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 omstændig affære kun muligt for virksomheder grundet høje omkostninger, til at blive noget der er muligt på hobby niveau. 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 en selektion af lommecomputere, med simple udbygning- og tilkoblingsmuligheder, arkitekturen som skal understørre 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 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:

Projektet er skrevet af Dennis Bundgaard Rasmussen. Det kunne ikke have været muligt uden hjælp fra vejlerede til rapport-struktur og specielt til Ib Helmer Nielsen og Steffen Vutborg fra It-teknolog uddannelsen. Disse to herrer var en uendelig hjælp til opsætning og forståelsen af hardwaren. Projektet havde været dødt i vandet uden dem.

Formålet med rapporten er at undersøge applikationsmulighederne for C++ til at programmere en robot. Der indgår ikke en diskussion af programmeringssprog, da C++ er teknologi-valget for dette semester. Rapporten er skrevet til eksamination til underviser og sensor, og til andre der kommer til at sidde med samme ønske om at lære mere om at programmere en robot med en lommecomputer.

Contents

[Abstract: 3](#_Toc530665582)

[Hovedafsnit 5](#_Toc530665583)

[Problemanalyse 5](#_Toc530665584)

[Problemfelt 5](#_Toc530665585)

[Problemformulering 5](#_Toc530665586)

[Proces 5](#_Toc530665587)

[Teknologi 5](#_Toc530665588)

[Arkitektur 5](#_Toc530665589)

[Implementering 5](#_Toc530665590)

[Refleksion 5](#_Toc530665591)

[Konklusion 5](#_Toc530665592)

[Litteraturliste (inkl. Alle kilder der er lavet henvisninger til) 5](#_Toc530665593)

[Bilag 5](#_Toc530665594)

# Hovedafsnit

## 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 hardware som Arduino og Raspberry er blevet tilgængelige, selv grupperne og kulturen omkring skabelsen af robotter er vokset så meget at hobbyister kan være med.

Denne rapport vil tage udgangspunkt i problemformuleringen:

*Hvordan kan en robot designes og programmeres, med fokus på valg af teknologi og robusthed i arkitekturen, til at kunne autonomisk finde en rute fra et punkt til andet*

* Hvad hardware komponent ville være mest realistisk at bruge i den tilladte tid?
* Hvordan kan arkitekturen i robotten designes til at holde kobling lav og samhørighed høj?
* Hvordan kan en model af virkeligheden simuleres så robotten kan kende sin egen placering.
* Hvordan kan en pathing-algoritme implementeres i robotten, og hvilke mulige algoritmer ville være brugbare i systemet?

Denne problemformulering er valgt, da den ikke afgrænser valg af hardware, lader softwarearkitekturen være åben nok til modificeringer er mulige og scoper ruteplanlægningen til endnu ikke valgt algoritme.

## Problemanalyse

## Problemfelt

Emnet for projektet er brugen af C++ til at styre en robot igennem en lommecomputer. C++ er valgt da det er et meget low-level sprog, hvilket henviser til niveau af abstraction over maskinsprog, og ikke niveau af sværhedsgrad. Robotten skal bruges til at undersøge pathing algoritmer og øve sammenkoblingen mellem hardware komponenter og software. Robotten skal kunne tage et input i form af et todimensionelt grid med en start og slutlokation og dernæst finde en rute imellem de to punkter.

Hvad er problemet

For hvem er det et problem

Hvordan kan vi løse det

## Problemformulering

*Hvordan kan en robot designes og programmeres, med fokus på valg af teknologi og robusthed i arkitekturen, til at kunne autonomisk finde en rute fra et punkt til andet*

* Hvad hardware komponent ville være mest realistisk at bruge i den tilladte tid?
* Hvordan kan arkitekturen i robotten designes til at holde kobling lav og samhørighed høj?
* Hvordan kan man forbinde robotten til en telefon, og hvilken teknologi vil være mest passende ( Wifi, bluetooth osv).
* Hvordan kan en pathing-algoritme implementeres i robotten, og hvilke mulige algoritmer ville være brugbare i systemet?

## Proces

## Plan

Formålet med projektet

Det første skridt i projektet var at vurdere hvilken onboard computer der skulle bruges som kontrolenhed. Dette blev gjort ved at sammenligne de to muligheder; Raspberry Pi 3B og Arduino Uno.

|  |  |  |
| --- | --- | --- |
|  | Raspberry Pi 3 | Arduino Uno |
| Programmeringssprog | C++, Python, Java, med flere. | C++, C, Python. |
| Kompleksitet | * *Styresystem*   Linux   * *Opsætning*   Middel besværlighed   * *Overførsel af filer*   USB kabel og pind   * *Test og debugging*   Test og debugging gøres seperat | * *Styresystem*   unødvendigt   * *Opsætning*   Er gjort   * *Overførsel af filer*   Automatisk upload af filer   * *Test og debugging*   Indbygget test og debugging. |
| udvidelsesmuligheder | I/O pin’s til komponenter, USB, HDMI og andre forbindelsesmuligheder | I/O pins til komponenter. Mulighed for påsættelse af Shields for forsimplet udvidelse. |

På baggrund af diagrammet blev der valgt at gå videre med Raspberrien. Til dette blev der undersøgt Linux styresystem, ROS (Robot Operating System), potentielle code environments, Git integration osv. Grundet vedholdende problemer i installations- og opsætningsprocessen blev der lavet en ændring i planerne. Der blev valgt at gå videre med Arduino i stedet, da denne har sit eget code environment, der ville ikke være compiler problemer, ikke nogen problemer med anderledes styresystem og overførsel af filer var markant mere strømlinet. Dette skift gjorde udviklingsprocessen markant nemmere.

Det var herefter meningen at en beslutning om programmeringssprog skulle forekomme. Dette er dog ikke relevant, da projektets primære fokus var tilegnelse af viden om C++ og brugen heraf.

## 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.

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 Arduino’s eget programmeringssprog en simplificeret version af C++ med nogle begrænsninger, hvilket betyder C++ koncepter er velkendte i Arduino udviklingsprocessen.  
C++ fungerer ved et 3-skridts 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. Dette kan føre til problemer hvor klasser er defineret flere steder, og dermed blevet uklart hvilken der er henvist til. Dette er i projektet løst ved at bruge pre-processorkommandoen ”#pragma once” hvilket fortæller pre-processoren den kun skal definere den én gang, og derefter lave referencer til den ene definition af filen. Dette vil normalt blive fanget af Linkeren.
* Compileren tager pre-processorerens output og laver denne fil om til assembly kode. Denne assembly kode bliver derefter lavet ind til Machine Code i form af et binært filformat.
* Linkeren tager object filerne som Compileren lavede og laver et library eller en executable fil. I dette stadie bliver alle objektfilerne linket sammen ved at referencer bliver sammenlagt med de rigtige adresser.

I C++ er der valgt at lave en separation i klasser i form af en header fil der fungerer som interface for klassen, og en .cpp fil der fungerer som implementation af klassen. Denne separation gør nogle forskellige problemstillinger nemmere at løse, herunder deling af kode,

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 hardware opsætning.  
Uno’en 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.

Det er vigtigt at pointere at selvom Setup er opsætning af forskellige pins og lignende, så er scopet på metoden det samme som alle andre scopes. Det er lokale variabler der bliver lavet. Hvis variablen ikke bliver lagt på heap’et, så er variablen ikke mulig at kalde på andre steder. Dette kan man komme rundt om, ved enten at lægge variablen på heap’et, eller ved at give variablet en pointer som kan references andre steder i koden. Når man bruger en reference til en memory address som parameter, IE en reference, så bruger man Pass by Reference. Dette er i modsætning til Pass by Value, hvor man bruger et objekt som parameter. Pass by Reference gør det muligt at ændre objektet man referer, uanset placering i koden, så længe objektet har en pointer, eller ligger på heap’et.

Dette gør det muligt at sætte multi-use objekter op i Setup() i stedet for som globale objekter, enten i form af memory addresses eller som heap-placerede objekter. I projektet er dette ikke gjort brug af, da der blev valgt at holde passes i parametre som pass by value i stedet for pass by reference for at undgå kompleksitet, og for at holde cohesion høj, da denne ville blive lavere hvis det var muligt at ændre på variablers indhold på tværs af klasser udenom encapsulation.

I projektet er der udover Arduino’en gjort brug af nogle forskellige hardwarekomponenter. Disse vil blive beskrevet med deres funktion og ikke deres virkemåde da dette er udenfor scope for dette projekt. Ovenpå Arduino’en er der påsat et Shield med Servo-kontrol til de to motorer der bliver brugt. Udover dette er der placeret en omdrejningsmåler under arduinoen, som kan måle hvor mange omgange et lille hjul har drejet, og på den måde kan afstandsmålingen af robotten kontrolleres. Til sidst er der påsat et bluetooth element, der gør kommunikation mellem arduino’en 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.

## 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 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 Dijkstra’s 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.

## Arkitektur

## Implementering

## Refleksion

# 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.