



CHALMERS

LÅT ROBOTEN VISA VÄGEN

Implementering av ett inomhusnavigeringssystem för
utveckling av roboten Meccanoid G15KS



Kandidatarbete utfört av:

Evelina Dahlsten, Johanna Gudmandsen, Jonathan Olsson,
Klara Persson, Vide Ramsten, Robert Slipac

Låt roboten visa vägen

Implementering av ett inomhusnavigeringssystem för utveckling av roboten Meccanoid G15KS

© Evelina Dahlsten, Johanna Gudmandsen, Jonathan Olsson, Klara Persson, Vide Ramsten, Robert Slipac, 2018.

Handledare: Jonas Fredriksson, Elektroteknik

Examinator: Yiannis Karayiannidis, Elektroteknik

Kandidatarbete 2018:18

Institutionen för Elektroteknik

Chalmers tekniska högskola

SE-412 96 Göteborg

Telephone +46 31 772 1000

Framsida: Bild på roboten Meccanoid G15KS, utvecklad av företaget Meccano, som detta projekt bygger på.

Förord

Först vill vi tacka vår handledare, Jonas Fredriksson. Tack för att du varit engagerad i vårt arbete under hela arbetets gång och tack för att du varje vecka varit med på våra möten och kommit med feedback för att hjälpa oss framåt.

Vi vill även tacka vår examiner, Yiannis Karayiannidis, för att du ville ta på dig ansvaret att bedöma vårt arbete. Ett speciellt tack för att du tog dig tid att träffa oss för ett möte innan slutrapporten skulle lämnas in och för den goda feedbacken du gav oss.

Vi vill vidare tacka fackspråk och biblioteket för deras tjänster. Föreläsningar och möten har varit väldigt givande och har gjort det lättare för oss att få en professionell utformning av vår rapport.

Avslutningsvis vill vi tacka institutionen för Elektroteknik som står bakom vårt projekt och låter oss arbeta inom ett område som vi tycker är väldigt intressant och givande.

Ett stort tack till er alla!

Chalmers Tekniska Högskola, Göteborg, Maj 2018

*Evelina Dahlsten, Johanna Gudmandsen, Jonathan Olsson, Klara Persson,
Vide Ramsten, Robert Slipac*

Abstract

Robotics is becoming more and more involved in the society it is clear that knowledge about construction and interaction with robots must spread. This Bachelor thesis focuses on the difficulties with indoor navigation, where the goal is to lead a user to a location in a room with the help of a robot. This paper describes a development of a navigation system implemented in a Meccanoid G15KS-robot. To make this robot easy to use, an Android application has been developed as its interface. This is made in a concrete and simple way and the user can, in a predefined room, select a few travel goals. When a request is sent, the robot will lead the user to this goal. The robot has a laserbased sensor to be able to analyze the environment around it. This has made it possible to create functionality such as localization as well as path planning. Utilizing the developed algorithms and the operating system ROS, the robot can move to a selected goal upon request.

Keywords: Application, Arduino, Bluetooth, Indoor navigation, Meccanoid G15KS, Navigation, Obstacle avoidance, Path planning, ROS, SLAM

Sammanfattning

Robotteknik utvecklas och blir mer integrerat i samhället. Detta medför att kunskapen om konstruktion och interaktion med robotar behövs i en större utsträckning. Detta kandidatarbete kommer att handla om svårigheter med och lösningar till inomhusnavigering, där målet i projektet är att kunna leda en person till en angiven plats. I rapporten beskrivs hur arbetet utvecklade en Meccanoid G15KS robot för att möjliggöra inomhusnavigering. För att göra roboten mer användarvänlig utvecklades även en Androidapplikation, där användaren kan välja några olika navigeringsmål. När en begäran från applikationen skickas kan roboten leda användaren till den rätta positionen. Roboten har en laserbaserad sensor för att kunna analysera omgivningen, med hjälp av denna möjliggörs vägplanering, hinderundvikning och lokalisering. Med hjälp av algoritmer och operativsystemet ROS kan roboten förflytta sig till den valda destinationen enligt begäran från användaren.

Nyckelord: Applikation, Arduino, Bluetooth, Hinderundvikning, Inomhusnavigering, Meccanoid G15KS, Navigering, ROS, SLAM, Vägplanering

Förkortningslista

Förkortning	Fullständigt uttryck
AMCL	Adaptive Monte Carlo Localization
EKF	Extended Kalman Filter
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
IPS	Indoor Positioning System
LiDAR	Light Detection And Ranging
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping

Begreppslista

Begrepp	Förklaring
Accelerometer	Ett instrument som vid en hastighetsförändring ger ifrån sig en elektrisk signal proportionell mot ändringen [1].
Arduino UNO	En typ av mikrokontroller [2].
Databuss	En ledning som utgör koppling mellan olika elektriska komponenter för kommunikation [3].
Gyroskop	Ett instrument för att mäta orientering och vinkelhastigheter [4].
I2C	Ett kommunikationsprotokoll som möjliggör kommunikation med flera moduler samtidigt över samma databuss [5].
LiDAR	Ett instrument för att mäta avstånd med hjälp av laserstrålar [6].
Meccanoid G15KS	En leksaksrobot skapad av företaget Meccano [7].
Mikrokontroller	En mindre dator, oftast bara ett kretskort som används som en del i exempelvis maskiner eller andra tekniska anordningar [8].
Nod	En nod är en process i ROS i form av ett skript som kan användas för databearbetning. Det kan ha ett välarbetat användargränssnitt och se ut mer som ett program än som en kod [9].
Open Source	Open source har betydelsen öppen källkod vilket innebär att materialet är fritt att använda [10].
Publisher	En publisher är i ROS en nod som kan skicka data till en topic [9].
ROS	Ett "meta"-operativsystem uppbyggt för att styra robotar [9].
Subscriber	En subscriber är en nod i ROS som läser data på en topic [9].
Topic	En topic är en meddelandekanal som noder kan skicka och mottaga data ifrån i ROS [11].

Innehåll

1	Inledning	1
1.1	Syfte och Mål	1
1.2	Problemformulering	2
1.3	Kravspecifikation	2
1.4	Avgränsningar	4
1.5	Relaterade arbeten	4
1.6	Rapportens struktur	5
2	Teknisk översikt	6
2.1	Meccanoid G15KS och bottenplattan	6
2.2	Drivning	7
2.3	ROS	7
2.4	Arduino UNO	8
2.5	Bluetooth-modul	9
2.6	LiDAR	9
2.7	IMU	10
2.8	Systemöversikt	11
3	Implementering av delsystem	13
3.1	Användargränssnitt	13
3.1.1	Val av användargränssnitt	14
3.1.2	Applikationens uppbyggnad	15
3.2	Lokalisering	16
3.2.1	Val av lokaliseringssystem	16
3.2.2	Algoritmer	17
3.2.3	Implementering av SLAM	19
3.3	Vägplanering och hinderundvikning	21
3.3.1	Val av vägplanering- och hinderundvikningsalgoritmer	21
3.3.2	Algoritmer	21
3.3.3	Implementering av vägplanering	23
3.4	Kommunikation mellan delsystem	24
3.4.1	Kommunikation - Applikation	25
3.4.2	Kommunikation - SLAM och Vägplanering	25
3.4.3	Kommunikation - Drivning	26
4	Kravverifiering	29
4.1	Delfunktioner	29
4.2	Säkerhet	30
4.3	Prestanda	32
4.4	Användargränssnitt	32
4.5	Sammanfattade resultat	32
5	Diskussion	34

6	Slutsatser och framtida arbete	36
	Referenser	37
A	Systemet i ROS vid navigering	I
B	Systemet i ROS vid SLAM	II

1. Inledning

Robotteknik är på framfart och automatiseringen breder ut sig. Idag går det att finna mobila robotar såväl på uppdrag på Mars som i tunnelbanan i Paris där de städar och håller rent [12]. Allt fler arbetsuppgifter utförs även i Sverige av robotar och upp emot 53 % av alla svenska jobb kan komma att automatiseras inom 20 år [13].

I takt med den ständiga utvecklingen kommer det behövas fler personer med kunskap om robotteknik och interaktionen mellan människa och robot. Det är människan som bygger framtiden och det är framförallt unga som kommer föra utvecklingen framåt. Därför är det extra viktigt att redan i tidig ålder introducera barn till teknik och få dem intresserade av dess breda användningsområde.

Det är just i barnens intresse som detta projekt kommer in i bilden. Plattformen som projekt bygger på, en Meccanoid G15KS-robot utvecklad av företaget Meccano, har skapats för att kunna ge barn och ungdomar ett roligt och intressant sätt att lära sig om automatisering och programmering [7]. Samma plattform som används i det här arbetet har använts i tidigare studier [14][15][16]. Ett tidigare arbetet resulterade bland annat i en ombyggnation av roboten. En bottenplatta med hjul skapades och ersatte fötterna för att öka stabiliteten och robotens hastighet. Det här arbetet kommer behandla andra utvecklingsområden, men återanvända vissa delar av tidigare utförda arbeten.

Specifikt behandlar detta projekt implementering av ett inomhusnavigeringssystem. Meccanoid-roboten ska kunna navigera utifrån en förutbestämd karta och samtidigt veta var på den specifika kartan som roboten själv befinner sig. Användaren ska kunna välja vart roboten ska åka och sedan ska den själv hitta till platsen och på vägen dit undvika hinder. Detta för att i förlängningen kunna fungera som en receptionist som ska följa besökare till platser de letar efter. Arbetet kommer utföras på Chalmers Tekniska Högskola och därmed kommer roboten testas i skolmiljöer. Förhoppningen är att roboten ska demonstreras i barnens egna miljö för att väcka deras intresse.

1.1 Syfte och Mål

Syftet med projektet är att förbättra kunskapen och förståelsen för inomhusnavigering. Detta uppnås genom att implementera ett navigeringssystem i en mobil robot. Målet med projektet är således att vidareutveckla roboten Meccanoid G15KS så att den kan navigera sig fram i ett område. Utöver detta är förhoppningen att roboten ska fånga barns och ungdomars intresse med sitt lekfulla utseende och få dem intresserade av att vilja arbeta med teknik i framtiden.

1.2 Problemformulering

Arbetet har delats in i fyra frågeställningar som undersöks och besvaras i senare delar av rapporten. När dessa frågeställningar är uppfyllda kommer robotens komponenter samverka på ett sådant sätt att roboten kan visa besökare dit de ska. Frågorna som lägger grunden till arbetet och som ska besvaras är följande:

- Hur ska roboten veta sin nuvarande position?
 - För att göra det behövs en lokaliseringsmetod.
- Hur ska roboten förflytta sig från punkt A till punkt B ?
 - För förflyttningen behövs en vägplaneringsmetod.
- Hur ska roboten undvika hinder?
 - För detta behövs metoder för hinderundvikning.
- Hur ska användaren kommunicera med roboten?
 - För kommunikationen behövs ett användargränssnitt.

Områden som arbetet kommer behandla är lokalisering, vägplanering, hinderundvikning och design av ett användargränssnitt. För att kunna avgöra robotens position behöver roboten någon form av sensor- och/eller kamerasystem. Förflyttning till navigeringsmålet sköts med en algoritm som kan hitta en väg. Även hinderundvikning kan hanteras med hjälp av en sådan algoritm men också från stöd av sensorer. Ursprungligen fanns ultraljudssensorer installerade och programmerade för att undvika kollision, men vidareutvecklingen kommer fokusera på att hitta en mer pålitlig lösning.

1.3 Kravspecifikation

Frågeställningarna som presenterades i föregående avsnitt konkretiserar uppgiften och ger en överskådlig bild av projektet. Frågeställningarna är dock väldigt omfattande och bryts därför ner i krav som sammanställts i en kravspecifikation, se Tabell 1.1. Kravspecifikationens funktion är att vara grund för verifikationen av projektet. Genom att verifiera kraven ges således ett mått på hur väl projektet uppfyller sitt syfte.

Huvudfunktionen för projektet, visa vägen till en plats, är uppdelad i delfunktioner som ska testas. Hur de ska testas och med vilken marginal på mätvärdet som krävs anges i Tabell 1.1. Övriga funktioner som också ska verifieras med en bestämd marginal är Säkerhet, Prestanda och Användargränssnitt. Resultatet av testerna redovisas i Kapitel 4.

Tabell 1.1: Kravspecifikationen sammanställer hur huvudfunktionens delfunktioner samt övriga funktioner ska testas och verifieras. Kriterierna anger vad som ska uppfyllas och målvärdet anger vilka mätvärden som testet ska ge för att det ska anses vara godkänt. Verifieringsmetoden anger hur testet ska utföras.

Kravspecifikation						
		Kriterier	Målvärde	Enhet	Verifieringsmetod	
1.	Funktion/er					
		Huvudfunktion	Visa vägen till en plats			
	1.1	Delfunktioner	Köra framåt/bakåt	Förflyttning minst 5 m utan att avvika från en rak linje med mer än ± 20 cm	cm	Test, visuell verifiering samt mätning
	1.2		Svänga	Svänga 90° utan att avika mer än ± 15°	°	Test och mätning av vinkel
	1.3		Hitta väg	Autonomt förplanera väg		Visuell verifiering av simulerad väg
	1.4		Förflytta till ett verkligt mål	Efter utförd förflyttning ska roboten vara ±50 cm från det verkliga målet	cm	Visuell verifiering och mätning
2.	Säkerhet					
	2.1		Upptäcka väggar	Identifiera och upptäcka alla väggar som är ett hinder för förflyttning		Avläsning av inputsignaler från sensorer vid test
	2.2		Upptäcka hinder (föremål, kanter)	Identifiera och upptäcka 80% av alla hinder ±80° från färdriktningen. Gäller hinder inom en radie om 50 cm		Avläsning av inputsignaler från sensorer vid test
	2.3		Undvika väggar	Undvika krock med samtliga väggar		Visuell verifiering och test
	2.4		Undvika hinder	Ska kunna undvika påkörning av 70% av upptäckta hinder		Visuell verifiering och test
3.	Prestanda					
	3.1		Uppdatering av position	Positionen ska uppdateras 3 gånger/sekund		Beräkning
4.	Användargränssnitt					
	4.1		Kommunikation	Roboten ska svara på användarens direktiv		Fysiskt test och kontroll att direktiv har utförts

1.4 Avgränsningar

För att nå goda resultat under en begränsad tid är det av stor vikt att begränsa arbetet. Nedan följer de begränsningar på robotens konstruktion och funktion som projektet utgår från.

- Robotens användning begränsas till inomhusmiljö då svårare terräng anses kräva för stor utveckling av den befintliga konstruktionen.
- Roboten kommer inte att vara anpassad för att gå i eller känna av trappor. Då det precis som föregående punkt kräver för stor utveckling av själva konstruktionen.
- Då projektet inte inkluderar byggnation av fysiska delar sätts inga krav på till exempel hastighet, vikt och storlek. Detta eftersom tidigare arbeten med roboten redan kravsatt och tagit fram lösningar för dessa faktorer.
- Det egenutvecklade användargränssnittet avgränsas till att styra roboten till förutbestämda koordinater. Det kommer således inte gå att klicka i en karta eller fjärrstyra roboten för att få den att köra till ett specifikt mål. Detta med anledning av att programmeringen anses vara för omfattande.
- Verifiering av gränssnittets användarvänlighet kommer inte att beaktas. Detta på grund av att huvudfokus i projektet ligger på implementering av ett navigeringssystem. Kommunikationen mellan människa och robot anses ligga inom annat område. Dock kommer användargränssnittets funktionalitet verifieras.

1.5 Relaterade arbeten

Arbete med autonoma fordon och robotar har pågått under lång tid. Som ett exempel landade det första kommersiella flygplanet med passagerare med hjälp av autopilot i London redan i juni 1965 [17]. Tekniken har utvecklats sedan dess och automation har idag ett brett användningsområde.

Liknande teknik och lösningar som undersöks och implementeras i detta projekt har i stor utsträckning använts för tidigare tillämpningar. Självgående gräsklippare är ett exempel på en autonom robot som på liknande sätt som roboten i det här projektet behöver hantera navigering för en säker användning. I en rapport om säker vägplanering för en robotgräsklippare skriver Biber, Schirmer och Stachniss att det är viktigt att roboten håller sig inom ett förutbestämt område [18]. Detta så att den till exempel inte trillar ner i en damm eller kör ut på gatan. En vanlig lösning på detta problem är att gräva ner en kabel längs tomtgräns och runt hinder. Roboten känner då av kabeln och håller sig på rätt sida om den. Det finns dock nackdelar med denna typ av teknik i form av användarvänlighet. Det är svårt för gemene man

att gräva ner en kabel runt hela tomtgränsen utifrån väldigt specifika instruktioner och om det skulle bli fel från början är det svårt att rätta till i efterhand. En lösning på problemet som de undersökte i sin rapport var att implementera en 2D laser och sedan navigera med hjälp av tekniken SLAM (Simultaneous Localization and Mapping) [18]. Denna teknik kommer presenteras mer i detalj i senare delar av rapporten.

Ett annat användningsområde för autonoma robotar är att transportera stora kvantiteter av material. AGV (Automated Guided Vehicles) användes från början till repetitiv transport av material inom tillverkningsindustrin. Idag används de även i samma syfte i andra miljöer så som i lager och containerterminaler [19]. Wu, Zhang, Zou, Zhao, Lou och Yin skriver i sin rapport om utvecklingen av koordinerade banstyrda AGVer hur de löst problemet med vägplanering. De löste det genom att med hjälp av en kamera följa en linje i marken så att fordonet alltid håller sig till den tänkta banan [20]. Att följa en målad linje på marken är dock ingen lösning för detta projekt då det hade gett permanent märkning av golvet i skolans lokaler, vilket inte är önskvärt.

Förutom utformningen av ett navigeringssystem för en autonom robot behandlar detta projekt ett sätt att öka teknikintresset hos barn. Enligt Ahmad, Mubin och Orlando är ett känt problem att upprätthålla intresset under en längre tid hos barn när de interagerar med robotar [21]. Deras studie genomfördes med tester där barn i olika testgrupper fick spela spelet Snakes & Ladders med en robot och sedan reagerade roboten på olika sätt i de olika testgrupperna. I den ena gruppen testades känslor och i den andra testades tidigare prestationer. I känslö-gruppen läste roboten av barnens ansiktsuttryck och svarade sedan baserat på om barnen såg glada eller ledsna ut. I den andra testgruppen svarade roboten med tidigare prestationer. Roboten memorerade bland annat barnens tidigare resultat och om barnet fick en stege sa roboten "Wow, this is great, you have a ladder. You are very lucky. I remember you had a ladder last time as well". Deras studie visade på att interaktionen som var resultatbaserad var den mest effektiva för att barnen skulle upprätthålla intresset.

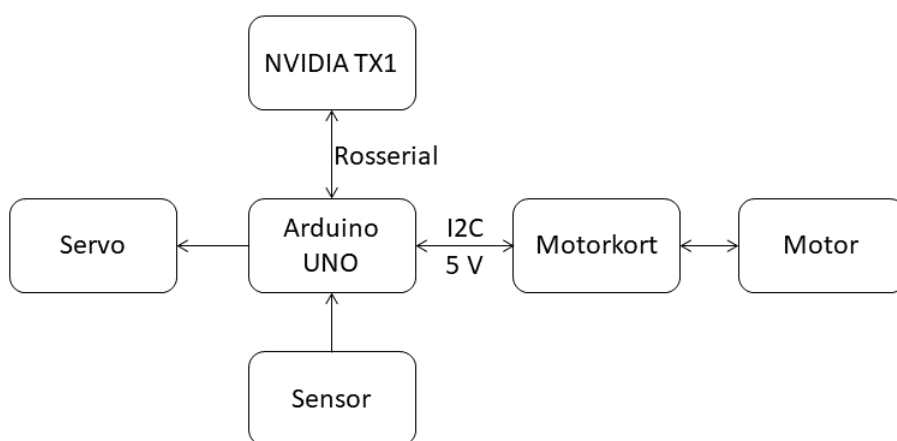
1.6 Rapportens struktur

Rapporten är uppdelad i sex kapitel vars uppgift är att på ett följsamt sätt få läsaren att förstå hur projekt genomförts. I Kapitel 2 beskrivs de komponenter som används i projektet, både de ursprungliga och de som lagts till. Även en systemöversikt över det utvecklade systemet presenteras. I nästkommande kapitel, Kapitel 3, förklaras det utvecklade systemet mer ingående. Där förklaras hur varje delsystem producerats, vilka val som gjorts och hur delsystemen fungerar tillsammans. Kapitel 3 följer kommunikationsvägen från användaren. Kapitlet börjar således med användargränssnittet och avslutar vid motorerna där användarens input övergår till den faktiska förflyttningen av roboten. Kapitel 4 redovisar verifiering av de krav som ställts upp och en diskussion gällande resultatet och själva arbetet ges i Kapitel 5. Slutligen avslutas rapporten med Kapitel 6 där det ges slutsatser och rekommendationer till vidare arbeten.

2. Teknisk översikt

I följande kapitel beskrivs projektets utgångsläge. Kapitlet innehåller en beskrivning av själva roboten, robotens drivning samt systemet som får komponenterna att samverka. Därefter redogörs teorin bakom de tillagda komponenterna, Bluetooth-modul, LiDAR och IMU. Bluetooth-modulen lades till i syfte att möjliggöra kommunikation mellan användare och robot och den specifika modellen valdes för att den var kompatibel med Arduino UNO. LiDARn och IMUn lades till i syfte att möjliggöra robotens navigering, dessa komponenter valdes för att de fanns tillgängliga för projektet och för att inte överskrida projektets budget.

Nedan visas en bild över det ursprungliga systemet i form av en systemöversikt, se Figur 2.1. I bilden visas vilka komponenter som fanns tillgängliga vid projektets start samt hur de var sammankopplade och hur de kommunicerade. Kapitlet avslutas med en systemöversikt över det befintliga systemet för att beskriva hur de olika delsystemen sitter ihop och kommunicerar. Figur 2.1 återfinns då i en uppdaterad version i slutet av kapitlet under Avsnitt 2.8, Systemöversikt, som visar hur systemet ser ut idag. Detta för att tydliggöra vilka tillägg och ändringar projektet har behandlat.



Figur 2.1: Systemöversikt över roboten från tidigare projekt. Systemöversikten visar hur de olika komponenterna kommunicerar och samverkar.

2.1 Meccanoid G15KS och bottenplattan

Roboten Meccanoid G15KS från företaget Meccano var sedan tidigare arbeten ihopbyggd och den liknar en karakteristisk humanoid robot, se Figur 2.2. Från tidigare arbeten satt en dator av typen Jetson TX1 från Nvidia på roboten. Däremot byttes denna ut på grund av det uppstod tekniska problem som glapp och plötsliga avstängningar. Det var även önskvärt att ha en datorskärm på roboten och på grund av detta användes istället en bärbar dator som kunde placeras på bottenplattan vid körning.

Robotens ursprungliga underrede var vid projektets start utbytt och den befintliga bottenplattan är ett resultat av tidigare arbeten, se [14] för mer information. Roboten har också försetts med batteridrift i tidigare arbeten. Bottenplattan fungerar både som förvaring till robotens olika komponenter och batterier samt som en stadig stomme för roboten att stå på.



Figur 2.2: *I bilden ses roboten Meccanoid G15KS som företaget Meccano utvecklat. Det är denna robot projektet bygger på.*

2.2 Drivning

För att styra robotens motorer används ett motorkort av typen MD25. Motorkortet är inkopplat i mikrokontrollen. Denna skickar värden till motorkortet via I2C, som är ett kommunikationssätt där mikrokontrollen kan kommunicera med flera moduler över samma databuss. Motorkortet matar sedan motorerna, EMG30, med rätt spänning. Motorkortet tar emot värden från 0 till 255 som anger hur motorerna ska rotera. Om värdet 0 skickas innebär det högsta hastighet bakåt, 128 gör att motorerna står stilla och 255 innebär att motorerna kör maximal hastighet framåt [22].

2.3 ROS

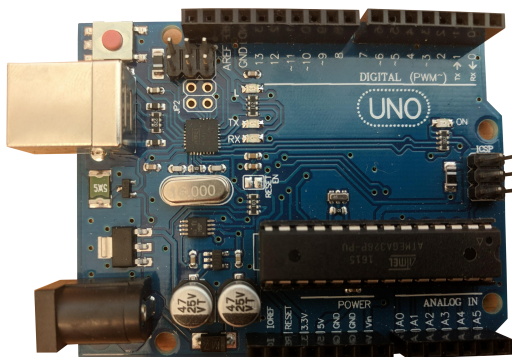
Eftersom roboten har ett flertal olika hård- och mjukvaror behövs ett system som hanterar själva kommunikationen mellan dem. ROS (Robot Operating System) används därför för att koppla ihop de olika komponenterna. ROS är ett open source-program [9], vilket innebär att det är öppet för allmänheten att använda sig av. Det finns flera fördelar med att använda open source. De främsta fördelarna är att det är gratis att använda och att det är lätt att hitta information och få hjälp med problem som andra användare har löst.

ROS är ett ”meta”-operativsystem, vilket innebär att det används på ett annat operativsystem. Det är uppbyggt av olika *noder* som fungerar som ett databearbetande program för de olika komponenterna. Noderna gör det möjligt att bygga upp ett system med kommunikationsvägar så att komponenterna kan påverka varandra [9]. Det finns olika typer av noder, till exempel *roserial* som kan sköta kommunikationen mellan arduino och huvuddatorn. Det finns även noder som sköter operationer i systemet och noder som visualiserar själva arbetet. En nod kan till exempel visualisera bilden från en inkopplad kamera.

Kommunikationen mellan de olika noderna i ROS sker bland annat via *topics*. Dessa kan beskrivas som meddelandekanaler där noderna kan publicera eller prenumerera på data [11]. En nod som publicerar data på en topic kallas *publisher* och en nod som prenumererar på data kallas *subscriber*. Subscribers kan använda informationen till sina egna funktioner. Denna kommunikation är dock asynkron, vilket betyder att de två olika noderna inte känner till varandra. Kommunikationen sker enbart mellan nod och topic. För synkron kommunikation används istället *services* där två bestämda noder har direktkontakt mellan varandra [11]. Utöver dessa kommunikationssätt kan data även sparas från en nod för att sedan spelas upp. Datan kan då återanvändas flera gånger och är statisk, den ändras alltså inte med tiden.

2.4 Arduino UNO

Mikrokontrollern, se Figur 2.3, som är av typen Arduino UNO är utvecklad av företaget Arduino. Arduinon har 14 pins som användas som in- eller utgångar till olika elektroniska komponenter [2]. Ström kopplas till mikrokontrollern via en extern ingång på kortet och kan sedan skickas vidare till andra inkopplade moduler via en 5V- eller 3V-utgång. Arduinon stödjer I2C-kommunikation. För att programmera mikrokontrollern används Arduino IDE som är företagets egna programmeringsmiljö. Programkoden laddas sedan över till Arduinon via en USB-B-port. Precis som ROS är Arduino IDE en open source-mjukvara [2].



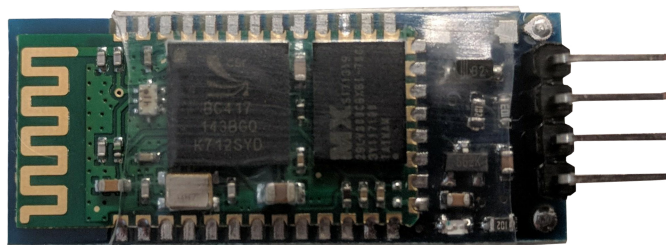
Figur 2.3: Mikrokontroller av typen Arduino UNO.

2.5 Bluetooth-modul

Den första tillagda komponenten är en Bluetooth-modul. Bluetooth är ett trådlöst kommunikationssätt som kan skicka samt erhålla information över 2.4 GHz-bandet. Det används främst för korta avstånd [23]. Ett Bluetooth-nätverk, ett *piconet*, är en samling av moduler som är ihopkopplade. Det består av en modul som är *master* och resterande moduler som fungerar som *slaves*, där mastern är den som styr systemet. Genom att begära data från, eller skicka data till, sina slaves koordinerar den kommunikationen i nätverket. En slave kan inte kommunicera med andra slaves och har som enda uppgift att lyssna till mastern [23].

Bluetooth-modulen i projektet är av typen HC-06 och används för att kommunicera mellan olika komponenter i systemet. Denna modul har 4 pins [24], vilket kan ses i Figur 2.4. Två av dessa används för att förse modulen med en ström- samt jordkälla. De två andra används för att skicka samt ta emot data. Detta sker via seriell kommunikation vilket innebär att det skickas en bit åt gången [25].

För att upprätta en koppling söker en Bluetooth-modul efter en annan modul, som lyssnar till förfrågningar. Den andra modulen svarar med sin adress. Denna adress används av den första modulen för att koppla ihop den med den andra. När de är kopplade kan data skickas och tas emot. Moduler kan även paras ihop för att senare automatiskt kopplas samman när de är tillräckligt nära varandra [23].



Figur 2.4: En Bluetooth-modul till en Arduino.

2.6 LiDAR

LiDAR (Light Detection And Ranging) är en avståndsmätare som med hjälp av studsande laserljus kan bestämma avstånd. I projektet används en LiDAR av modellen URG-04LX-UG01 från företaget Hokuyo, se Figur 2.5. Enligt Hokuyo är den energisnål och har en effektförbrukning på 2.5 W, [26]. Den kan mäta avstånd mellan 20 - 5600 mm med en felmarginal på ± 30 mm. Vinkelupplösningen är 0.352° vilket innebär att avståndet mellan varje laserstråle som LiDARn skickar ut är 0.352° . Den har även en vinkelbredd på 240° , vilket betyder att föremålet som den är monterad på måste rotera för att 360° ska kunna mätas.

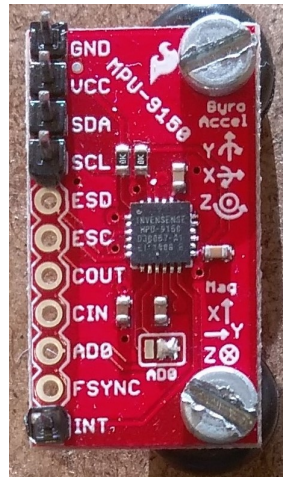


Figur 2.5: *LiDAR av modellen URG-04LX-UG01 från företaget Hokuyo.*

2.7 IMU

En IMU (Inertial Measurement Unit) används för att mäta rotation och acceleration runt olika axlar [27]. Med hjälp av gyroskop och accelerometer kan IMUn mäta hur den orienterar sig över tid genom att mäta förändringar. IMU har ett stort antal användningsområden inom navigering för både mark- och luftburna fordon. Det återfinns dessutom en IMU i de flesta mobiltelefoner. Den IMU som används i detta projekt är av modellen MPU-9150 av företaget TDK InvenSense, se Figur 2.6. Det är en IMU med nio frihetsgrader, vilket innebär att gyroskop, accelerometer och kompass kan mätas separat. Den kommunicerar med en I2C-databuss och behöver en inspanning på 3.3V [27].

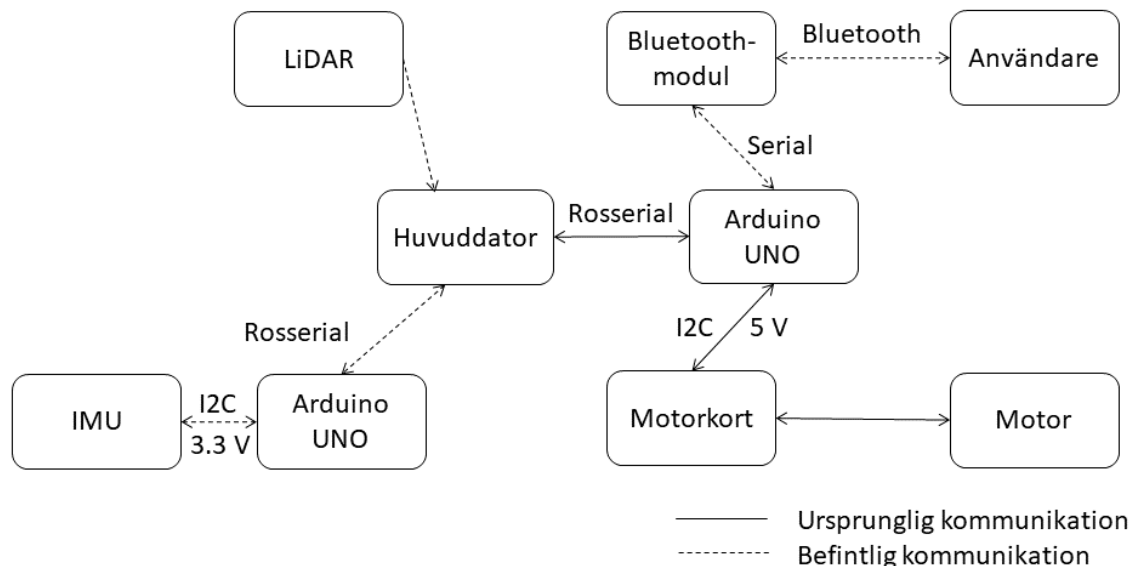
Om IMUn används under långa perioder kan små fel vid mätningarna ackumuleras och visar då att den befinner sig på en annan plats än vad den egentligen gör, detta fenomen kallas för drift. På grund av detta brukar oftast en IMU användas tillsammans med andra system för att mäta position och acceleration.



Figur 2.6: IMU av modellen MPU-9150 från företaget TDK InvenSense.

2.8 Systemöversikt

Robotens olika komponenter är nu presenterade och komponenternas samverkan sammanställs i det här avsnittet i form av en systemöversikt, se Figur 2.7. Som nämnts tidigare är figuren den uppdaterade systemöversikten av det ursprungliga systemet. Skillnaderna från den tidigare systemöversikten, se Figur 2.1 är att servo, sensorerna och NVIDIA TX1 är borttagna och IMU, LiDAR, Arduino UNO, Bluetooth-modul och en Användare är tillagt. Motivering och tillvägagångssätt av förändringarna kommer presenteras i nästkommande kapitel.



Figur 2.7: Uppdaterad systemöversikt. Heldragen linje innebär att kommunikationen är oförändrad från tidigare projekt och streckad linje innebär att en helt ny kommunikationsväg är upprättad.

Det befintliga systemet är uppdelat i de fyra delsystem som följer nedan:

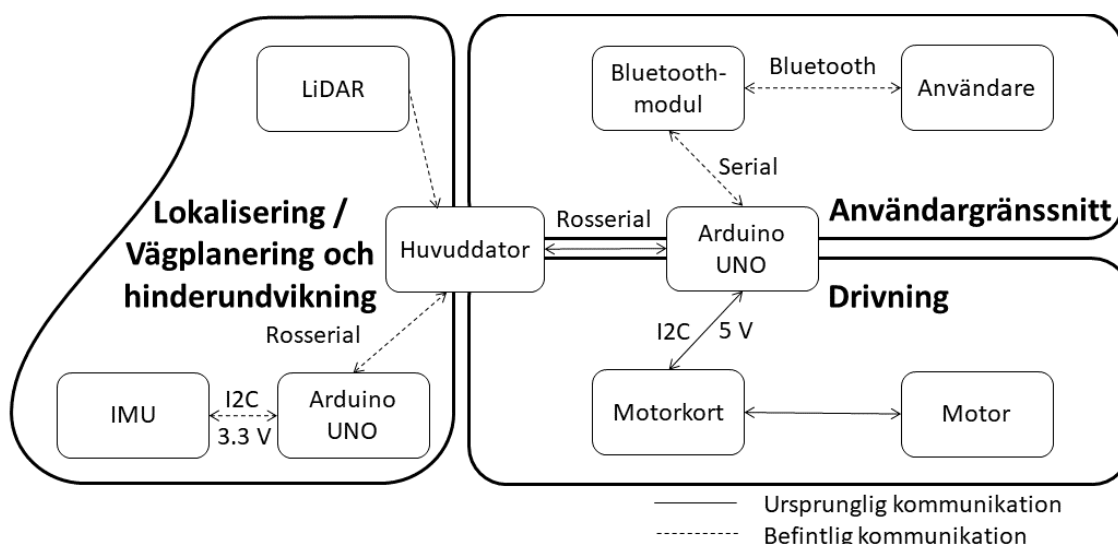
- Användargränssnitt
- Lokalisering
- Vägplanering och hinderundvikning
- Drivning

Dessa fyra delsystem är baserade på frågeställningen som formulerades i inledningen av rapporten. Varje delsystem innefattas av en eller flera av frågorna. Användargränssnittet behandlar hur användaren och roboten ska kommunicera med varandra. Lokaliseringen är för att roboten ska kunna avgöra sin nuvarande position. Vägplanering och hinderundvikning är hur roboten ska kunna upptäcka och undvika hinder. Drivningen syftar till förflyttning av roboten, men då hårdvaran är implementerad och utvecklad i tidigare arbeten läggs enbart fokus på utveckling av mjukvaran.

3. Implementering av delsystem

Som nämnts tidigare består det utvecklade systemet av fyra delsystem. De kommer i detta kapitel beskrivas mer ingående där fokus ligger på hur de olika delsystemen är valda, utvecklade och hur de kommunicerar med varandra. Kapitlets olika avsnitt följer indatan från användaren till robotens faktiska förflyttning. Delsystemen förklaras således utgående ifrån den ordning som systemet behandlar användarens indata. Därav är kapitlets första del användargränssnitt, och det sista beskriver hur kommunikationen till motorerna sker och hur roboten då förflyttar sig. Kapitlet avslutas med en mer ingående beskrivning av hur kommunikationen mellan de olika delsystemen sker.

I Figur 3.1 visas vilka komponenter som används i vilket delsystem. Användargränssnittet behandlar endast kommunikation mellan användare och huvuddator. De två delsystemen Lokalisering samt Vägplanering och hinderundvikning arbetar båda två med LiDAR, Arduino, IMU och huvuddator och är därför presenterade inom samma område i figuren.



Figur 3.1: I bilden återfinns systemöversikten och de inringade delarna visar vilket delsystem som arbetar med vilka komponenter.

3.1 Användargränssnitt

En del av projektet var att utveckla ett funktionellt gränssnitt för kommunikation mellan robot och människa. Det här avsnittet beskriver hur valet av användargränssnitt gjordes, hur användargränssnittet utformades och varför.

3.1.1 Val av användargränssnitt

Det ursprungliga användargränssnittet var svåränvänt för nya användare, därför bestämdes att ett nytt användargränssnitt skulle införas. Kommunikationssättet som användes vid projektets start var att de kommandon som roboten skulle följa lades till i programkoden. Den uppdaterade koden skickades sedan till roboten. Detta gjorde att det blev svårt för en utomstående att ge roboten kommandon eftersom den behövde sätta sig in i hela programkoden. Kodskrivning kräver dessutom programmeringskunskaper, vilket inte är en självklarhet att användaren har. Speciellt inte då projektets önskan är att det ska vara lättänvänt för barn.

Av de fyra kommunikationssätten som var aktuella att implementera: textbaserat gränssnitt, röststyrning, hemsida och en applikation, valdes en applikation. Detta valet grundades på följande:

- Det ursprungliga systemet använde ett textbaserat gränssnitt, vilket inte upplevdes vara intuitivt. Användaren måste dessutom veta exakta kommandon samt veta var i koden dessa ska placeras.
- Röststyrning valdes bort på grund av att det ansågs vara en för stor uppgift för det här projektet. För röststyrning behöver även specifika kommandon memoreras av användaren.
- En hemsida behöver tre steg för att starta. Internet ska öppnas, hemsidans namn behöver skrivas in i webbläsaren och sedan ska hemsidan laddas. Dessa tre steg anses vara omständiga. Eftersom roboten är mobil är det även viktigt med ett väl anpassat användargränssnitt för mobila enheter. En hemsida skulle därför behöva anpassas till både datorer och mobila enheter.

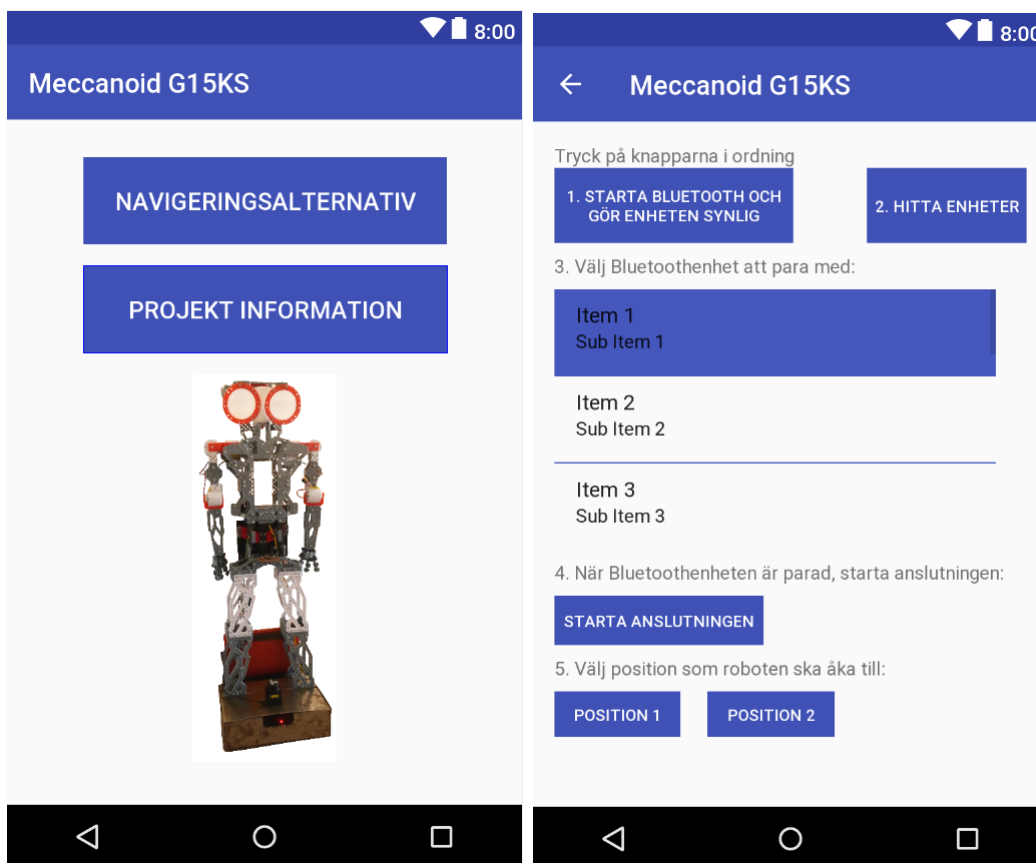
En applikation kan göras enkel och intuitiv att använda med grafiska verktyg. Kommandon kan byggas in i tryckknappar och en redan nedladdad applikation kräver enbart ett steg för att startas. Eftersom en applikations syfte är att visas i mobila enheter passar en applikation in i kriterierna att vara anpassad för mobila enheter. För det här projektet är det också viktigt att någon form av trådlös kommunikation kan upprättas då en användare ska kunna finna sig bredvid roboten och på ett enkelt sätt kunna skicka kommandon till den, vilket fungerar väl för en applikation.

Applikationer kan programmeras på många olika sätt och i många olika miljöer. Då projektet hade begränsad tid och det inte fanns något kompatibelt program med både Android och iOS, valdes att enbart programmera för ett av operativsystemen. Applikationen valdes att utvecklas enbart för Androidenheter. Detta för att lanseringen av en iOS-applikation, till skillnad från en Androidapplikation, behöver gå via appstore vilket kostar pengar. Utvecklingen av applikationen valdes att göras i Android Studio eftersom det är det ledande programmet för utveckling av Androidapplikationer. Programmet kan användas för avancerade applikationer och det uppdateras kontinuerligt [28].

3.1.2 Applikationens uppbyggnad

Som nämndes i Avsnitt 1.4, Avgränsningar, begränsas styrningen av roboten till enbart förutbestämda koordinater. För att minska risken för att felvärden skickas till roboten valdes därför att all styrning av roboten ska ske genom tryckknappar och andra grafiska verktyg.

Förhoppningen med arbetet var att väcka teknikintresset hos barn, därför valdes att göra applikationen så tydlig som möjligt med målet att användaren enkelt skulle kunna styra roboten. Applikationens hemskärm gjordes med två valmöjligheter, se vänstra delen av Figur 3.2. Till höger i figuren visas skärmen för navigeringsalternativ, den har byggts upp så att alla steg är synliga. Till exempel aktiveras Bluetooth manuellt i applikationen och användaren måste aktivt välja Bluetooth-enhet att para med. En anslutning måste också aktivt startas. På detta sätt får användaren förståelse för det bakomliggande systemet. I applikationen finns även en knapp som användaren kan klicka på för att läsa information om projektet.



Figur 3.2: Den vänstra bilden illustrerar applikationens hemskärm och den högra visar skärmen då användaren tryckt på "Navigeringsalternativ". Användaren ställs då inför 5 steg för att upprätthålla en Bluetooth-kommunikation och välja mål.

3.2 Lokalisering

För autonom styrning av en robot behöver den kunna lokalisera sin egen position i ett rum. Detta kapitel behandlar vilket lokaliseringssystem som implementerats, vilka algoritmer som används för lokalisering samt varför dessa valdes framför andra. En djupare förklaring av algoritmerna som används ges även.

3.2.1 Val av lokaliseringssystem

Det finns många olika system som löser lokaliseringsproblem. GPS (Global Positioning System) är ett av de vanligast förekommande. GPS använder sig av satelliter för lokalisering vilket lämpar sig bäst utomhus då signalen försvagas av hinder så som väggar och föremål [29]. För inomhuslokalisering används därför IPS (Indoor Positioning System) vilket är ett samlingsnamn för lokalisering inomhus. IPS kan liknas vid GPS men istället för satelliter kan bland annat sensorer användas för att mäta avstånd. Mätdata kan sedan användas för lokalisering [30].

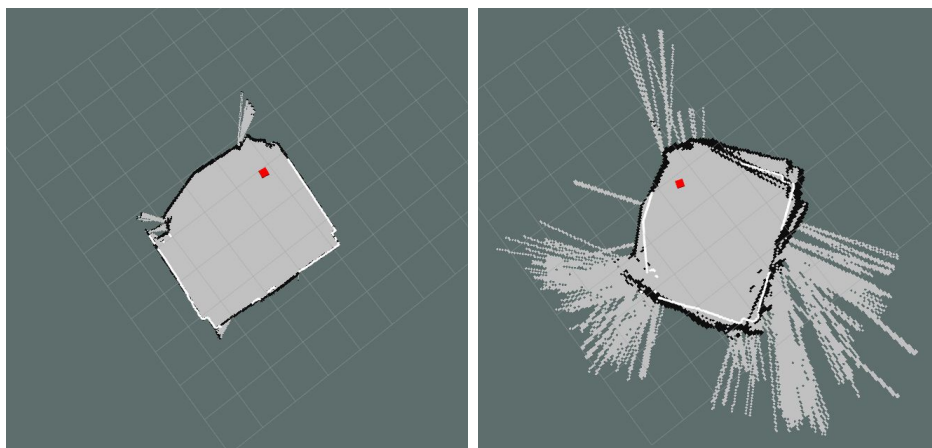
En lösning till lokaliseringsproblemet med IPS är att montera stationära sensorer, så kallade *beacons*, i rummet som beräknar position genom trilateration [31]. Trilateration innebär att tre sensorer skickar ut en signal till en mottagare och punkten där de tre signalerna möts är platsen som mottagaren befinner sig på, denna position kan sedan beräknas [32].

En annan lösning är användning av SLAM (Simultaneous Localization And Mapping). SLAM är idag en välkänd teknik som implementeras i robotar som arbetar i allt ifrån inomhusmiljöer till miljöer djupt under havsytan [33]. Med SLAM kan en robot placeras på en sedan tidigare okänd plats och stegvis rita upp en virtuell karta samtidigt som roboten vet var på kartan den själv befinner sig [33].

SLAM-lokalisering jämfördes med ett sensorbaserat system från företaget Marvelmind Robotics, där SLAM valdes främst för systemets flexibla användning. Lösningen från Marvelmind Robotics hade stor uppsättningsfas och begränsades till ett enda område [31]. Därför valdes SLAM framför Marvelminds system då det kan användas i nya områden utan att montering av beacons behöver göras. En annan fördel med SLAM är att det kan använda sig av en LiDAR, vilket är en komponent som fanns tillgänglig och Marvelminds system hade behövt köpas in vilket hade medfört en högre kostnad.

I detta projekt valdes att dela upp lokaliseringssystemet i två steg. Ett steg som ritar upp en karta och ett som lokaliserar roboten i den uppritade kartan. Detta gör det möjligt att hårdkoda koordinater som användaren senare kan välja att navigera roboten till. Detta innebär även att två olika koordinatsystem introduceras. Nämligen det globala och det lokala koordinatsystemet. Det globala syftar till kartans koordinatsystem medan det lokala är robotens egna koordinatsystem.

Den specifika kartuppritningstekniken som användes var Hector-SLAM. Valet stod mellan ROS-paketen Hector-SLAM och gMapping. Båda paketen gör det möjligt att genom att köra runt roboten längs kanterna i ett rum rita upp en noggrann karta. Kartan kan skapas genom att robotens reella position kontinuerligt loggas och sätts i relation till de uppmätta avstånden till andra föremål. I ett test av de två kartuppritningsteknikerna filtrerade Hector-SLAM bort felvärden i en betydligt högre grad än vad gMapping gjorde, se Figur 3.3. Valet blev därför Hector-SLAM.



Figur 3.3: Den vänstra bilden illustrerar den visuella kartan skapad med Hector-SLAM. Högra figuren illustrerar samma karta men istället uppritad med gMapping. De vita linjerna som är långt utanför de svarta konturerna kring det ljusare området på bilden till höger är felvärden. Vänstra bilden har något felvärde men strecken utanför konturerna beror främst på små luckor i testrummets väggar. Bilden gör det tydligt att Hector-SLAM bättre filtrerar bort felvärden än vad gMapping gör.

Algoritmen AMCL valdes för lokalisering i den karta som Hector-SLAM ritat upp. Valet gjordes för att AMCL finns inkluderat i ett befintligt paket i ROS som är enkelt att implementera. En nackdel är att både AMCL och Hector-SLAM kan ha svårt att lokalisera sig i en korridor där omgivningen ser i princip likadan ut för LiDARn [34]. Mer detaljerad förklaring om AMCL finns i Avsnitt 3.2.2.

3.2.2 Algoritmer

Nedan presenteras först den algoritm som Hector-SLAM använder för kartuppritningen, EKF (Extended Kalman Filter), och därefter förklaras algoritmen för robotens lokalisering mer i detalj.

EKF

Det finns tre olika metoder inom SLAM som vanligen används. Nämligen EKF, *Graph-Based Optimization Techniques* och *Particle Methods* [34]. Där Hector-SLAM använder den förstnämnda. EKF estimerar både robotens och andra föremåls positioner i en karta med hjälp av en kovariansmatris [34]. Kovariansmatrisen är en $n \times n$ matris som innehåller det statistiska värdet på sambandet mellan två slumpade

(stokastiska) variabler [35]. Medan roboten kör runt i rummet och gör mätningar uppdateras både vektorn med robotens position och kovariansmatrisen med hjälp av EKF.

EKF-algoritmen estimerar robotens position med Gaussian-fördelning (som kan liknas vid normalfördelning) som representeras av ekvation 3.1 [34].

$$p(x_t, m | Z_T, U_T) \sim N(\mu_t, \Sigma_t) \quad (3.1)$$

$$Z_T = \{z_1, z_2, z_3, \dots, z_T\}$$

$$U_T = \{u_1, u_2, u_3, \dots, u_T\}$$

Där x_t är robotens position och m är positioner av föremål. Således bildar m själva kartan. Z_T är robotens mätning av området som görs med hjälp av någon typ av sensor. Om det antas att det görs en mätning för varje tidssteg $t_1, t_2, \dots, t_T$ är Z_T således robotens samlade mätningar vid varje tidssteg. U_T är robotens relativa rörelse som fås genom avståndsmätning mellan två positioner som kommer i följd av varandra. Alltså en avståndsmätning från position 1 vid tiden $t - 1$ till position 2 vid tiden t . Detta kan till exempel göras genom avläsning av motorernas vinkel-pulsgivare. Datan i Z_T och U_T är således kända och givet denna data fås robotens position, x_t , och kartan, m . Index t innebär att de är beroende av tiden och T är en specifik tid, till exempel sluttiden när navigeringen är avslutad. Det antas oftast att m är oberoende av tiden då föremål som väggar och liknade inte vanligtvis flyttas med tiden.

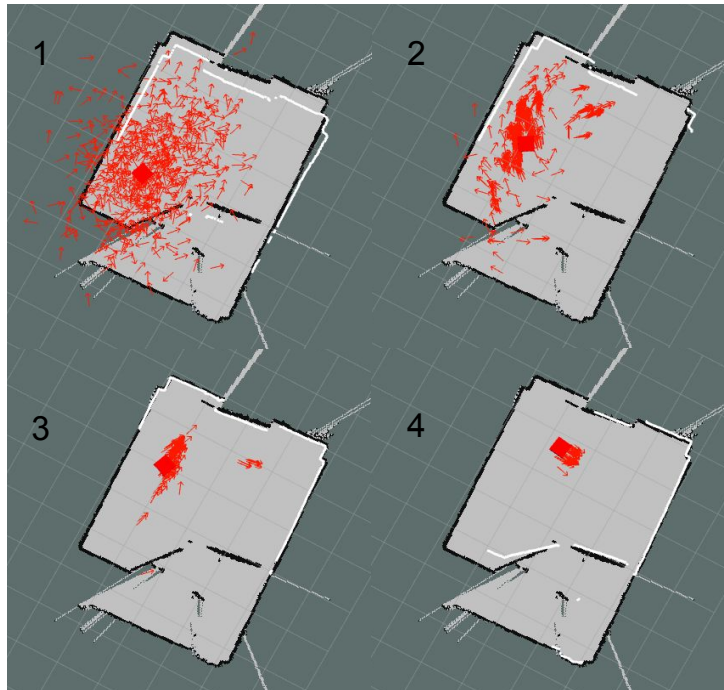
Den första delen av formel 3.1, $p(x_t, m | Z_T, U_T)$ är mer känd som Bayes sats $p(A|B)$ som anger sannolikheten för A givet B [36]. SLAM-problemet kan således liknas vid Bayes sats och problemet estimeras med Gaussian-fördelning $N(\mu_t, \Sigma_t)$ med EKF-algoritmen. Vektorn μ_t innehåller den bästa möjliga estimeringen av robotens egna- och föremåls position. Matrisen Σ_t är kovariansen av robotens bedömning av det troliga felet av den bästa estimeringen i μ_t . EKF estimerar således position av robot och karta och uppskattar med sannolikhetslära hur väl de stämmer överens med idealvärden.

AMCL

Den algoritm som används för att robotenheten ska kunna lokalisera sig i den globala kartan är AMCL (Adaptive Monte Carlo Localization). Algoritmen används på en sparad karta och baseras på sannolikhet med hjälp av ett partikelfilter [37]. Algoritmen bygger på att rummet inte förändras över tid eftersom partiklarna utgår från en tidigare uppritad karta.

I initialtillståndet finns partiklar uniformt distribuerat i kartan då enheten skulle kunna befinna sig på vilken position som helst. Sannolikheten är jämt fördelad mellan alla möjliga positioner. När enheten förflyttar sig i rummet längs den tänkta vägen samplas sedan partiklarna tills de kommit tillräckligt nära den exakta positionen. De partiklar med felaktig approximation tas bort när de inte längre är

sannolika. Möjlig orientering och riktning visualiseras med röda pilar i kartan, vilka till en början är fler. Orientering och riktning på pilarna är då mer osäker. När roboten sedan förflyttas blir det färre pilar och orientering samt riktning av pilarna blir mer sannolik och korrekt, se Figur 3.4.



Figur 3.4: Figuren illustrerar hur AMCL fungerar i en visuell karta. I bild 1 ses hur pilarna är utspridda över en stor area då roboten har en dålig estimering av dess position i rummet. Approximeringen blir bättre desto mer roboten förflyttar sig i området, se bild 2, 3 och 4.

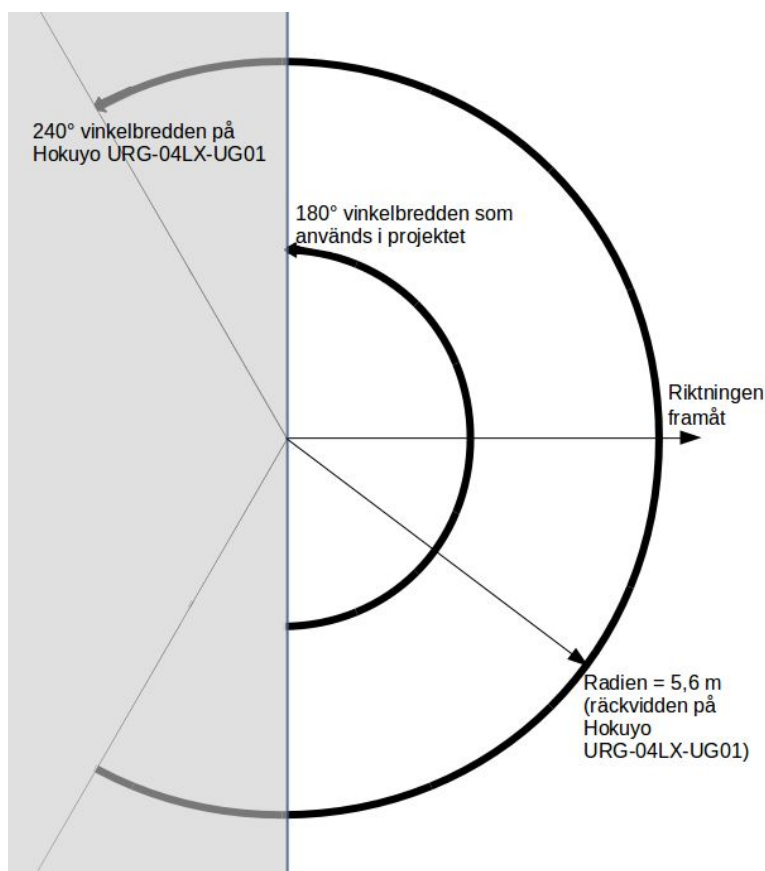
3.2.3 Implementering av SLAM

I Avsnitt 3.2.1 valdes Hector-Slam som metod för att rita upp kartan och AMCL som algoritm för mätning och lokalisering i den uppritade kartan. I det här avsnittet beskrivs steg för steg hur implementeringen och användningen av systemen fungerar.

Kartan som ritas upp med Hector-SLAM fås genom att roboten manuellt körs runt längs väggarna i rummet. Genom att estimerar hur kartan borde fortsätta och jämföra denna med de faktiska värdena från LiDARN bekräftas rummets dimensioner och sparas ner som en fil som återanvänds vid navigering i samma rum. Robotens LiDAR skickar ut laserstrålar längs det horisontella planet i samma höjd som LiDARN är monterad, på bottenplattan 20 cm ovanför markhöjd. Placeringen gjordes för att LiDARN ska se en majoritet av alla hinder. LiDARN läser avstånden till alla föremål i dess omgivning och skickar dessa till Hector-SLAM som hanterar alla mätvärden och omvandlar mätdatan till en rutnätad karta. Med detta menas en karta där varje punkt i kartan får ett koordinatvärde i x- och y-led. Förutom koordinatvärden får även punkten ett värde som sätts olika beroende på ifall punkten är ockuperad av hinder, fri från hinder eller ännu okänd, det vill säga ännu inte avskannad. Om Li-

DARn läser av ett värde som är kortare än maxavståndet på kameran sätts punkten in som ett hinder i kartan, medan alla punkter fram till samma punkt blir markerade som fria punkter. Ifall kameran får sitt maxvärde avläst blir alla punkter fram till maxavståndet fria punkter, medan de längre bort blir markerade som okända punkter.

I Figur 3.5 visualiseras hur LiDARn används i projektet. Den faktiska produkten har en vinkelbredd på 240° men då den monterades på bottenplattan till roboten behövdes vinkelbredden minskas till 180° för att undvika att robotens ben skulle behandlas som hinder. Därav förändrades även LiDARns mittpunkt, riktningen rakt fram, till 90° istället för 120° .



Figur 3.5: Grafisk förklaring över LiDARn som används. Den har en räckvidd på 5,6 m, total vinkelbredd på 240° varav 180° används i projektet. Mittpunkten som är i riktningen rakt fram är för LiDARn på 120° men i projektet är den korrigerad till 90° .

Vid navigering i befintlig karta använts AMCL för att bestämma robotens position på kartan. Av AMCL läses kartan som Hector-SLAM tidigare skapat, filen innehåller data som AMCL använder tillsammans med avståndsdaten som LiDARn kontinuerligt samlar in för att avgöra positionen. Denna position förbättras om roboten rör sig runt i kartan. Positionen och kartan kan sedan användas för vägplanering och hinderundvikning.

3.3 Vägplanering och hinderundvikning

I detta avsnitt beskrivs val av vägplaneringsalgorithm, hur implementeringen av olika funktioner har gjort det möjligt för roboten att planera en väg till ett mål och samtidigt undvika hinder på vägen dit. Det ges även en mer djupgående förklaring av algoritmerna som används i projektet.

3.3.1 Val av vägplanering- och hinderundvikningsalgoritmer

Vägplaneringsalgoritmer kan till viss del användas för hinderundvikning eftersom de filtrerar bort uppenbart omöjliga vägar. Ett objekt i ett känt område skulle i vissa fall kunna byta position och hindra roboten från att köra den planerade vägen. Därför har detta problem delats upp i två delar, en vägplaneringsdel och en hinderundvikande del.

Från ett urval av vägplaneringsalgoritmer är *Dijkstras algoritm* en vedertaget använd vägplaneringsalgorithm. Algoritmen hittar den billigaste vägen till en given destination genom att vikta alla olika delsträckor till destinationen utifrån avstånd. Summan av delsträckorna ger den totala kostnaden för förflyttningen [38]. En annan algorithm för vägplanering är *Carrot-planner*. Algoritmen arbetar från mål till start genom att placera ut en "morot" vid given destination och letar sedan efter möjliga vägar tillbaka [39]. Dijkstras algorithm valdes för detta projekt på grund av att algoritmen är väletablerad och beprövad, betydligt mindre information och resultat fanns att tillgå om Carrot-planner. Testning gjordes med Dijkstras algorithm och testerna visade rimliga resultat. Därför fortsatte projektet med Dijkstras algorithm.

Lösningarna som undersöktes för hinderundvikning var *DWAplanner* och *Trajectory rollout planner*. Båda algoritmerna fanns som nedladdningsbara paket från ROS-biblioteket. Innehållsmässigt är dessa paket väldigt lika, det som skiljer dem åt är hur de hanterar accelerationer [40] [41]. Båda algoritmerna testades på samma hinderbana för att med säkerhet veta vilken som snabbast når sitt mål. Efter tester valdes DWAplanner som tog 38 sekunder i jämförelse med Trajectory Rollout planner som tog 3 minuter och 58 sekunder.

3.3.2 Algoritmer

Nedan följer en mer djupgående beskrivning av algoritmerna som användes för vägplanering och hinderundvikning. Dijkstras algorithm användes för vägplanering och DWA användes för hinderundvikning.

Dijkstras algoritm

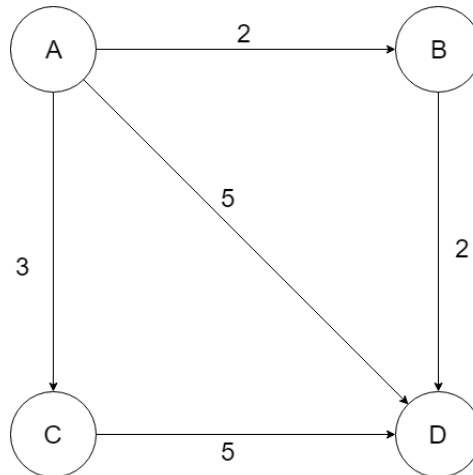
Dijkstras algoritm förtydligas med pseudokoden i Algorithm 1 [42]. Till en början initieras startpunkten och två vektorer. Den första innehåller avstånd till alla platser och den andra säger att alla platser är obesökta. I avståndsvektorn initieras avståndet till alla platser till oändligheten för att det sedan ska kunna uppdateras med korrekt data.

Från startpunkten söker algoritmen alla möjliga vägar till respektive plats i rummet och sätter en viktning på platserna baserat på avståndet till dem. När en plats viktning är bestämd skrivs värdet som tidigare var oändligheten över till den bestämda viktningen. Platsen flyttas även från den obesökta vektorn till en ny, besökta platser-vektor. Processen går sedan vidare tills dess att algoritmen har gått igenom alla möjliga platser. Då har algoritmen information om hur långt det är mellan alla platser i kartan och det är då algoritmen bestämmer väg från start till mål eftersom det är först då algoritmen kan veta vilken väg som är den kortaste.

Algorithm 1 Dijkstra(Graph,source)

```
for vertex v in Graph do
    dist[v] =  $\infty$ 
    previous[v] = None
end for
Ensure: dist[source]= 0
Ensure: Q=the set of all nodes in Graph
while Q is not empty do
    u=node in Q with smallest dist[]
    remove u from Q
    for each neighbor v of u do
        alt=dist[u]+dist_between(u,v)
        if alt < dist[v] then
            dist[v]=alt
            previous[v]=0
        end if
    end for
end while
```

Ett annat exempel på hur Dijkstras algoritm fungerar återfinns i Figur 3.6. Kartan har fyra platser, A, B, C och D. Dess viktade värde är siffrorna vid varje sträcka. Kartan är även riktad då de olika vägarna är representerade av pilar. I figuren går det att se att den rakaste vägen inte alltid är den billigaste då totalsumman blir mindre om det väljs att först gå via B för att komma från A till D.



Figur 3.6: Ett illustrativt exempel av en graf där Dijkstras Algoritm kan appliceras. Den billigaste vägen från A till D går genom B. Svaret anger en summa av $2+2=4$. Alla andra vägar från start till mål ger en högre summa.

DWAplanner

Den hinderundvikningsalgoritm som används stödjer vektorer i ett känt 2D-plan med variabler x , y och θ [40]. DWA (Dynamic Window Approach) är en algoritm som hanterar hastigheter och accelerationer som måste begränsas. Detta betyder att när roboten når en farlig zon sänks hastigheten och en väg runt hindret söks. En zon räknas som farlig när robotens position befinner sig nära ett objekt eller en vägg. I den visuella kartan beskrivs den farliga zonen med svart färg, se Figur 3.4. DWA är specialiserad på att hantera dessa begränsningar för att avgöra vilken alternativ väg en robot kan ta. Först söker den efter vilka fria vägar som är möjliga och sedan väljer roboten den säkraste vägen, en väg fri från hinder. Denna sökning har ett kort tidsintervall för att den inte ska stå stilla för länge. Målet är att nå en destination med maximal säkerhet, det vill säga undvika all kollision.

3.3.3 Implementering av vägplanering

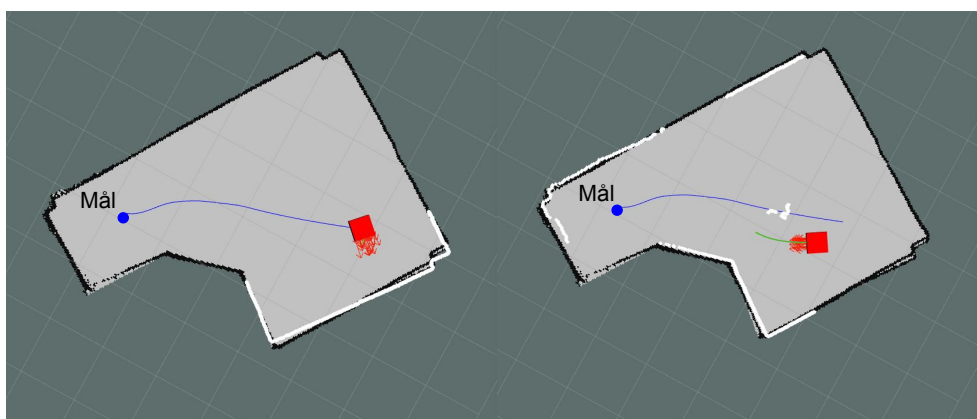
Dijkstras algoritm och DWAplanner använder tre viktiga komponenter som måste fungera för att få navigeringssystemet att fungera. Dessa tre består av den globala- och lokala karta som nämndes tidigare. Den globala kartan beskriver hela rummet med alla hinder definierade i koordinater. Det är denna karta som ritas upp med hjälp av Hector-SLAM. Den lokala kartan är en karta som uppdateras kontinuerligt med hjälp av lasern från LiDARn för att uppdatera hinder samt möjlig väg. För att kunna hantera beräkningen skapas en kostnadskarta som sammanställer avstånd från den globala och lokala kartan. Kostnadskartan har som uppgift att låta Dijkstras algoritm beräkna vägen med lägst kostnad. Dessa komponenter möjliggör tillsammans en navigation till målet.

Vägplaneringens fokus var att hitta billigaste väg från start till mål och samtidigt undvika alla hinder. De hinder som skulle undvikas delades upp i två olika kategorier. I den första kategorin fanns fasta hinder, vilket exempelvis var väggar, och i den

andra fanns rörliga hinder, exempelvis människor. På grund av detta valdes att ha en uppdelad implementering av vägplanering och hinderundvikning. Navigeringen skedde därav i två steg. Det första steget var global planering, där Dijkstras beräknade billigaste väg från startposition till slutposition. Det andra steget var lokal planering, där AMCL räknade ut hur nya hinder som uppkom längs vägen skulle undvikas.

För vägplaneringsdelen av navigeringen används en redan uppritad karta över ett område. Då planeringen skedde med hjälp av detta redan kända område kunde Dijkstras-algoritm användas. När vägplaneringen var färdig började roboten åka mot målet. Förutom den sparade kartan fick roboten även in information om omgivningen, som kontinuerligt sparas i den lokala kartan, med hjälp av LiDARn. Om LiDARn fick information om att omgivningen skiljde sig från den sparade kartan antogs detta vara ett nytt hinder som dykt upp. Med hjälp av DWApplanner räknades då en alternativ väg ut till målet. Eftersom DWApplanner används vid mer oberäknliga hinderundvikningar fick denna en högre prioritet än Dijkstras algoritm som primärt bestämde vägen.

I Figur 3.7 visas hur de olika vägplaneringsalgoritmerna fungerar tillsammans. Den vänstra bilden visualiserar den väg Dijkstra's algoritm valt när den fått en position i kartan. Den högra bilden visar hur den lokala vägplaneraren, DWApplanner, tar över den globala vägplaneringen för att undvika det nyupptäckta hindret. För att undvika detta hinder var roboten tvungen att räkna ut hur den skulle ta sig runt hindret, denna planering indikerades med den gröna linjen i figuren.



Figur 3.7: I den vänstra bilden visas hur global vägplanering har räknat ut bästa väg till slutpositionen, i den högra bilden ses hur den lokala vägplaneringen tar över då ett oväntat hinder uppkommer.

3.4 Kommunikation mellan delsystem

I detta avsnitt kommer det tas upp hur de olika delsystemen kommunicerar med varandra och hur kommunikationen mellan hårdvara och mjukvara fungerar.

3.4.1 Kommunikation - Applikation

Applikationen kommunicerar via Bluetooth. Androidtelefonen agerar som en *master* och kopplas samman med Bluetooth-modulen HC-06, ett *slave*, och de bildar tillsammans en *Piconet*. I applikationen finns två positioner hårdkodade, dessa finns som knappar i applikationen; Position 1 och Position 2. Vid knapptryck på någon av dessa skickas ett meddelande från applikationen till Bluetooth-modulen. Arduinon, som modulen är kopplad till, läser in detta meddelande och publicerar en hårdkodad position till topicen *move_base_simple/goal* (se Bilaga A).

3.4.2 Kommunikation - SLAM och Vägplanering

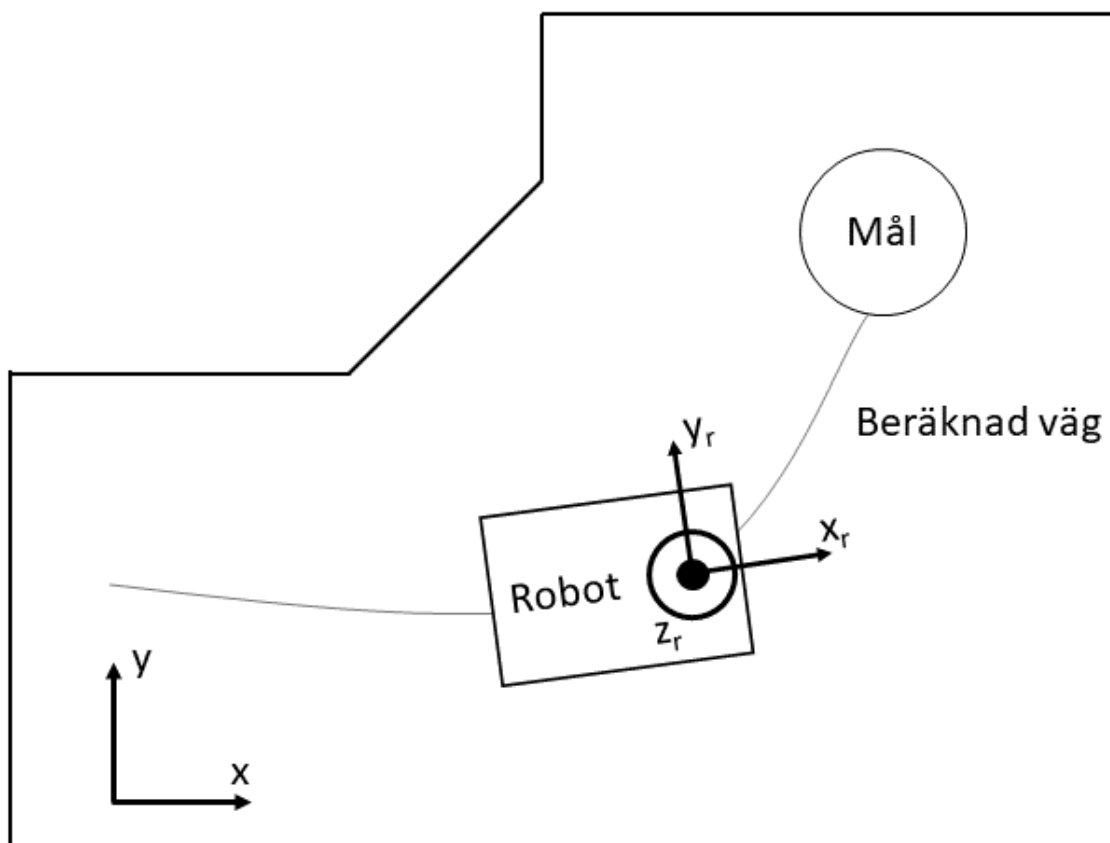
I Bilaga A och B visas hur kopplingarna i ROS är gjorda för att nå det önskade målet. Datan från LiDARn avläses i en nod vid namnet */urg_camera* och denna data hanteras sedan av Hector-SLAM som finns i ROS under namnet */Hector_node*. För att kontrollera datan från LiDARn finns ytterligare en nod, */laser_scan_mather_node*, mellan dessa som kontrollerar att sannolikheten av matchningen mellan LiDARn och Hector-SLAM är tillräckligt stor för att utskrivning till kartan ska ske. För att kunna visualisera robotens karta i verktyget *RViz* inom ROS skapas en ny nod som sköter denna funktion, namnet på noden är */robot_state_publisher*. Därefter kan Hector-SLAM prenumerera på topics med dessa källor och i sin tur rita ut en karta.

Den uppritade kartan benämns som den globala kartan, den lokala kartan i sin tur är kopplad till LiDARns nod */urg_camera*. Båda dessa noder är kopplade till AMCL. Med hjälp av dessa kopplingar kan roboten visualisera en karta över rummet och samtidigt lokalisera sig i miljön. All denna data läses in av noden */move_base*. Denna nod läser även in användarens önskade koordinater via Bluetooth-modulen. På */move_base*-noden samlas all data och det är där som den billigaste vägen beräknas. Denna nod skickar ett *geometry_msgs/Twist* meddelande till topicen *cmd_vel*. Detta meddelande består av två vektorer för linjär- och vinkelhastighetsvärden enligt Tabell 3.1 [43]. Linjärvektorn är riktning och vinkelvektorn är rotation kring respektive axel. Denna information skickas till motorerna, se Bilaga A och B för en tydlig graf.

Tabell 3.1: Information om hur ett *geometry_msgs/twist*-meddelande är uppbyggt.

Linjärvektor	Vinkelvektor
x_L	x_V
y_L	y_V
z_L	z_V

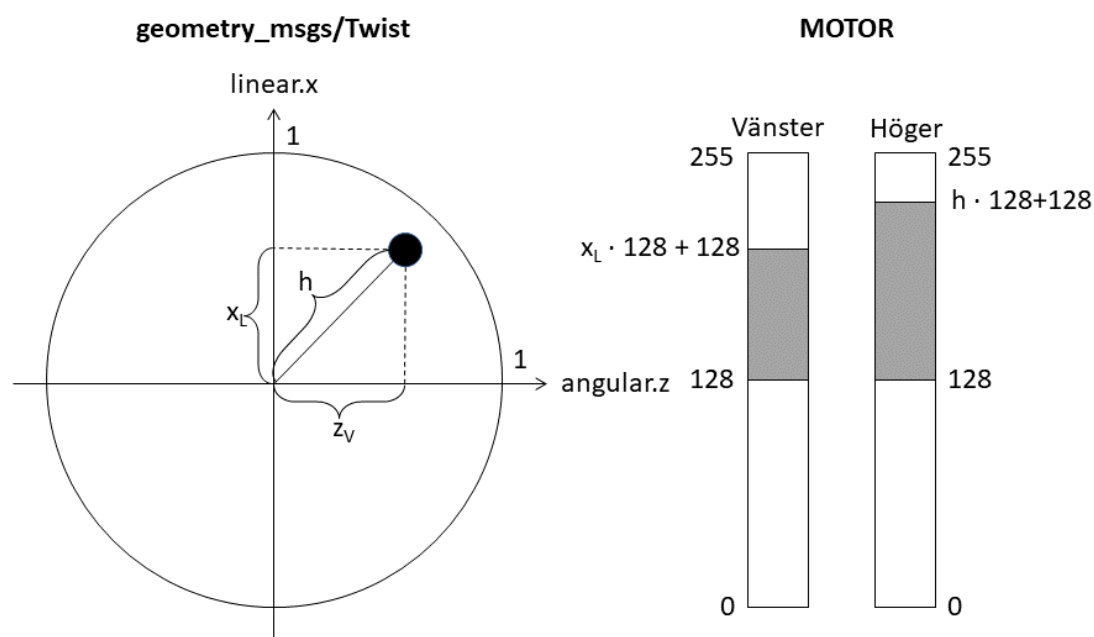
Det är endast x_L och z_V som används i detta projekt där x_L och z_V är decimaltal i intervallet $[-1,1]$ med kriteriet $\sqrt{x_L^2 + z_V^2} \leq 1$. x_r är riktning framåt i robotens x-led, z_r är rotationshastighet kring robotens z-axel med positiv riktning moturs, se Figur 3.8. Om scenariot är enligt Figur 3.8 kommer vägplaneringen skicka ett positivt x_L - och z_V -värde för att robotens lokala x_r är i positiv riktning för att kunna ta sig till målet. Även z_V kommer vara positivt då z_r roteras moturs.



Figur 3.8: Grafiskt exempel på en planerad väg där robotens lokala och globala koordinater är utmärkta. x och y är de globala koordinaterna och x_r , y_r och z_r är de lokala koordinaterna.

3.4.3 Kommunikation - Drivning

Arduinon som kommunicerar med motorerna via motorkortet, MD25, prenumererar på en topic vid namn `cmd_vel`. Meddelande som publiceras på denna kanal är av typen `geometry_msgs/Twist`, vilket som tidigare nämnts beskriver två vektorer för linjär- och vinkelhastighetsvärden. Motorkortet använder sig utav värden från 0 till 255 där 0 är högsta fart bakåt, 128 stilla och 255 högsta fart framåt. Därför måste meddelande från `cmd_vel` omvandlas till värden som motorkortet kan tolka vilket görs enligt Figur 3.9. Omvandlingen blir annorlunda beroende på vilken kvadrant av cirkeln punkten befinner sig i enligt Tabell 3.2. I figuren och tabellen används h som hypotenusan, $h = \sqrt{x_L^2 + z_V^2}$.



Figur 3.9: Robotens position som utgörs av koordinater i x - och z -led i den vänstra bilden görs om med ekvationerna för kvadrant 1 till ett motorvärde mellan 0 - 255 som motorerna tar emot och får då information om hur mycket höger respektive vänster motor ska rotera

Tabell 3.2: Tabellen visar ekvationerna som omvandlar en position till ett motorvärde. Kvadrant representerar vilken kvadrant som meddelandetypens koordinater befinner sig i, i en cirkel.

Kvadrant	Motor vänster	Motor höger
1	$x_L \cdot 128 + 128$	$h \cdot 128 + 128$
2	$h \cdot 128 + 128$	$x_L \cdot 128 + 128$
3	$x_L \cdot 128 + 128$	$-h \cdot 128 + 128$
4	$-h \cdot 128 + 128$	$x_L \cdot 128 + 128$

Enligt Tabell 3.2 och Figur 3.9 kan motorvärdena bli 256. I programkoden för denna komponent finns därför en begränsning som gör att värdet maximalt kan bli 255. Vid axlarnas gränser, när $z = 0$ eller $x = 0$, blir det specialfall. Vid $z = 0$ blir det ingen skillnad då $x = h$ men vid $x = 0$ ska roboten rotera. Det görs genom att istället för ekvationerna i Tabell 3.2 sätts motorvärdena till ekvationerna 3.2 och 3.3 istället.

$$\text{Motor vänster} = -z_V \cdot 128 + 128 \quad (3.2)$$

$$\text{Motor höger} = z_V \cdot 128 + 128 \quad (3.3)$$

För att bättre kontrollera och minska risken för att roboten kolliderar skalas hastigheten om till en tiogradig skala där 1 är långsammast och 10 snabbast. Vid uppstart har denna valt att sättas till 3 och kan ändras genom att publicera ett nytt tal på en

topic vid namn *speed*, se Bilaga A. Slutgiltiga motorvärdet skickas från Arduinon till motorkortet och vidare till slutpunkten på systemet, nämligen elmotorerna, EMG30. Alla mellansteg som tidigare har förklarats leder slutligen fram till att roboten rör sig i den riktning som efterfrågats.

4. Kravverifiering

I detta kapitel verifieras de krav som presenterades i Avsnitt 1.3. I kravspecifikationen är kraven uppdelade i fyra underrubriker, delfunktioner, säkerhet, prestanda och användargränssnitt där samtliga kommer verifieras. Därefter presenteras ett sammanfattande resultat som går igenom svaren på de frågorna som formulerades i Avsnitt 1.2 Problemformulering.

4.1 Delfunktioner

Huvudfunktionen delades in fyra delfunktioner som testats och presenteras i detta avsnitt. De delfunktioner som testats är ifall roboten kan köra framåt/bakåt, svänga, hitta väg samt veta när den är framme. Kraven och hur de testades finns beskrivna nedan. Efter att kravverifieringsmetoderna hade specificerats kunde en testbana för roboten sättas upp. För detta valdes ett rum med tydligt definierad geometri. I detta rum användes en och samma punkt som målpunkt i alla tester för att kunna avgöra ifall roboten kunde ta sig till samma punkt vid upprepade tillfällen. Testbanan visas i Figur 4.1.



Figur 4.1: I bilden visas resultatet av projektet. Den röda fyrkanten med endast några få röda pilar visar på att roboten vet med noggrannhet var den befinner sig på kartan. Den blå linjen är den planerade vägen av Dijkstras som sedan tas över av DWApplanner, som representeras av den gröna linjen, när ett hinder har upptäckts.

Krav 1.1

Krav 1.1 utfördes för att testa om roboten kan köra framåt/bakåt. Testet utfördes genom att låta roboten köra på en rak sträcka från olika positioner i rummet till samma mål. Resultatet mättes med linjal och roboten avvek enligt Tabell 4.1.

Tabell 4.1: I tabellen presenteras resultatet för kravverifieringen av funktion 1.1.

Kriterium	Test 1	Test 2	Test 3
Köra framåt/bakåt	3 cm	0 cm	0 cm

Kravet var att roboten inte skulle avvika mer än 20 cm, vilket uppfylldes med god marginal.

Krav 1.2

Krav 1.2 utfördes för att testa om roboten kunde svänga. Kravet om att kunna svänga testades genom att rita ut en pil 90° från robotens position. Resultatet verifierades med mätning och roboten avvek enligt Tabell 4.2.

Tabell 4.2: I tabellen presenteras resultatet för kravverifieringen av funktion 1.2.

Kriterium	Test 1	Test 2	Test 3
Svänga	-5°	10°	2°

Kravet var att roboten inte skulle avvika mer än $\pm 15^\circ$, vilket uppfylldes.

Krav 1.3

Krav 1.3 utfördes för att testa om roboten kan hitta en väg till ett förutbestämt mål. Kravet om att hitta en väg verifierades genom att skicka koordinater till roboten och sedan visuellt testa om den autonomt kunde förplanera en väg i en uppritad karta, detta utan att roboten utförde någon förflyttning. Detta klarade roboten vid samtliga tre tester.

Krav 1.4

Krav 1.4 utfördes för att testa om roboten kan förflytta sig till ett förutbestämt mål. Kravet om att hitta väg verifierades genom att skicka koordinater till roboten och sedan visuellt kontrollera ifall roboten förflyttade sig mot det förbestämde målet samt mäta avståndet från platsen roboten stannat på till den skickade koordinaten. Se Tabell 4.3 för resultat.

Tabell 4.3: I tabellen presenteras resultatet för kravverifieringen av funktion 1.4.

Kriterium	Test 1	Test 2	Test 3
Avstånd till verkligt mål	10 cm	6 cm	9 cm

Kravet var att roboten inte skulle avvika mer än 50 cm, vilket uppfylldes.

4.2 Säkerhet

Säkerheten är viktig för att roboten inte ska förstöras eller förorsaka skada på föremål eller människor på något vis. Vid verifiering av säkerhetskraven användes samma testbana som i verifieringen av delfunktionskraven. Förutom att köra denna bana användes en rad olika flyttbara hinder som placerades ut på olika platser på banan. De krav som här verifierades berörde robotens förmåga att upptäcka och undvika

väggar och hinder. Verifieringsmetoderna specificeras för respektive krav i avsnitten nedan.

Krav 2.1

Att kunna upptäcka väggar verifierades med hjälp av inputsignaler från LiDARn när den globala kartan ritas upp för första gången. Detta testades också genom visuell verifiering av att rummets dimensioner stämde överens med det som ritades ut i kartan. Testet genomfördes tre gånger och alla gånger upptäckte roboten alla väggar som var ett hinder för förflyttning.

Krav 2.2

Att upptäcka hinder testades på två olika sätt. Dels för den globala kartan, och dels för den lokala. Den globala kartan testades genom att hinder i verkligheten och hinder i kartan jämfördes i likhet med varandra. I den lokala kartan testades att placera ut hinder i vägen när roboten var i rörelse mot det tänkta målet, för att se ifall dessa dök upp i robotens vy. Båda varianter av test gjordes tre gånger och gav ett utfall om 100%. Roboten anses därmed ha uppfyllt kravet.

Krav 2.3

Test för undvikning av väggar skedde under tiden de andra testen pågick genom att samtidigt kolla att roboten inte kolliderade med en vägg. Det visuella testet visade att roboten undvek väggar i 100% av fallen.

Krav 2.4

Kravet om att undvika hinder delades in i två delar som testades separat. Hinder vars dimensioner syns av LiDARn och hinder vars dimensioner inte syns av LiDARn. De förstnämnda testades genom att flera hinder placerades i körbanan för roboten. Tre tester utfördes och visuella kontroller visade att roboten inte kolliderade med någon av hindren, samt att den hittade vägar runt dessa hinder i 100% av fallen.

För hinder vars dimensioner inte syns av LiDARn blev verifieringen annorlunda. Det står klart att LiDARn inte upptäcker föremål med utstickande delar mer än 20 cm ovanför markytan. Till exempel upptäcks och undviks inte en bordsskiva utan enbart ben på bord högre än 20 cm. Verifieringen gick ut på att se hur roboten reagerade på olika bredd mellan bordsbenen. Testet utfördes genom att utgå ifrån en benbredd lika stor som robotens bredd och sedan öka med 10 cm för att se var gränsen går när roboten avgör att den kan åka igenom.

Resultatet visade att när benen placerades med 70 cm mellanrum planerade roboten sin väg mellan de två benen, men stannade och avbröt vägen när den närmade sig hindret och insåg att marginalerna i sidled var för små. Vid 90 cm mellanrum mellan bordsbenen körde däremot roboten igenom. Det medför att till exempel bord med ett mellanrum mellan benen på 70 cm eller mer gör att roboten inte kommer

tolka detta som ett hinder utan kan välja att köra mellan dess ben. Är dessutom mellanrummet över 90 cm kommer då en kollision att ske när roboten försöker köra mellan bordsbenen och därmed krocka. Resultaten av dessa tester visas i Tabell 4.4.

Tabell 4.4: Hinder med ben

Benbredd	Utfall
40cm	Körde väg runt hindret
50cm	Körde väg runt hindret
60cm	Körde väg runt hindret
70cm	Planerade väg genom hindret men stannade
80cm	Planerade väg genom hindret men stannade
90cm	Kolliderade med hindret

4.3 Prestanda

Uppdatering av robotens position kan väljas i en parameter i AMCL. Denna är satt till att uppdatera en gång per 0.1 m. Maximala hastigheten är 0.57 m/s [14]. Antalet uppdateringar av positionen som görs är enligt Ekvation 4.1, 5.7 gånger per sekund. Detta uppfyller kravet, men vid lägre hastighet kommer uppdateringen bli lägre.

$$0.57/0.1 = 5.7 \text{ gånger/s} \quad (4.1)$$

4.4 Användargränssnitt

Applikationen svarar på en förfrågan från användaren och kan i ett förutbestämt område med förutbestämda mål få roboten att leda användaren till dessa. Kravet är därmed uppfyllt.

4.5 Sammanfattade resultat

Problemformuleringarna som introducerades i inledningen har under projektets gång blivit implementerade. Dessa frågor svaras på nedan. Då förflyttningen och hinderundvikningen samverkar beskrivs de under samma punkt.

- Hur ska roboten veta sin nuvarande position?
 - Roboten använder sig av en algoritm vid namn AMCL som i en redan uppritad karta kan räkna ut var den befinner sig med hjälp av att analysera kartan och estimerar var det är mest sannolikt att den är positionerad.

-
- Hur ska roboten förflytta sig från punkt A till punkt B? Och, hur ska roboten undvika hinder?
 - Detta gör roboten genom att tolka en sparad uppritad karta, den globala kartan, som en graf med noder och viktade kanter. Dijkstras algoritm använder grafen för att räkna ut en väg som undviker hindren. I de fall den lokala kartan upptäcker ett hinder på vägen, som inte finns i den globala kartan, tar DWAp planner över styrningen och planerar en väg som undviker hindret.
 - Hur ska användaren kommunicera med roboten?
 - Detta sker via ett gränssnitt, en Androidapplikation, som utvecklats under projektets gång.

5. Diskussion

Testerna som presenterades i föregående kapitel uppfyller projektets krav till god marginal i de flesta fall. Huvudfunktionen för projektet, att kunna leda en användare till en bestämd plats, är uppfylld med några felmarginaler som presenterades under resultatet för delfunktionerna. Roboten kan till exempel köra framåt/bakåt längs en raksträcka utan att avvika mer än 3 cm. Detta är ett värde som är bättre än förväntat då kravet hade en maximal felmarginal på 20 cm. Det antas att det nästintill korrekta värdet kommer från att roboten hela tiden läser av sin omgivning och kan korrigera under tiden den rör på sig. Det är även så att roboten förflyttar sig från start till mål med kortast möjliga väg. Att då avvika från linjen skulle medföra en omväg och det alternativet valdes alltid bort. Testet där det verifierades om roboten kunde svänga eller inte gav också ett bra resultat. Den största avvikelserna roboten gjorde från 90° var $+10^\circ$, då kravet maximalt fick vara $\pm 15^\circ$ ansågs resultatet vara väl godkänt. Resultatet är spritt som kan beror på handhavandefel vid utritning av pil. Detta test hade kunnat göras säkrare till exempel genom att ange koordinater med mål som skiljer 90° . Vad gäller igenkännande av mål kommer roboten mycket nära sin slutdestination men inte exakt. Som mest kom roboten 10 cm ifrån det tänka målet. Detta tros bero på att algoritmen AMCL inte alltid kan bestämma korrekt position med så små marginaler. I detta avseende anses dock 10 cm vara fullt godkänt.

Robotens säkerhet försäkras inte alltid i detta projekt. Robotens LiDAR sitter så pass nära marken att den bara tar hänsyn till hinder 20 cm från markplan. Detta gör att roboten kan se och undvika samtliga väggar och solida objekt. Vid hinder som har ben med någon slags skiva minst 20 cm över golvet får roboten problem. Detta är för att LiDARn inte kan se skivan och således inte undvika den. Om det är ett objekt med flera ben, såsom ett bord, kan roboten undvika det om benen befinner sig så pass nära varandra att roboten anser att vägen mellan benen är en farlig zon. Roboten kommer då förflytta sig runt detta hinder. Om benen är långt ifrån varandra kommer utrymmet mellan ses som säker zon och kollision med bordsskivan kommer ske.

Det är värt att nämna att testerna är utförda i en optimal miljö. Resultatet av testerna kommer därav inte stämma överens med hur robotens faktiska användning i den tänkta miljön hade fungerat. Det är därav mycket möjligt att resultatet av liknande test i en mer komplex miljö hade uppbringat fel och problem som idag inte är synliga. Exempelvis hade det varit intressant att se hur roboten skulle klara av en korridor, där teorin säger att algoritmen AMCL och Hector-SLAM har svårt för att lokalisera i en karta på områden där det ser mer eller mindre likadant ut.

I inledningen påvisades en önskan om att unga skulle få upp intresset för teknik. Det har inte testats i detta projekt då det hade varit för tidskrävande och inte direkt följer projektets fokus. Testerna visar dock att roboten inte kolliderar med hinder, vilket innebär att den skulle vara säker för unga att använda. Däremot skulle användargränssnittet troligtvis bli svårhanterligt då användarvänligheten av applikationen inte anses vara tillräckligt utvecklad.

I det stora hela är gruppen mycket nöjd med resultatet av projektet. Syftet med projektet var att få ökad förståelse för implementering av ett navigeringssystem i en autonom robot. Det goda resultatet på testerna visar på att gruppen har förstått hur arbetet ska utföras. Efter utfört arbete var det även möjligt för gruppen att besvara de frågor som sammanställdes under Problemformulering som tyder på att förståelse har uppnåtts.

Samhälls- och etiska aspekter

Vid arbete med robotteknik och autonoma system bör det tänkas på hur detta kan komma att påverka samhället. Roboten som är utvecklad i detta projekt skulle efter en vidareutveckling kunna användas på exempelvis sjukhus, hotell, museum, större butiker eller andra platser där personer behöver vägvisning. Att inte betala anställda för att göra detta utan ha en robot som exempelvis visar patienter till rätt avdelning hade låtit en anställd kunna fokusera på viktigare arbete. För att detta ska fungera behöver människan vänja sig vid att robotar är en del av samhället. Detta är en väldigt stor uppgift och troligtvis kommer det ta lång tid innan detta är verklighet.

Inom ramarna för detta projekt finns det inga tydliga kopplingar till hur det skulle kunna ha en stor påverkan på samhället. I förlängningen av projektets mål, navigering och lokalisering, finns det många spännande utvecklingsområde. Till exempel skulle robotar kunna utvecklas som kan användas för att hitta offer och/eller skadade i katastrofområden. Förutom detta finns det andra områden där det är riskabelt att skicka in människor eller andra levande varelser där det hade varit fördelaktigt att istället kunna använda sig utav en robot, som till exempel vid släckning av eld. Då det kan vara livsfarligt för människor att vistas i ett eldhärjat område.

En annan aspekt som är beaktansvärd är interaktionen mellan robot och människa. Hur roboten ska bete sig i närheten av människor är också en viktig aspekt att ta hänsyn till. Det finns många viktiga sociala koder och beteenden som måste tas hänsyn till för att människor ska kunna använda robotar och samtidigt känna sig trygga. Som exempel från detta projektet tar inte roboten hänsyn till om hindret som upptäcks är ett föremål eller något levande, detta kan göra att människor kan känna sig obekväma i dess omgivning.

6. Slutsatser och framtida arbete

Projektet har genomfört och resultatet faller inom dess marginaler. Roboten kan via en egenutvecklad Androidapplikation leda en användare till en utsatt plats i ett mindre område. Roboten undviker även enklare hinder som är synliga på 20 cm från golvet.

Implementering av navigeringssystem är ett komplicerat problem med många olika lösningar där tid inte funnits till att testa fler än vad rapporten tagit upp. Exempelvis finns det fler metoder som möjliggör kartuppritning istället för Hector-SLAM, som kan vara intressanta att testa. Även för Androidapplikationen finns stora utvecklingsmöjligheter. Exempelvis kan möjligheter för WiFi-kommunikation analyseras. Detta skulle medföra att en användare kan stå positionerad var som helst i det valda området och då utöka robotens funktionalitet till att först hämta användaren och sedan leda denne till det bestämda målet. Applikationen kan även ses över visuellt då interaktionsdesignen ej varit ett fokus i detta arbete, detta är ett sätt att öka användarvänligheten.

Även för mjukvaran finns förslag på förbättringar. I dagsläget behövs en manuell körning av roboten, i det rum som ska användas, innan den kan navigera självständigt. Något som borde göras i framtida projekt är att automatisera denna process. Dessutom kan både SLAM och IMUn optimeras, dess parametrar kan ställas in noggrannare och ett arbete med att försöka minimera felvärden undersökas.

Vissa komponenter och mjukvara kan behövas ses över och bytas ut. För att kunna applicera detta projekts lösning på mer verkliga miljöer, exempelvis ett klassrum, behöver LiDARNs uppgraderas. Nuvarande lösning hanterar hinder på 20 cm höjd, vilket gör att exempelvis bord inte kan hanteras väl av roboten. En LiDAR som har möjlighet att upptäcka hinder i mer omfattande höjdskillnader hade förbättrat lösningen för att testas i mer komplexa miljöer.

En aspekt som inte haft stort fokus i projektet är själva interaktionen mellan människa och robot. Detta är en studie som anses vara mycket intressant och relevant för ett framtida arbete.

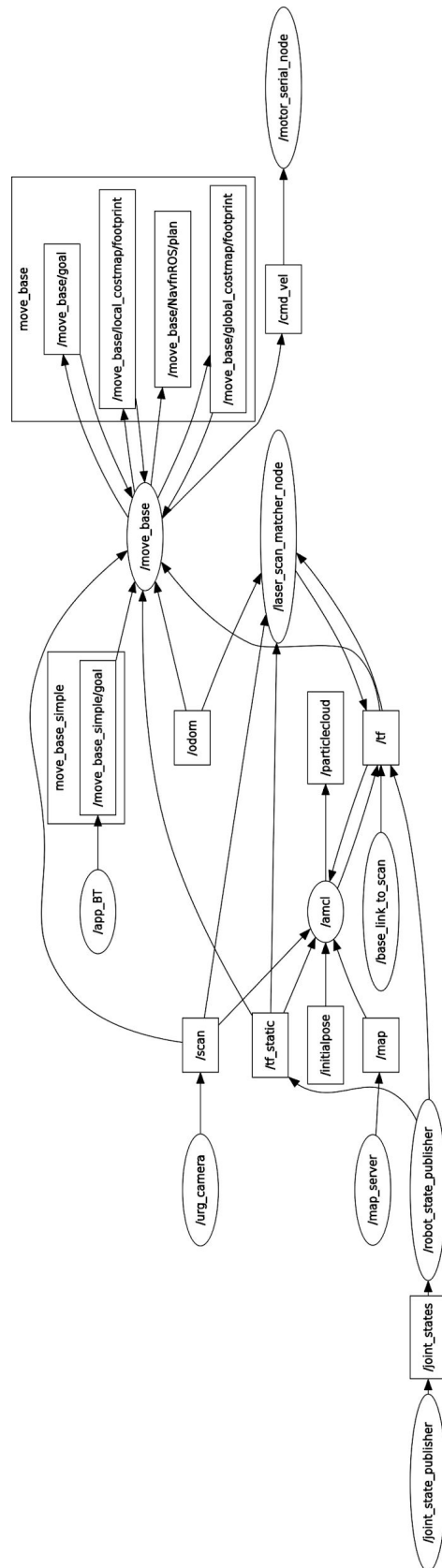
Referenser

- [1] Sparkfun, *Accelerometer Basics*, 2018. URL: <https://learn.sparkfun.com/tutorials/accelerometer-basics>.
- [2] Arduino, *ARDUINO UNO REV3*, 2018. URL: <https://store.arduino.cc/arduino-uno-rev3>.
- [3] J. Strömbergson, *Så fungerar systembussarna*, 2008. URL: <https://techworld.idg.se/2.2524/1.168072/sa-fungerar-systembussarna>.
- [4] M. Brain, *How Gyroscopes Work*, 2018. URL: <https://science.howstuffworks.com/gyroscope.htm>.
- [5] I. I2C Info - I2C Bus och Protocol, *I2C Info - I2C Bus, Interface and Protocol*, 2018. URL: <http://i2c.info/>.
- [6] A. Lotsson, *lidar*, 2017. URL: <https://it-ord.idg.se/ord/lidar/>.
- [7] Meccano, *The History of Meccano*, 2018. URL: <http://www.meccano.com/about>.
- [8] A. Lotsson, *Mikrokontroller*, 2017. URL: <https://it-ord.idg.se/ord/mikrokontroller/>.
- [9] Ros Wiki, *ROSIntroduction*, 2018. URL: <http://wiki.ros.org/ROS/Introduction>.
- [10] redhat, *What is open source?*, 2018. URL: <https://opensource.com/contact>.
- [11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler och A. Mg, "ROS: an open-source Robot Operating System", *Icra*, årg. 3, nr Figure 1, s. 5, 2009, ISSN: 0165-022X. DOI: <http://www.willowgarage.com/papers/ros-open-source-robot-operating-system>. URL: <http://pub1.willowgarage.com/~konolige/cs225B/docs/quigley-icra2009-ros.pdf>.
- [12] R. Siegwart och I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. 2004, vol. 23, s. 47–82, ISBN: 026219502X. DOI: 10.1109/ROBOT.2010.5509725.
- [13] S. Fölster, "Vartannat jobb automatiseras inom 20 år", s. 1–28, 2014. URL: <http://www.stratresearch.se/sv/ssf/Nyheter/2014/Vartannat-jobb-automatiseras/%5Cnhttp://www.stratresearch.se/Documents/Folder.pdf>.
- [14] J. Carlsson, E. Claesson, A. Edvardsson, C. Hansson, S. Marklund och W. Mo, "Humanoid Robot", diss., Chalmers tekniska högskola, 2017.
- [15] E. Norén, E. Tegehall och D. Wall, "Enhancing the Meccanoid -Lillasyster is on her way", diss., Chalmers tekniska högskola, 2016.
- [16] H. Lindén, N. S. Kurien och F. Bergström, "ROS Framework for an Autonomous Robot with Visual Guidance System", System, Control och Mechatronic, tekn. rapport, 2016.
- [17] Allt om vetenskap, *Kan ett flygplan landa automatiskt?*, 2008. URL: <http://www.alltomvetenskap.se/nyheter/kan-ett-flygplan-landa-automatiskt>.
- [18] R. Schirmer, P. Biber och C. Stachniss, "Efficient path planning in belief space for safe navigation", i *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2017-Septe, IEEE, sept. 2017, s. 2857–2863,

-
- ISBN: 978-1-5386-2682-5. DOI: 10.1109/IRoS.2017.8206117. URL: <http://ieeexplore.ieee.org/document/8206117/>.
- [19] I. F. A. Vis, "Survey of research in the design and control of automated guided vehicle systems", *European Journal of Operational Research*, årg. 170, nr 3, s. 677–709, 2006, ISSN: 03772217. DOI: 10.1016/j.ejor.2004.09.020.
 - [20] X. Wu, Y. Zhang, T. Zou, L. Zhao, P. Lou och Z. Yin, "Coordinated path tracking of two vision-guided tractors for heavy-duty robotic vehicles", *Robotics and Computer-Integrated Manufacturing*, årg. 53, s. 93–107, 2018. DOI: 10.1016/j.rcim.2018.03.012.
 - [21] M. I. Ahmad, O. Mubin och J. Orlando, "Adaptive Social Robot for Sustaining Social Engagement during Long-Term Children–Robot Interaction", *International Journal of Human-Computer Interaction*, årg. 33, nr 12, s. 943–962, 2017, ISSN: 15327590. DOI: 10.1080/10447318.2017.1300750.
 - [22] Robot Electronics, *MD25 - 12v 2.8A dual H-bridge motor driver*, 2018. URL: <http://www.robot-electronics.co.uk/htm/md25tech.htm>.
 - [23] Sparkfun, *Bluetooth Basics*, 2018. URL: <https://learn.sparkfun.com/tutorials/bluetooth-basics>.
 - [24] X. Ling, *Product Data Sheet*, 2018. URL: <https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf>.
 - [25] Sparkfun, *Serial Communication*, 2018. URL: <https://learn.sparkfun.com/tutorials/serial-communication/all>.
 - [26] Hokuyo, *Distance Data Output/URG-04LX-UG01*, 2014. URL: <https://www.hokuyo-aut.jp/search/single.php?serial=166>.
 - [27] TDK, *MPU-9150*, 2018. URL: <https://www.invensense.com/products/motion-tracking/9-axis/mpu-9150/>.
 - [28] E. Bessarabova, *6 Differences Between iOS and Android App Development*, 2018. URL: <https://www.upwork.com/hiring/for-clients/ios-vs-android-app-development/>.
 - [29] D. Schneider, *New Indoor Navigation Technologies Work Where GPS Can't*, 2013. URL: <https://spectrum.ieee.org/telecom/wireless/new-indoor-navigation-technologies-work-where-gps-cant>.
 - [30] O. R. Popoola och S. Sinanovic, "Design and analysis of collision reduction algorithms for LED-based indoor positioning with simulation and experimental validation", *IEEE Access*, årg. 6, s. 10 754–10 770, 2018, ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2801626.
 - [31] Marvelmind robotics, *Description*, 2017. URL: <https://marvelmind.com/product/starter-set-hw-v4-9-plastic-housing/>.
 - [32] GISGeography, *Trilateration vs Triangulation – How GPS Receivers Work*, 2018. URL: <https://gisgeography.com/trilateration-triangulation-gps/>.
 - [33] T. Bailey och H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part I", *IEEE Robotics and Automation Magazine*, årg. 13, nr 3, s. 108–117, 2006, ISSN: 10709932. DOI: 10.1109/MRA.2006.1678144.
 - [34] D. R. Tobergte och S. Curtis, *Handbook of Robotics*, 9. 2013, vol. 53, s. 1689–1699, ISBN: 9788578110796. DOI: 10.1017/CB09781107415324.004.

-
- [35] Nationalencyklopedin, *kovariansmatris*, 2018. URL: <https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/kovarians>.
- [36] —, *Bayes sats*, 2018. URL: <https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/bayes-sats>.
- [37] D. Fox, W. Burgard, F. Dellaert och S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots", *Aaai-99*, nr Handschin 1970, s. 343–349, 1999. DOI: 10.1.1.2.342. URL: <http://dl.acm.org/citation.cfm?id=315322>.
- [38] ROS Wiki, *navfn*, 2014. URL: <http://wiki.ros.org/navfn>.
- [39] —, *carrot planner*, 2018. URL: http://wiki.ros.org/carrot_planner.
- [40] —, *dwa local planner*, 2018. URL: http://wiki.ros.org/dwa_local_planner.
- [41] —, *base local planner*, 2018. URL: http://wiki.ros.org/base_local_planner#TrajectoryPlannerROS.
- [42] GITTA, *Dijkstra Algorithm: Short terms and Pseudocode*, 2018. URL: http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html.
- [43] ROS Wiki, *geometry_msgs/Twist Message*, 2018. URL: http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html.

A. Systemet i ROS vid navigering



B. Systemet i ROS vid SLAM

