren returneret. Algoritmen har en O-notation af O(N) hvor N er Dette gør algoritmen lineær i effektivitet, da den skalerer baseret på afstanden af end- og startPoint’s X og Y værdier.

# Refleksion

Projektet UvU-bot gik bedre end forventet. Der var en del problemer, specifikt med operativsystem, programmeringsmiljø og hardwareopsætning. Det var imponerende frustrerende, da jeg på ingen måde kunne finde ud af hvad problemet var, specielt med dualbooting. Dette satte computeren ud i et par dage indtil en løsning blev fundet. Studeringen af det valgte materiale gik ganske smertefrit. Det tog en del tid, men jeg satte mig or at læse den samme mængde tid som havde jeg haft almindelig undervisning, både for at kunne nå at komme igennem det på en fornuftig hastighed, men også for at holde en rytme med at stå op om morgenen og arbejde. De to af kurserne stod at det ville tage 13 uger med 4 timer om ugen. Jeg nåede igennem dem på den måned jeg afsatte, så jeg er godt tilfreds.

## Undervisningsmateriale

Undervisningsmaterialet semestret blev bygget op af var selvvalgt og godkendt af Brian Hvarregård. Det bestod af 2 elementer; C++ og Algoritmer. Jeg ville gerne lære mere om C++ og hvordan man skabte robotter, hvilket betød jeg skulle lære et nyt programmeringssprog og hvordan man koblede hardware og software sammen. Jeg valgte kurset på Pluralsight, C++ fundamentals, with Kate Gregory (Gregory, n.d.) som beskrev brugen, logikken og tanken bag C++ virkelig godt. Ud fra dette begyndte jeg at lave applikationer i C++, herunder noget grafisk UI med QT frameworket, og en inventory sorter til spillet Path of Exile.

Da jeg syntes jeg havde godt nok styr på C++ gik jeg videre til Algoritmer, med kurset (Roughgarden, u.d.). Det var et ret svært kursus, navnlig fordi det antog viden om teori jeg ikke havde. Jeg lærte dog en imponerende mængde, specielt om at tjekke effektiviteten af en algoritme, om at undersøge Big O og se hvad der får en algoritme til at blive langsommere. Jeg har siden jeg startede med at programmere haft en tanke om at jeg skulle bruge så lidt ressourcer som muligt, som om jeg arbejder med en 1 MHz processor og 128 mb ram. Det her kursus hjalp mig til at komme væk fra den tankegang.

Materialet der blev fundet til studeringen af C++ var utroligt fyldestgørende. Katie, instruktøren på Pluralsightkurset (Gregory, n.d.), var virkelig god til at beskrive tingene. Det var tydeligt hun havde brugt C++ i lang tid, og var vant til at videregive information så det var forståeligt. Der var noget forvirring omkring brugen af header filer kontra cpp filer, og hvordan de hver især skulle sættes op, men ellers gav det rigtig god mening. Jeg vil dog sige at jeg syntes C++ er et utroligt bloated sprog som har mange ting der er direkte unødvendige, såsom structs. Til gengæld har man en kontrol over sproget og computeren igennem sproget, som var både skræmmende og imponerende.

Algoritmekurset var som nævnt ret hardcore. Der var en del mere matematik end jeg havde forventet, men det var spændende at finde ud af hvad det er der gør algoritmer optimale og hvad der forsinker dem meget. Specielt viden om hvordan computeren behandler koden når den er kompileret ned til machine code var brugbar og har fået mig til at genoverveje hvor mange variabler jeg rent faktisk har brug for.

Hvis jeg skulle lave projektet igen, med den viden jeg har nu, ville jeg droppe Raspberrien. Opsætningen og programmeringen af den er markant bedre skabt til andre applikationer og til at bruge Python i stedet som sprog. Jeg havde nok også taget fat i It-teknologerne markant oftere i stedet for at prøve at løse problemerne selv. Specielt hardwareproblemerne havde jeg slugt min stolthed over noget hurtigere.

Hvis jeg skulle arbejde videre med Uvu-botten ville jeg implementere A\* i fuld fjor, og gøre den omnidirectional i stedet for quad-directional. Det ville sagtens være muligt at implementere A\* i en quad-directional maskine, man får bare ikke nær så meget ud af algoritmen, som hvis det var muligt for robotten at bevæge sig i alle retninger.

## Fremgangsmåde for projektet:

Processen for udviklingen af projektet startede ud med nogle mål.

* Studer materiale til C++, så det kan bruges
* Studer materialet til algoritmer så en god løsning kan bruges i applikation.
* Find ud af hvordan man laver en robot.
* Undersøg microcontrollers, navnlig Raspberry Pi og Arduino.
* Undersøg hardware til robotten så den kan køre.

Måden dette blev sat op på var ved at bruge Kanban, for at holde styr på hvor i projektet jeg var henne. Jeg brugte Trello til at holde styr på tasks, hvor jeg inddelte mit board i 5 sektioner. Jeg brugte så Trickle-down til at flytte tasks så der maksimalt var 3 på Doing og 1 på testing.

|  |  |  |  |
| --- | --- | --- | --- |
| To do | Doing | Testing | Done |

I den første måned af projektet var fokusset på at studere materialet til C++ der var blevet indsamlet. Det indsamlede materiale bestod af teorien omkring C++, hvordan man programmerer i C++ og hvordan compileren fungerer. Compileren blev der læst yderligere op på, da dette var et essentielt stykke viden, for at kunne sikre sig at koden opførte sig ens på Pc’en der blev programmeret på, og microcontrolleren koden blev kørt på. Denne måned gik forholdsvis smertefrit, med den eneste hindring værende opsætning af programmeringsmiljø. Dette skyldes Visual Studio har deres egen C++ compiler, som ikke er kompatibel med Raspberrien, så der skulle findes et alternativ. Alternativet blev først Atom, men opsætningen af IntelliSense på denne var en større opgave.  
Der blev der i stedet valgt Visuel Studio Code, som viste sig at være det bedste fra begge verdener; nem at bruge med IntelliSense fra Visual Studio, og nem at konfigurere ligesom Atom.

Efter programmeringssproget var det næste fokus at opsætte Raspberrien og udviklingsmiljøet til denne.   
Dette inkluderede opsætning af Ubuntu på maskinen, hvilket først blev gjort i en Virtuel Maskine, indtil det viste sig dette ikke var nok.  
Der blev afsat en uge til opsætning af Ubuntu og Raspberrien. Denne tidsramme blev overskredet da dual-booting af Windows og Ubuntu ikke ville fungere ordentligt. Dette viste sig at være et partitionsproblem, og krævede en ny installation af begge styresystemer. I stedet for ugen der blev afsat tog det to 2½ uger, hvor størstedelen blev brugt på Ubuntu installation, geninstallation, opsætning af Windows, undersøgelse og frustration.

Der var i starten planer om at bruge ROS, Robot Operating System, et software udviklet til skabelsen af robotter, primært i Python og C++, som skulle gøre udviklingen af robotten nemmere på Raspberrien. Dette viste sig dog at være avanceret at sætte op da det kun var funktionelt med Ubuntu, så det blev ignoreret da det ville tage for lang tid at opsætte Derefter blev der undersøgt udvikling og opkobling af Raspberrien direkte, udenom ROS. Dette var dog problematisk, specifikt fordi opsætningen af Ubuntu Linux tog utroligt lang tid, og da det endelig var sat op var det ikke funktionelt med programmering til C++ uden en stor skift i udviklingsprocessen. Der var også problemer i compileropsætning samt koden der blev skrevet på Ubuntu kunne ikke køre på Raspberrien, grundet en fejl i Makefile opsætningen. Denne tog omkring en uge at forsøge at løse, før der blev givet op. Det var her skiftet til en simplere microcontroller blev undersøgt. Denne undersøgelse bliver beskrevet i Microcontroller afsnittet af Teknologidelen af rapporten. Skiftet her gjorde det muligt at fortsætte med projektet,

Til design af robotten blev der brugt et Setup lavet af SparkFun der hedder RedBot. Dette er et ikke-længere understøttet chassis, som var muligt at låne. Designet af robotten har to lag. Øverste lag hvor Arduinoen er blevet fastsat, og nederste lag hvor servoerne har påsat hjul. Imellem de to lag er der ved hjælp af elastikker fastsat et batteri. Forrest på robotten er der sat et stabiliserende element med en kugle, der stoppede robotten i at vælte forover. Imellem de to servoer er der påsat omdrejningsmåleren på de to hjulaksler.

# Konklusion

Rapporten viser at man ved at bruge en Arduino Uno med to servoer, kan lave en prototype der kan fragte objekter fra et punkt til et andet. Denne kan bygges som en modificeret tre-lags arkitektur der kan designes med lav kobling og høj samhørighed til at understøtte en algoritme der gør robotten i stand til at finde en rute fra et punkt til et andet, og udføre ruten. Der blev igennem undersøgelse vist at kontrol af robotten igennem en ekstern enhed ville gå imod konceptet af robotten, og at robotten i stedet kunne udbygges med en databaseopsætning til senere iterationer. Dette vil

# Litteraturliste

Euler, L. (1736). Seven Bridges of Königsberg. I l. Euler, *Solutio problematis ad geometriam situs pertinentis.*

Gerry O'brien, K. G. (u.d.). *Introduction to C++*. Hentet fra edx.org: https://www.edx.org/course/introduction-c-plus-plus-3?source=aw&awc=6798\_1544896292\_bf7c4a012e9835f28aad776ad0aa6387&utm\_source=aw&utm\_medium=affiliate\_partner&utm\_content=text-link&utm\_term=301045\_https%3A%2F%2Fwww.class-central.com%2F

Gregory, K. (n.d.). *Pluralsight.com/courses/cpp-fundamentals*. From www.Pluralsight.com: https://www.pluralsight.com/courses/cpp-fundamentals

*Magician\_Encoder*. (u.d.). Hentet fra Github.com: https://github.com/sparkfun/Magician\_Encoder

Michael levin, N. R. (u.d.). Hentet fra https://www.coursera.org/learn/algorithms-on-graphs?ranMID=40328&ranEAID=SAyYsTvLiGQ&ranSiteID=SAyYsTvLiGQ-Ix87ntzgsE8YjRqgE25RXw&siteID=SAyYsTvLiGQ-Ix87ntzgsE8YjRqgE25RXw&utm\_content=10&utm\_medium=partners&utm\_source=linkshare&utm\_campaign=SAyYsTvLiGQ

O'Brien, G. (u.d.). *Intermediate C++.* Hentet fra edx.org: https://www.class-central.com/course/edx-intermediate-c-7590?utm\_source=fcc\_medium&utm\_medium=web&utm\_campaign=mooc\_report\_programming\_march\_2018

Roughgarden, T. (u.d.). *Algorithms: Design and Analysis, part 1.* Hentet fra Coursera.org: https://www.coursera.org/specializations/algorithms?ranMID=40328&ranEAID=SAyYsTvLiGQ&ranSiteID=SAyYsTvLiGQ-kEqRFJVy5zI2\_tyrU3nEUQ&siteID=SAyYsTvLiGQ-kEqRFJVy5zI2\_tyrU3nEUQ&utm\_content=10&utm\_medium=partners&utm\_source=linkshare&utm\_campaign=SAyYsTvLiGQ

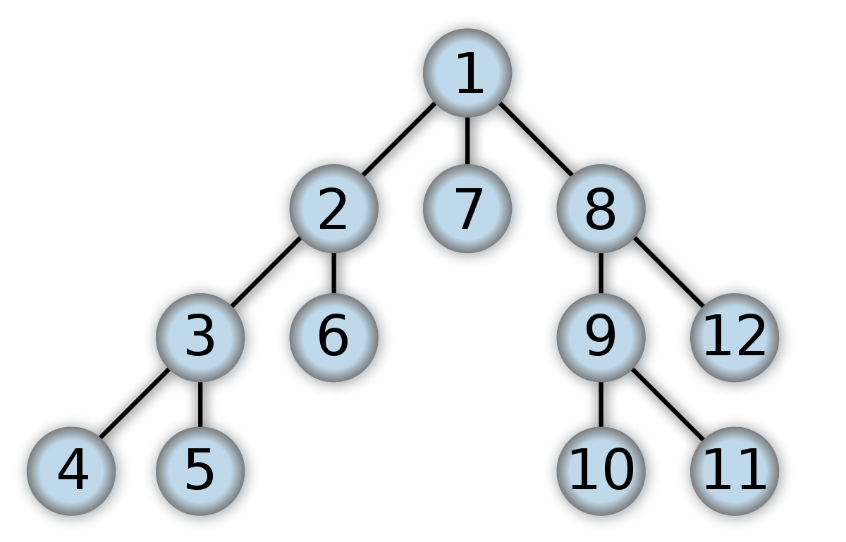
Stroustrup, B. (u.d.). Hentet fra http://www.stroustrup.com/

Webster, M. (u.d.). */robot*. Hentet fra merriam-webster.com: https://www.merriam-webster.com/dictionary/robot

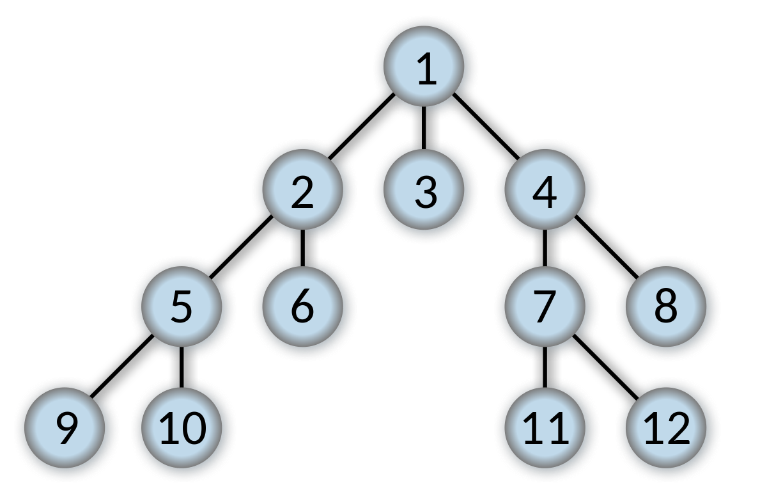
Wikipedia, I. d. (u.d.). *Include Directice, C / C++*. Hentet fra Wikipedia: https://en.wikipedia.org/wiki/Include\_directive

# Bilag

Angivelse af sti til repo.



Depth-first search, Wikipedia.



Breath-first Search, Wikipedia.

1. https://isocpp.github.io/ [↑](#footnote-ref-1)
2. https://www.arduino.cc/en/Products/Compare [↑](#footnote-ref-2)
3. https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/ [↑](#footnote-ref-3)
4. https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/ [↑](#footnote-ref-4)
5. <https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide/all>, diagram der beskriver pins’ene outputsettings for styring. [↑](#footnote-ref-5)
6. Motorklassen fra Objektlaget. Har funktionalitet for opsætningen af hardwaren. [↑](#footnote-ref-6)