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.

Contents

[Abstract: 3](#_Toc532328902)

[Forord: 4](#_Toc532328903)

[Problemområde 6](#_Toc532328904)

[Problemfelt 6](#_Toc532328905)

[Problemformulering 7](#_Toc532328906)

[Teknologi 8](#_Toc532328907)

[Styring af motorerne. 9](#_Toc532328908)

[Lokalisering og simulering af robottens placering i et miljø. 10](#_Toc532328909)

[Pathing 11](#_Toc532328910)

[Arkitektur 12](#_Toc532328911)

[Implementering 12](#_Toc532328912)

[Refleksion- 12](#_Toc532328913)

[Konklusion 12](#_Toc532328914)

[Litteraturliste 12](#_Toc532328915)

[Bilag 12](#_Toc532328916)

# 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)?

# Analyse

## 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 gøres da 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.

### 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**:**

* Intet operativsystem
* Nemme pins
* Bedst egnet til C++.
* Nemt library til brug af pins og skift af spænding
* Kan blive brugt af 4 AA batterier
* Super nem overførsel og kørsel af program

Ulemper:

* Langsommere end Raspberry
* Kun usb og strømkabel påsat
* kræver elektronik viden
* Ingen networking mulighed uden eksternt modul.
* Ingen harddisk.

**Raspberry Pi 3B+:**

Fordele:

* Hastighed
* Mere komplet pc
* Opkoblingsmuligheder udenfor pins
* Kræver ikke viden om elektronisk opsætning af hardware, kun software er nødvendigt
* Meget god til networking og internetbaserede løsninger
* Harddisk påsat.

Ulemper:

* Operativsystem man skal sætte sig ind i og som skal lukkes ned for at undgå korruption i filer
* Linux baseret kommandoer
* Højere strømkrav alt efter påsætning af ekstra dele, kræver specielt batteri.
* Mere besværlig opkobling af pins,
* Mere kompleks.
* Mere besværligt library at bruge, specielt til C++.
* Bedst egnet til Python

Som man kan se, er begge microcontrollers med deres fordele og ulemper valg 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 og projekter der skal kunne køre på batter i en længere periode.

For at finde ud af hvilken microcontroller der vil passe til robotten, blev der fastsat nogle krav robotten skal opfylde for at kunne besvare problemformuleringen.

Kompleksitetskrav

* Den skal kunne køre ved hjælp af servoer.
* Der skal være en måde at holde styr på hvor langt den har kørt.
* 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 batter 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. Det kunne også blive gjort med en Virtuel maskine der kører Linux distribution, dette blev dog vurderet for komplekst til tidsbegrænsningen for projektet.

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.

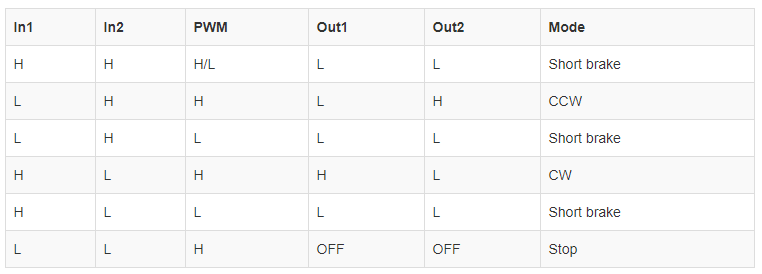
I projektet er der udover Arduinoen gjort brug af nogle forskellige hardwareopkoblinger der giver tilføjet funktionalitet og mulighed for påvirkning af fysiske elementer. Disse vil blive beskrevet med deres funktion og ikke deres virkemåde da dette er udenfor scopet for dette projekt.  
Ovenpå Arduinoen 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.

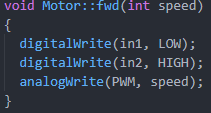
### Styring af motorerne.

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.

## Design

### Use cases

## 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. Denne blev opbygget nedenfra med Model laget som agerede ansvarlig for hardwaremanipulation. Denne bliver styret af en controller som igen bliver styret af Super.

De tre lag med deres segmenter er:

Super:

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

Controllers:

* CtrMotor
  + Har kendskab til Motor klassen. Har skabelse af Motor objekter til brug også.
* CtrPlanning

# Implementering

# Refleksion

Problemer

Hvad ville jeg have gjort anderledes

Fremtidige applikationsmuligheder for Uvu botten – modularitet med grafisk interface.

# Konklusion

# Litteraturliste

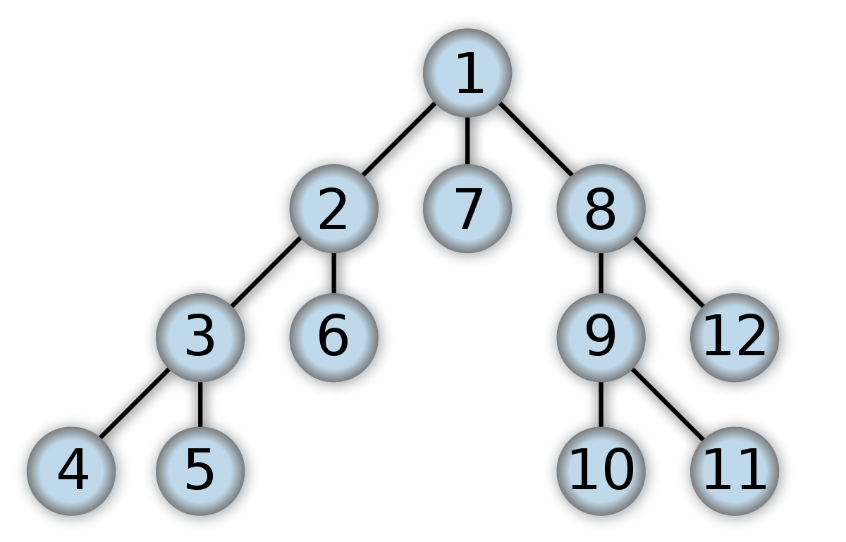
Euler, L. (1736). Seven Bridges of Königsberg. I l. Euler, *Solutio problematis ad geometriam situs pertinentis.*

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

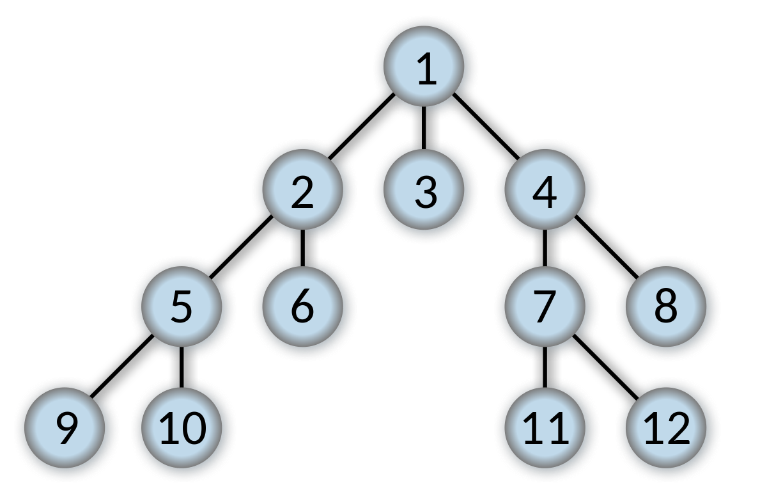
Webster, M. (u.d.). */robot*. Hentet fra merriam-webster.com: https://www.merriam-webster.com/dictionary/robot

# 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)