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:

Rapporten er skrevet af Dennis Bundgaard Rasmussen til 4. semester på Datamatiker uddannelsen på University College Nordjylland (UCN). Rapporten omhandler robotten UvU-bot og skabelsen af denne. 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

Robotter er i dag at finde overalt omkring os. Alt fra selvkørende biler til samlebånd og legetøj. Omkostningerne og processen ved at lave en robot er faldet drastisk efter små microcontrollers er blevet udbredt. Kulturen omkring skabelsen af robotter er vokset så meget at det er blevet muligt for hobbyister at lave deres egne robotter og Internet of Things løsninger uden at det bliver umuligt finansielt. Dette skift i økonomi har betydet at flere virksomheder producerer deres egne små generelle komponenter til brug for hobbyister og prototypeskabelse.

Med så mange muligheder kan det dog blive svært at finde frem til hvilken kombination af microcontroller og hardware der vil være mest passende. Der er stor forskel mulighederne de forskellige microcontrollers har, både i opkobling, sværhedsgrad af brug og hardwareudvidelse. Dette gør skabelsen af en robot til en mere advanceret affære, beslutninger skal tages for at finde den mest passende opsætning til at løse formålet med robotten.

Denne rapport undersøger hvordan en robot kan designes og implementeres til automatisk at fragte et objekt fra et punkt til et andet.

note: Der forekommer i dette projekt ikke en analyse af programmeringssprog, C++ blev valgt på forhånd som værende sproget der skulle bruges. Dette blev gjort da C++ er den valgte teknologi for dette semester.

# 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

## Design

## Arkitektur

Modificeret 3-lags arkitektur

Super er erstattet af Loop.

Cohesion / kobling

## 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++

I projektet er der brugt C++ som programmeringssprog. C++ blev valgt da det ligger meget tæt på hardware laget, og der dermed ikke er meget abstraktion imellem programmerede kommandoer 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 resourcer 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.

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 også 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. 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. Her kan man kigge på Java, C# og Python som alle er closed source.

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 Machine code med varierende former for optimering af hastighed eller læsbarhed, alt efter hvilken indstilling man har sat på compileren.
* 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 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.

### Arduino Uno

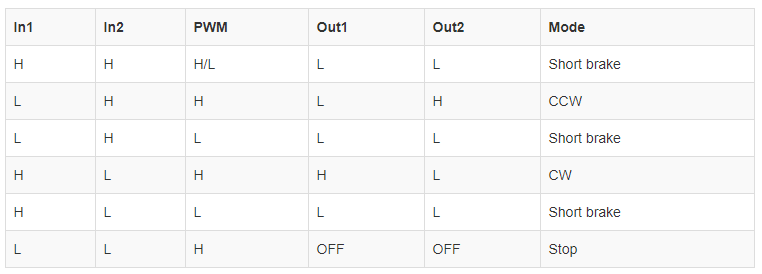
Programmer i Arduino kaldes for sketches, på baggrund af at Arduino primært er brugt til prototyping før man fører kodebasen over på en større eller mere specifik hardwareopsætning.  
Unoen bruger en modificeret Main metode som indgangspunkt i programmet, hvor main er delt op i Setup() og Loop(). Dette har den samme funktionalitet som main funktionen i andre sprog.

Formålet med denne opdeling er at holde det uendelige loop og setup’et separat, så der ikke bliver lavet opsætningen af hardware ved hver iteration af loopet. 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. Dette gør koden markant hurtigere, hvilket man kan se hvis man compiler koden, sætter en log til outputtet i Assembly i stedet for Machine Code og ser handlingerne computeren har tænkt sig at køre.

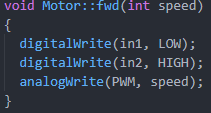
I projektet er der udover Arduinoen gjort brug af nogle forskellige hardwarekomponenter der kontrollerer de enkelte hardwarekomponenter. Disse vil blive beskrevet med deres funktion og ikke deres virkemåde da dette er udenfor scopet for dette projekt. Ovenpå Arduinoen er der påsat et Shield med Servo-kontrol 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 drejet, 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 stacken. 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 ligger uden for scopet af dette projekt.

### Styring af motorerne.

Styringen af motorerne bliver gjort igennem manipulation af pins’ene på Unoen. Dette gøres ved at sætte deres spænding til enten høj eller lav, på baggrund af dette diagram:

[[2]](#footnote-2)

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. Forwardfunktionaliteten ser sådanne ud:



hvor DigitalWrite er en funktion fra Arduino librariet der bliver brugt til at manipulere en pins outputmode i henhold til skemaet ovenover for enten at få motoren til at køre med, eller mod, uret. Dette kan ses i Diagrammet hvor CCW står for Counter Clock Wise. Her bliver pin

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

Simulation af verdenen for en robot er en problematik der kan løses på forskellige måder. Én måde er at tage et billede og behandle billedet. Dette kræver dog en genkendelsesalgoritme for at kunne bestemme hvad det er robotten kigger på. En anden mulighed er at lave en abstraktionsløsning over verdenen omkring robotten. Dette gøres ofte ved kun at videregive nødvendig information, såsom placering og afstande mellem to punkter, for at gøre det muligt for robotten at kunne bestemme sin egen placering.

I dette projekt er der gjort brug af abstraktionsmetoden hvor robottens verden er simuleret igennem et todimensionelt grid. Her er robottens placering kendt – dens startpunkt – og robottens endepunkt – dens slutpunkt. 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.

Lokalisering i et grid giver et specifikt problem; hvor lang er afstanden mellem hvert punkt i grid’et? Dette problem har forskellige løsninger, alt efter hvilket hardwaremodul der er tilgængeligt, og om et modul er tilgængeligt overhovedet. I projektet er der gjort brug af en afstandsmåler. Denne bestemmer mængden af omdrejninger hjulene har foretaget ved at blive påmonteret hjulenes spinaksel. Dette modul gør det muligt at fastsætte en værdi til hvert punkt i grid’et, hvor robotten kan lave et tjek og se om den har kørt langt nok til at kunne gå videre til næste handling i listen.

### Pathing

På baggrund af lokaliseringsmodulet i programmet og 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 Uvus 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\*.  
Den mest effektive vej igennem et netværk er et problem der først blev opdaget 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 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.

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. 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 blev gjort ved hjælp af A\* algoritmen. Denne algoritme er en overbygning af Dijkstra’s algoritme, og bliver brugt til at finde den mest effektive rute fra et punkt til et andet, igennem et netværk af forbundne punkter.  
Dér hvor Dijkstra’s algoritme og A\* algoritmen er forskellig, er at Dijkstra’s algoritme ikke har kendskab til distancen fra et givent punkt til endepunktet, og kan derfor ikke beregne den mest effektive vej fra startpunktet til endepunktet.

# 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.

1. https://isocpp.github.io/ [↑](#footnote-ref-1)
2. <https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide/all>, diagram der beskriver pins’ene outputsettings for styring. [↑](#footnote-ref-2)