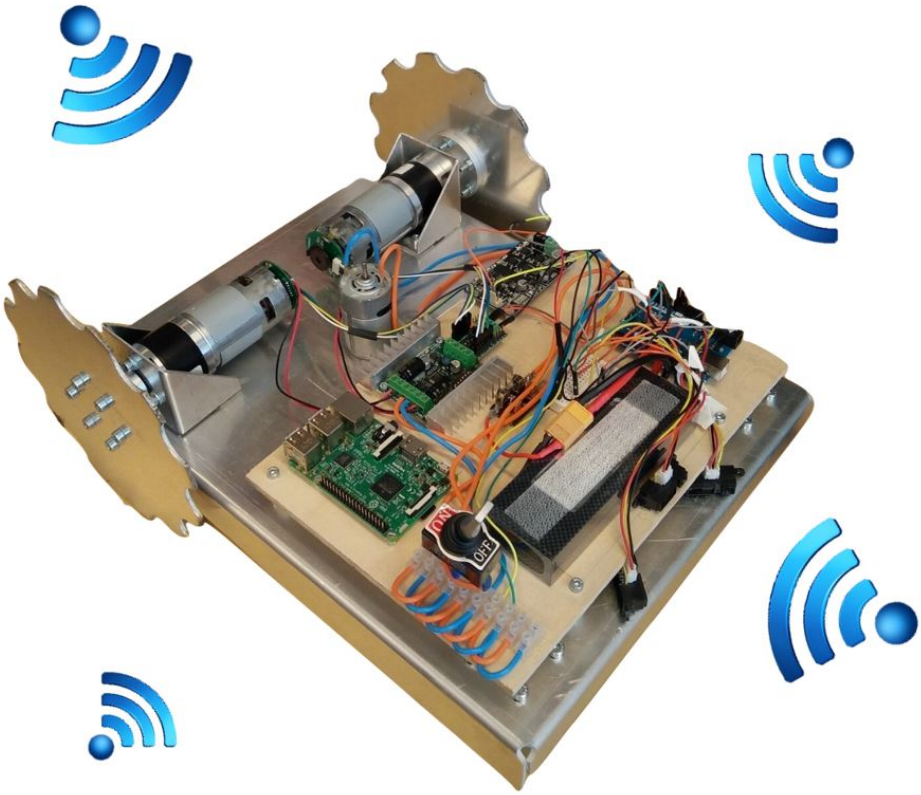
CHALMERS
UNIVERSITY OF TECHNOLOGY



Robotgräsklippare med ruttplanering

En robotgräsklippare som planerar sin rutt med hjälp av ett lokalt positioneringssystem

Ett kandidatarbete vid Institutionen för Signaler och System

Alexander Sjögren
Anton Gustafsson
Mickel Hoang
Viktor Josefsson

Institutionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2017

Kandidatarbete 2017 : SSYX02-17-81

Robotgräsklippare med ruttplanering

En robotgräsklippare som planerar sin rutt med hjälp av ett lokalt positioneringssystem

ALEXANDER SJÖGREN
ANTON GUSTAFSSON
MICKEL HOANG
VIKTOR JOSEFSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Signaler och System
Chalmers Tekniska Högskola
Göteborg, Sverige 2017

Robotgräsklippare med ruttplanering
En robotgräsklippare som planerar sin rutt med hjälp av ett lokalt positioneringssystem
ALEXANDER SJÖGREN
ANTON GUSTAFSSON
MICKEL HOANG
VIKTOR JOSEFSSON

Handledare: Balázs Kulcsár, Signaler och System
Examinator: Torsten Wik, Signaler och System

Kandidatarbete 2017 : SSYX02-17-81
Institutionen för Signaler och System
Chalmers Tekniska Högskola
412 96 Göteborg
Sverige
Telefon: +46 31 772 1000

Omslag: Foto på den framtagna prototypen adderad med animerade symboler för trådlös kommunikation.
Skriven i LaTeX
Göteborg, Sverige 2017

Sammanfattning

Detta projekt tar upp några av de nackdelar som finns med dagens robotgräsklippare.

Syftet med projektet är att utveckla en fullt fungerande robotgräsklippare. Vidare fokuserar projektet på att utveckla en lösning för ett kostnadseffektivt och noggrant positioneringssystem för gräsklipparen. Denna lösning är baserad på avståndsmätningar gjorda med ultraljudssensorer som använder trilaterationsberäkningar för att bestämma robotgräsklipparens position. Fortsättningsvis, då gräsklipparen vet var på gräsmattan den befinner sig, har en ruttplaneringsalgoritm utformats för att effektivt kunna klippa hela gräsmattan. Denna lösning syftar att kunna jämföras med marknadens "dumma" robotgräsklippare som kör slumpvis mellan begränsningskablar. På grund av att det tar extra tid att klippa gräsmattan, drar den mer energi. Detta vill projektet lösa via användning av en smart ruttplanering.

Resultatet av detta projekt var tre separata system som fungerar oberoende av varandra. Systemen som utvecklades var ett positioneringssystem, en ruttplaneringsalgoritm och en robotgräsklippare som kan klippa gräs och undvika hinder. På grund av den begränsade tiden och att prestandan för positioneringssystemet inte var bra nog, kunde dessa delar inte kopplas ihop och testas tillsammans. Rapporten tar också upp nya tillvägagångssätt för framtida arbeten med positioneringssystemet och ruttplaneringsalgoritmen.

Nyckelord: robotgräsklippare, ruttplanering, positioneringssystem, ultraljud, trilateration.

Abstract

This thesis work targets some of the pitfalls with nowadays robotic lawn mowers.

The purpose of the project is to develop a fully functional robotic lawn mower. However the project concentrate on developing a solution for a cost effective and accurate positioning system for the mower. The solution involves distance measurement by ultrasonic sensors using trilateration for position calculations. Further, by means of the local position information, a route learning method has been proposed to cover the cutting area effectively. The proposed solution aims to compare with the market's "dumb" lawn mowers that moves randomly between boundary wires. Due to the extended time it takes to cut the grass it consumes more energy. The project aims to solve this problem by a smart route planning algorithm.

The results of this thesis was three separate systems working independtly. The systems developed was a positioning system, a route planning algorithm and a robotic lawn mower that cuts grass and avoid obstacles. Due to limited time and the fact that the performance of the positioning system was not good enough, these parts did not get connected and tested together. The thesis also suggest new approaches for future work on the positioning system and the route planning system.

Keywords: robotic lawn mower, path planning, positioning system, ultrasonic, trilateration.

Förord

Denna rapport är, tillsammans med en robotgräsklippare, ett positioneringssystem och en ruttplanerande algoritm, resultatet av ett kandidatarbete som utfördes under vårterminen 2017 vid institutionen för Signaler och System på Chalmers Tekniska Högskola.

Vi vill tacka Göran Stigler, Reine Nohlborg och Jan Bragee för den vägledning och hjälp vi fick med konstruktionen av gräsklipparen. Vi vill också rikta ett stort tack till vår handledare Balázs Kulcsár, som med stor entusiasm och lysande optimism, guidat och hjälpt oss genom projektet.

Alexander Sjögren, Anton Gustafsson, Micke Hoang, Viktor Josefsson

Göteborg, maj 2017

Innehåll

Figurer

Tabeller

1
Inledning

1.1 Bakgrund

Gräsklipparen uppfanns år 1827 av britten Edwin Beard Budding och patenterades senare år 1830. Idén med roterande klingor som klipper gräs fick han från en textilfabrik där man använde just roterande klingor för att bli av med uppstickande fibrer. Tidigare hade man slagit gräsmattor med lie eller klippt dem med exempelvis fårsaxar. Detta var givetvis tidskrävande och innebar mycket hårt arbete[1]. Idag, närmare 200 år senare talar mycket för att en ny övergång håller på att ske, nämligen den mellan vanliga gräsklippare och robotgräsklippare.

Den första robotgräsklipparen lanserades av Husqvarna år 1995. Detta är nu över två decennier sedan. Mycket har hänt gällande prestanda, men funktionen är fortfarande densamma, nämligen att den klipper gräsmattan åt användaren utan att något betydande arbete måste utföras av användaren. Dagens robotgräsklippare fungerar huvudsakligen på samma sätt, de har alla ett chassi med hjul och knivar, samt någon lösning för att navigera sig runt på gräsmattan. Dagens robotgräsklippare använder sig generellt sett av en så kallad begränsningskabel. Kabeln sänder ut en svag radiosignal och när robotgräsklipparen kommer fram till kabeln känner den av signalen och vänder. Begränsningskabeln läggs ut runt den önskade klippytan vid installation, och därefter blir den snabbt övervuxen av gräset[2]. De flesta gräsklipparna kör sedan i stort sett slumpvis inom det begränsade området. De senaste åren har dock vissa tillverkare börjat använda sig utav GPS-navigering. Denna är inte tillräckligt precis i sig för att exakt kunna lokalisera gräskipparen, men är ett bra komplement som hjälper till med att känna av vilka områden som gräsklipparen vistats lång tid på[3]. Vidare använder sig vissa gräsklippare av en så kallad guidekabel, denna används speciellt om man exempelvis har en smal passage som gräsklipparen måste ta sig igenom. Gräsklipparen kör som vanligt tills den hittar guidekabeln som den sedan följer tills den tagit sig igenom passagen.

När gräsklipparen börjar få en låg batterinivå återvänder den till sin laddstation. Detta har tillverkare löst på olika sätt. Vissa gräsklippare följer begränsningskabeln tills den kommer till laddstationen. Detta skapar dock slitage längs kanterna på gräsmattan, så att tillverkare tenderar numera att låta gräsklipparen söka efter laddstationen helt slumpvis, eller med hjälp av en guidekabel. Gräsklipparen kör då slumpvis tills den hittar guidekabeln, som leder den till laddstationen[4].

Trots robotgräsklipparens bekvämliga fördelar har den ännu inte kunnat konkurrera ut den vanliga gräsklipparen. Detta beror på att dagens robotgräsklippare presterar sämre än vanliga gräsklippare på flera områden[5]. Dels klarar de inte av allt för stora lutningar. De bästa klipparna klarar cirka 25 graders lutning. Svårigheterna ligger här i att gräsklipparen måste ha en så pass stark motor att den orkar ta sig upp för backen, samtidigt som den inte får tippa över eller slira. Detta kan vara extra problematiskt vid ojämn terräng, exempelvis då det finns rötter och stenar i backen[6].

En annan egenskap hos robotgräsklippare är att de har en rekommenderad ytkapacitet. Detta på grund av begränsande batterier och navigeringssystem som är långt ifrån perfekta. De bästa robotgräsklipparna klarar av ytor i storleksordningen 5000 m^2. Robotgräsklippare tar dessutom längre tid på sig att klippa en hel gräsmatta då de inte kör kortaste vägen[7]. Dessa problem skulle lösas om gräsklipparna körde en mer optimal rutt. Idag används flera olika lokala positionssystem tillsammans med ruttoptimerande algoritmer för att lösa liknande problem. Applikationerna hittas dock främst inomhus på grund av mer gynnsamma förhållanden[8].

Om robotgräsklipparens problem kunde lösas skulle en mycket bredare kundkrets öppna sig, främst för kunder med stora gräsmattor, gräsmattor med ojämn terräng eller gräsmattor med svår geometri. Med andra ord finns det alltså en stor utvecklingspotential för robotgräsklippare, och det är därmed också ett intressant projekt att titta närmare på. En annan sak som ska beaktas är att robotgräsklippare fortfarande är dyrare i relation till vanliga gräsklippare. De flesta robotgräsklippare kostar idag mellan 7 000 kr och 40 000 kr[9]. En intressant fråga är därför huruvida man kan bygga en egen robotgräsklippare till ett lägre pris.

1.2 Syfte

Det finns i huvudsak två syften med projektet. Dels är syftet att konstruera och bygga en fungerande robotgräsklippare till en förhållandevis låg ekonomisk kostnad. Mer specifikt ska gräsklipparen klara av att klippa en enkel plan gräsmatta - utan allt för svår terräng och geometri - på en rimlig tid. Syftet är också att försöka få gräsklipparen att optimera sin rutt. Detta ska åstadkommas genom användning av ett precist positioneringssystem. Om gräsklipparen kan positioneras tillräckligt precist, kan en optimal rutt programmeras upp. Positioneringssystemet som ska konstrueras och utvärderas bygger på ett koncept. Konceptet är ett lokalt positioneringssystem som använder sig av trilateration med hjälp av ultraljud.

1.3 Uppgiftsbeskrivning

Nedan specifieras vilka uppgifter den färdiga prototypen skall klara av. Därefter utvecklas de olika uppgifterna något och det ges också en bild över vilka komponenter som måste tas med i den färdiga prototypen. Mer specifikt ska den färdiga prototypen kunna:

- klippa gräs
- röra sig på egen hand
- hantera hinder
- välja rutt på ett optimerat sätt

Målet är också att detta ska kunna göras till en förhållandevis låg kostnad. Nedan följer en kort förklaring av varje deluppgift.

Klippa gräs
För att kunna klippa gräs krävs ett klippaggregat innehållande knivar och en motor som får knivarna att rotera. Vidare måste knivarna omslutas av ett chassi för att prototypen ska vara tillräckligt säker.

Röra sig på egen hand
För att gräsklipparen ska kunna röra sig krävs någon form av hjul. För att kunna röra sig på egen hand måste hjulen dessutom drivas med någon typ av motorer. Klipparen måste också kunna svänga och bromsa, vilket innebär att motorerna måste styras och regleras med något styrkort.

Hantera hinder
För att gräsklipparen ska kunna hantera ett hinder krävs någon form av sensor eller kamera som kan uppfatta hinder. Vidare måste gräsklipparen ta ett beslut för vad den ska göra när den upptäcker ett hinder, för att sedan kunna fortsätta klippa gräsmattan. För detta krävs någon slags mjukvara.

Välja rutt på ett optimerat sätt
För att spara tid och energi ska gräsklipparen utvecklas för att klippa hela gräsmattan på ett optimerat sätt. För att detta ska åstadkommas krävs att gräsklipparen vet sin (lokala) position. Ett positioneringssystem ska därför konstrueras. Vidare krävs en ruttplaneringsalgoritm som planerar gräsklipparens rutt.

Konstruktionen av positioneringssystemet ska bygga på ett koncept - trilateration med ultraljud. För att kunna bestämma en position med den matematiska metoden trilateration, så krävs ett antal referenspunkter. Utöver referenspunkterna krävs också avstånden mellan dessa och den sökta positionen. Konceptet bygger på att dessa avstånd fås med hjälp av ultraljudsmottagare och -sändare. Referenspunkterna ges var sin sändare och på gräsklipparen monteras en mottagare. Avstånden fås sedan genom att mäta tiden det tar för ultraljudssignalerna att röra sig mellan sändare och mottagare.

1.4 Avgränsningar

Projektet har en preliminär budget på 5000 kr. Utöver detta finns även en budget på 2000 kr till material från Chalmers prototyplab. Den begränsade budgeten medför att valet av komponenter måste göras varsamt. Det finns alltså inte utrymme för att köpa in flera dyra komponenter med samma uppgift och sedan testa dem.

En observation som ska göras är att projektets budget är lägre än inköpskostnaden för de absolut flesta robotgräsklipparna (vilket nämndes i "Bakgrund"). Denna avgränsning medför med andra ord att prototypen som tas fram främst bör jämföras med gräsklippare i budgetklassen.

Tiden till projektet är begränsat till fem månader och det kommer att vara fyra arbetande projektmedlemmar under denna tid. Den begränsade tiden medför att alla tänkbara lösningar inte kan utvärderas och testas. I och med detta kommer många val baseras på antaganden och teorier. Det finns heller inte tid till att själva konstruera och bygga vissa komponenter och därför kommer många delar att behöva köpas. Då slutprodukten inte ska säljas till någon kund, utan snarare är till för att idéer ska kunna testas och utvärderas, kommer den estetiska delen av gräsklipparen inte beaktas. Fokus läggs istället främst på funktionaliteten. Vidare kommer följande avgränsningar att göras på grund av den begränsade tiden:

- Då det finns erfarenhet i gruppen av Raspberry Pi och Arduino övervägs endast dessa som val av enkortsdatorer.
- Det kommer inte utredas huruvida motordrivna hjul är den bästa driftslösningen. Trots andra existerande lösningar som larvband, kommer av enkelhet hjul att väljas.
- Gräsklipparens klippaggregat kommer inte att optimeras.
- Ingen tid kommer att spenderas på att få gräsklipparen att köra upp för backar.
- Gräsklipparens ljudnivå kommer att bortses ifrån.
- Gräsklipparen kommer inte att konstrueras för att klara väderförhållanden i form av av regn, snö och kraftig vind.
- Delar av använd programmeringskod kommer att hämtas från andra källor. Koderna kommer sedan att anpassas efter projektets önskemål.
- Alternativa positioneringssytem kommer inte att övervägas, då syftet delvis är att utvärdera det tidigare nämnda konceptet.
- Vid test av positioneringssystemet kommer inga hinder att förekomma. Med andra ord kommer inte de två målen "undvika hinder" samt "ruttoptimering" att testas samtidigt. Testerna kommer dessutom att göras på begränsade ytor. Noteras ska också att syftet är att undersöka hur stor yta positioneringssystemet är kapabelt för, snarare än att att systemet skulle utformas för en specifik yta.

1.5 Metodik

För att uppnå önskat resultat kommer arbetet ske enligt en viss metodik. Till en början kommer en övergripande litteraturstudie om robotgräsklippare att göras. Därefter specificeras målen för projektet och mer djupgående litteraturstudier kan göras. När tillräckligt mycket kunskap har erhållits kommer en prototyp börja konstrueras, och sedan tillverkas. Den färdiga prototypen kommer sedan testas och förbättras successivt genom en iterativ process.

1.5.1 Informationsinsamling

Under litteraturstudien kommer dokument från diverse källor att studeras. Till en början kommer främst litteratur som behandlar robotgräsklippare på ett övergripande plan att studeras. Detta för att få en bred förståelse kring robotgräsklippare. Vidare kommer mer specifika områden att studeras, så som teknik för positionering och hur en algoritm för ruttplanering skrivs. Under litteraturstudien kommer i första hand vetenskaplig litteratur att studeras. Sökmotorer som Google kommer dock också att användas för att nå webbsidor med information som inte hittats i vetenskapliga rapporter. Detta kan exempelvis gälla information om kommersiella produkter eller instrument. Det kommer också att tittas närmare på tidigare kandidatarbeten med liknande uppgifter.

Under projektet kommer det dyka upp frågor, dessa kommer i första hand att ställas till projektets handledare. Då ytterligare expertis krävs, kommer andra ämneskunniga på Chalmers att konsulteras.

1.5.2 Konstruktion av prototyp

När tillräcklig information erhållits är tiden inne för att börja ta fram en konstruktion. Konstruktionen kommer i så stor utsträckning som möjligt att stödjas av beräkningar, specifikationer eller antaganden. Konstruktionen av prototypen kommer att behandla säkerhet, klippaggregat, motoruppsättning, utformning av hjul, sensorer, positioneringssystemet, utformning av chassi, elektronikens sammansättning samt hur delarna ska assembleras. Det kommer att skapas en ritning för den assemblerade gräsklipparen, samt ytterligare ritningar för de delar som väljs att egentillverka. Ritningarna kommer att göras med hjälp av Catia V5 då gruppen har tidigare erfarenheter i detta program.

1.5.3 Tillverkning av prototyp

Elektroniska delar till prototypen, såsom sensorer, motorer, batterier kommer att inhandlas, alternativt lånas från institutionen som ansvarar för projektet.

För att tillverka ett chassi till prototypen, samt montera ihop alla komponenter kommer Chalmers prototyplab användas. Här finns det tillgång till material i form av trä och metall, samt ett stort antal maskiner för bearbetning. Det finns dessutom tillgång till 3D-utskrift. Detta bör vara tillräckligt för att tillverka nödvändiga delar till prototypen, samt assemblering.

1.5.4 Utvärdering och test av prototyp

Då prototypen är färdig ska den testas. Flera olika tester kommer att utföras. Dels kommer positioneringssystemets precision att testas. Det kommer även göras tester för hur väl mjukvaran fungerar, alltså ruttplaneringen och hur väl den undviker hinder. Dessutom kommer ett klipptest att göras, alltså hur väl den klipper gräs.

Positioneringssystemet testas lämpligtvis inomhus först, för att sedan testas utomhus i mån av tid. Klipptestet utförs givetvis utomhus på en öppen gräsmatta, för enkel installation och överblick. Förhoppningsvis kan tester senare också göras i en vanlig trädgård för att säkerställa att robotgräsklipparen beter sig som tänkt.

1.6 Översikt av rapporten

Detta avsnitt beskriver kortfattat rapportens upplägg. Rapporten startar med en inledning som sedan följs av ett teoriavsnitt där de viktigaste begreppen för konstruktionsdelen tas upp. Därefter följer konstruktionsdelen som beskriver hur gräsklipparen konstruerades, tillverkades och hur komponenter valdes. Rapportens avslutande delar är resultatdel och slutsats. I resultatdelen beskrivs hur tester utförts och dess resultat och i slutsatsen diskuteras utfallet av dessa och framtida studier och utvecklingar.

2
Teori

I detta avsnitt förklaras ett antal tekniska begrepp, verktyg och instrument som behandlas i rapporten. De teoretiska avsnitten är här indelade i kategorier som speglar dess användning i projektet.

2.1 Motorsystem och styrning

I detta avsnitt behandlas den teori som senare tas upp i rapporten för hur motorer kan väljas, styras och regleras.

2.1.1 Modell för beräkning av motorstorlek

För att få en uppfattning om vilket moment motorn måste leverera kan en enkel modell bestående av ett sluttande plan användas. Modellen kan användas för att beskriva motorns svåraste uppgift, start från stillastående i uppförsbacke. Modellen är enkel men förväntas vara tillräcklig för uppgiften. Modellen tar inte hänsyn till exempelvis massströghetsmoment eller rullmotstånd i gräs, utan valet av motor görs genom att välja en motor som är starkare än det förenklade fallet.[10]

I figur 2.1 delas tyngdkraften upp i komposanter där den ena är parallell med planet och den andra kraften är vinkelrät mot planet. Den kraft som är intressant i det här fallet är den kraft som vill få hjulet att rulla ner för backen. Den andra kraften längs med planet är friktionskraften f.

Genom att summera krafterna i planets riktning fås

$$\sum F_x = m \cdot a = f - m \cdot g \cdot sin(\alpha) \tag{2.1}$$

Sambandet mellan friktionskraften f och momentet T är

$$T = f \cdot R \tag{2.2}$$

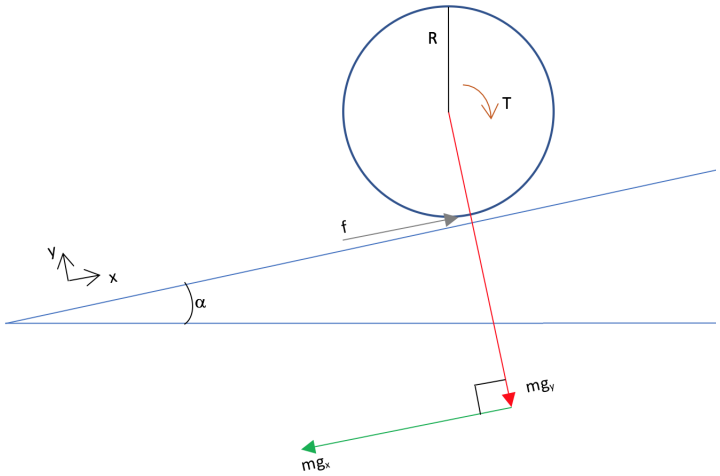
där R är radien på hjulet.
Lös ut f och sätt in i ekvation (2.1) som ger

$$m \cdot a = T/R - m \cdot g \cdot sin(\alpha) \tag{2.3}$$

Lös sedan ut momentet T

$$T = R \cdot m \cdot (a + g \cdot sin(\alpha)) \tag{2.4}$$

Det beräknade momentet skall sedan fördelas över antalet motorer som används för att driva klipparen. Det ska noteras att denna modell förutsätter att hjulen inte slirar.



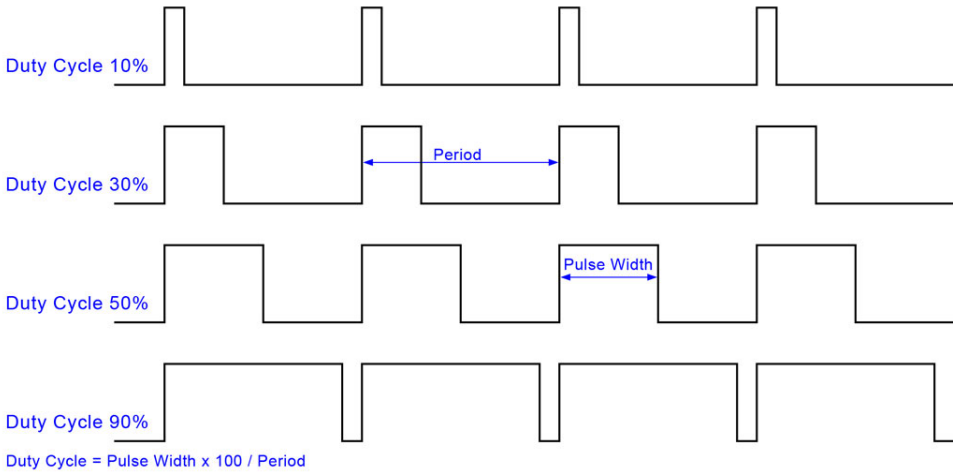
Figur 2.1: Modell för att beräkna vilket moment som behövs för att driva roboten uppför en sluttning med vinkel α

2.1.2 Pulsbreddsmodulering (PWM)

Pulsbreddsmodulering är en slags digital signal som enkelt kan beskrivas som en strömbrytare som mycket snabbt växlar mellan "av och på"[11].

PWM fungerar på så sätt att den kontinuerligt varierar effektmatningen genom att slå på och av spänningen. Detta måste ske betydligt snabbare än vad den anslutna apparaten kan urskilja för att inte störa apparatens utförande. En PWM-signal består i huvudsak av två parametrar, cykeltid och tillslagstid. Tillslagstiden beskriver hur mycket signalen är i påslaget tillstånd som en procentandel av den totala tiden det tar för en cykel att fullbordas. Cykeltiden avgör hur snabbt en PWM-signal fullbordar en cykel och därmed hur snabbt det växlar mellan det höga och låga tillståndet. Genom att använda dessa snabba pulser för att driva apparater visar det sig att utsignalen beter sig som en konstant lägre spänning[12]. Motorn uppfattar konstant spänning tack vara att den är av induktiv karaktär. Detta gör att genom att ändra på de två parametrarna kan apparater styras på olika hastigheter och i idealfallet är moment proportionellt mot ström och spänning proportionellt mot varvtal[13].

Figur 2.2 visar hur arbetscyklers intensitet ändras då längden på tillslagstiden ändras. Det vill säga att ju längre tillslagstid desto högre varvtal. Det går också att se att periodtiden är konstant i samtliga fall.



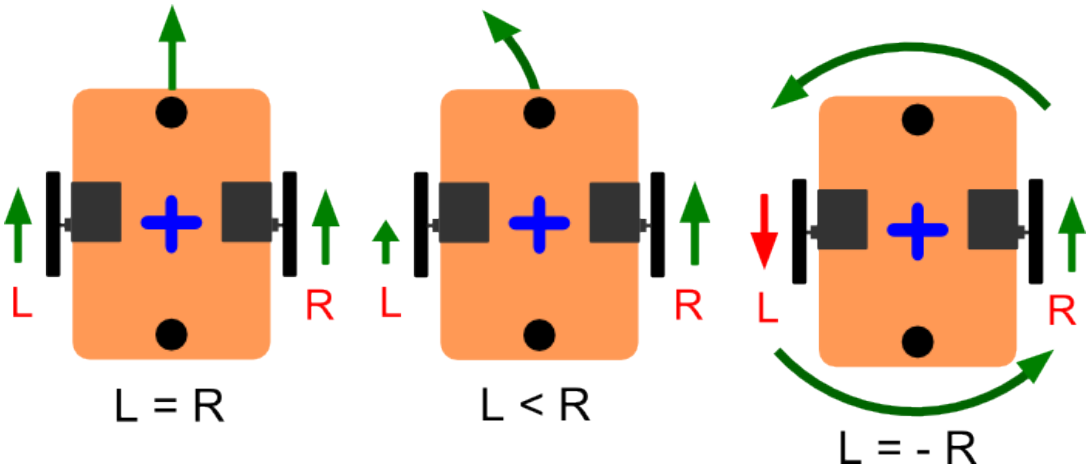
Figur 2.2: Figuren visar förhållanden mellan olika intensiteter i arbetscykler[14]

2.1.3 Differentiell styrning

Differentiell styrning består av två hjul som är fästa på två motorer. Denna typ av styrning innebär att motorerna drivs oberoende av varandra. Genom att köra båda hjul lika fort åker roboten rakt fram och om hjulen drivs med olika hastighet så svänger roboten. Detta medför att om hjulen roterar åt motsatta håll kan roboten i princip rotera på stället. Robotar med denna styrning kan därmed göra mycket skarpa svängar[15].

Differentiell styrning används ofta på robotar då styrningen anses vara mycket lätt att applicera och använda[16]. Det kan dock vara svårt att få roboten att köra rakt fram då motorerna ofta går olika trögt samt att de drivs oberoende av varandra. Detta går däremot att lösa genom användning av återkoppling som hela tiden reglerar motorernas varvtal, genom att variera spänningen över motorerna.

Figur 2.3 illustrerar hur hjulen ska drivas för att köra rakt fram och att styra roboten åt olika håll.



Figur 2.3: Figuren illustrerar användningen av differentiell styrning[17]

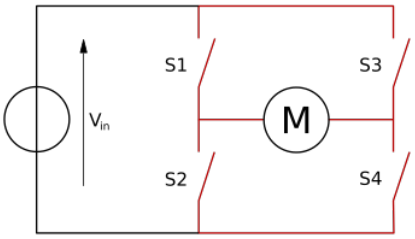
I figur 2.3 visar den vänstra bilden att när hjulen roterar i samma riktning och hastighet kommer roboten att köra rakt framåt eller bakåt. Om hjulen roterar i samma riktning, men i olika hastigheter, kommer roboten att svagt svänga åt det håll där den långsammare motorn sitter. Bilden i mitten visar hur roboten svänger om det högra hjulet roterar snabbare än det vänstra. Roboten svänger då svagt åt vänster. Om hjulen roterar lika snabbt, men åt olika håll, kommer roboten att rotera på stället runt mittpunkten mellan robotens hjul. I fallet på bilden till höger kommer roboten att rotera kring det utmarkerade plustecknet.

2.1.4 Återkopplat system

Ett återkopplat system bygger på att man har ett önskat börvärde på utsignalen. En signal som man sedan mäter och man får signalens ärvärde. Utsignalen som mäts kan vara av olika typ den kan till exempel vara ett varvtal eller en elektrisk spänning. Själva återkopplingen är att börvärdet och ärvärdet jämförs med varandra och ett reglersystem ser till att minska skillnaden mellan dessa. Regleringen kan ske på olika sätt exempelvis proportionellt, integregrering eller derivering, där de olika regleringssätten kan kombineras med varandra efter hur snabbt man vill att systemet ska arbete för att minska skillnaden. De är också olika bra på att kunna ta bort skillnaden helt och hållet för ett visst börvärde då tiden går mot oändligheten.

2.1.5 H-brygga

En H-brygga är en elektronisk krets som ofta används till att möjliggöra drift åt båda riktningar för likströmsmotorer. H-bryggor fungerar på sätt att de ändrar polariteten över en last genom att de möjliggör att en spänning kan läggas på över en last åt båda riktningar[16].



Figur 2.4: Modell över uppbyggnad av H-brygga [18]

Figur 2.4 illustrerar hur en H-bryggas krets är uppbyggd, där den rödmarkerade delen är själva H-bryggan och komponenten markerad med M är lasten, som i detta fall kan ses som en motor. H-bryggan är uppbyggd av 4 transistorer som agerar som strömbrytare (S1, S2, S3, och S4) som antingen kan vara slutna eller öppna. Spänningen över motorn byter polaritet genom att antingen ha S1 och S4 slutna och S3 och S4 öppna eller tvärt om. Det går också att bromsa motorn genom att kortsluta antingen S1 och S3 eller S2 och S4 samtidigt som de andra är avslagna. Den roterande motorn genererar då en spänning som gör att motorn kommer vilja rotera åt det motsatta hållet, vilket gör att den bromsar [16].

2.2 Hinderdetektering

I detta avsnitt behandlas den teori som senare kommer att användas under utvecklingen av gräsklipparens system för att upptäcka hinder.

2.2.1 IR-sensor

IR-sensorer fungerar på så sätt att en ljussensor placeras bredvid en LED-lampa, där båda är riktade åt samma håll. Sensorn är inställd till att detektera en viss våglängd på ljuset i det infraröda (IR) spektrumet. Genom att använda en LED-lampa som sänder ut samma våglängder som ljussensorn är inställd på att detektera, kan intensiteten av vågorna som sensorn tar upp mätas. Detta är möjligt då ljusvågorna från LED-lampan studsar mot objekt som befinnner sig framför sensorn, för att sedan studsa tillbaka mot ljussensorn. Genom att sedan mäta skillnaden i intensitet för ljusvågorna kan avståndet till objektet beräknas [19].

Eftersom ljussensorn tar upp reflekterat ljus från olika objekt har färg på objektet betydelse för sensorns prestation. Då vita objekt reflekterar betydligt mer ljus än svarta, kan ljusvågorna som når sensorn vara för svaga för att den ska kunna uppfatta dem. Detta eftersom svarta ytor absorberar det mesta av energin i ljusvågorna, vilket resulterar i att sensorn kan ha svårt att detektera mörka föremål [20]. Även genomskinliga föremål så som glas kan vara svåra att detektera.

2.3 Positionering

Nedan behandlas teori som kommer att vara väsentlig för positioneringssystemet som behandlas senare i rapporten.

2.3.1 Time of Arrival

Time of Arrival (ToA) är en teknik som går ut på att mäta ankomsttiden för en signal[21]. Signalen sänds från en sändare och tas emot av en mottagare och tiden det tar för signalen att nå mottagaren kallas för ankomsttid. Eftersom signalen har en känd hastighet kan då avståndet mellan sändaren och mottagaren beräknas[22].

Sändaren och mottagaren måste vara synkroniserade i tid för att detta ska vara möjligt. Det går dock att använda en liknande metod om de inte är synkroniserade. Denna metod fungerar på så sätt att en sändare och en mottagare sätts på båda objekten där avståndet emellan vill mätas. Då kan sändare och mottagare mycket enklare synkroniseras på respektive objekt. Sändaren kommer därmed att sända en signal precis då mottagaren tar emot en signal. ToA beräknas sedan genom att mäta tiden det tar för signalen att nå mottagaren och komma tillbaka till sändaren. Avståndet kan nu beräknas.

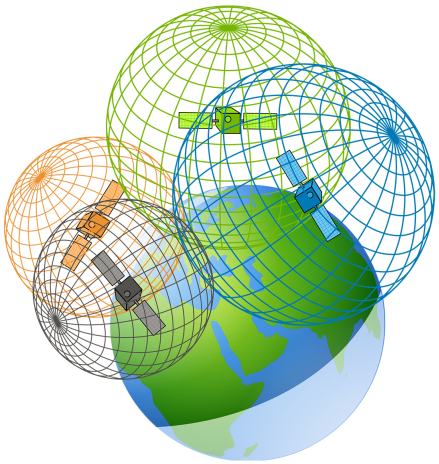
ToA-system använder formel (2.5) för att beräkna distansen. Formeln lyder: hastighet multiplicerat med tid är lika med distans[23].

$$v \cdot t = d \tag{2.5}$$

Då hastigheten av signalen är känd och ankomsttiden mäts genom användning av ToA kan formeln ovan användas för att beräkna avståndet. Om systemet använder metoden där synkroniseringen mellan de olika objekten är besvärlig kan samma formel användas. Enda skillnaden är att eftersom signalen färdas fram och tillbaka blir tiden dubbelt så lång. Därmed divideras ankomsttiden med två och kan då användas i formeln som vanligt.

2.3.2 Trilateration

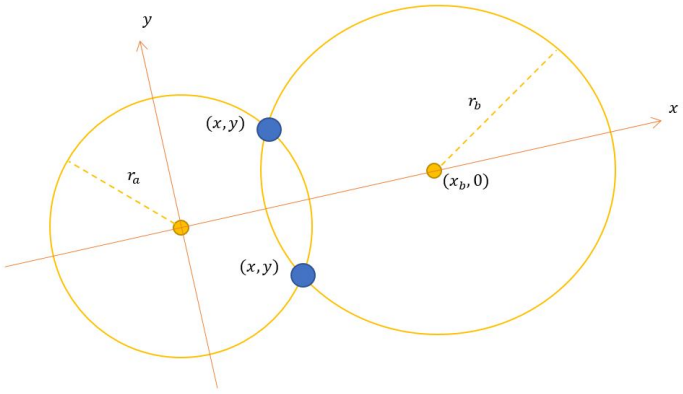
Trilateration är en metod för att bestämma den relativa eller absoluta positionen av en punkt. Den okända positionen ges av skärningspunkten från tre (eller flera) sfäriska ytor. De sfäriska ytorna fås genom att mäta avstånden mellan den okända punkten och ett antal referenspunkter. Trilateration är en mycket vanlig positioneringsmetod som bland annat används av GPS-system. I detta fall mäts avståndet mellan GPS-mottagaren och ett antal satelliter. Då avståndet mellan mottagaren och en satellit är känt kan en tänkt sfär med samma radie som det uppmätta avståndet placeras kring satelliten. Om detta görs för minst tre satelliter kommer GPS-mottagarens position ges av skärningspunkten mellan sfärerna [24]. I figur 2.5 illustreras det hur detta ser ut för fyra satelliter. Viktigt att notera är dock att detta enbart gäller för ett perfekt scenario. Då mätningarna inte är helt exakta kommer inte de tre (eller flera) sfärerna få en gemensam skärningspunkt [25].



Figur 2.5: Illustration över GPS med trilateration [26]

I följande stycke beskrivs hur metoden fungerar teoretiskt, exemplet som tas upp är i två dimensioner, men fungerar på liknande sätt i tre dimensioner.

Först introduceras formeln för två cirklar som korsar varandra, se figur 2.6. Detta kan liknas med att mottagaren bara har kontakt med två satelliter. Ett lokalt koordinatsystem skapas med origo i den ena cirkelns centrum och x-axeln i riktning så att den går igenom den andra cirkelns centrum.



Figur 2.6: Trilateration med två referenspunkter

Ekvationerna för de båda cirklarna fås då enligt ekvation (2.6) och (2.7)

$$r_a^2 = x^2 + y^2 \tag{2.6}$$

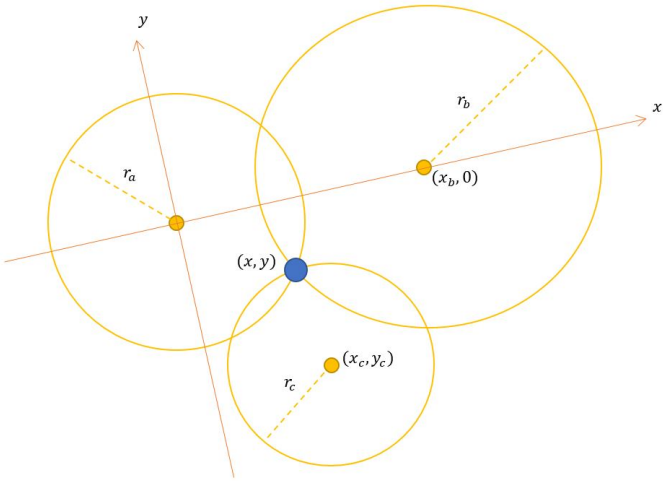
$$r_b^2 = (x - x_b)^2 + y^2 \tag{2.7}$$

Där x_b x-koordinaten i det lokala koordinatsystemet som introducerades och r_a, r_b är respektive cirkels radie. Om dessa två ekvationer kombineras fås koordinaterna för de två skärningspunkterna som

$$x = \frac{r_a^2 - r_b^2 + x_b^2}{2x_b} \tag{2.8}$$

$$y = \pm\sqrt{r_a^2 - \left(\frac{r_a^2 - r_b^2 + x_b^2}{2x_b}\right)^2} \tag{2.9}$$

I detta scenario finns det alltså två möjliga positioner där den okända punkten kan befinna sig (markerade med blått i figur 2.6). För att lösa denna tvetydighet adderas en tredje cirkel (referenspunkt), se figur 2.7.



Figur 2.7: Trilateration med tre referenspunkter

Ekvationen för den nyligen introducerade cirkeln fås då som

$$r_c^2 = (x - x_c)^2 + (y - y_c)^2 \tag{2.10}$$

Där x_c och y_c är koordinaterna för den nya cirkeln. Vidare kan nu ekvationerna för de tre cirklarna (2.6), (2.7) och (2.10) kombineras för att få den gemensamma skärningspunkten som

$$x = \frac{r_a^2 - r_b^2 + x_b^2}{2x_b} \tag{2.11}$$

$$y = \frac{x_c^2 + y_c^2 + r_a^2 - r_c^2 - 2xx_c}{2y_c} \tag{2.12}$$

Då denna metod bygger på att mätningarna är exakta måste den i allmänhet kompenseras på något sätt vid praktisk användning[25].

2.4 Ruttplanering

I detta avsnitt presenteras de ruttplaneringsalgoritmer som har influerat de algoritmer som togs fram under projektet.

2.4.1 Online-Search-Agents

När en autonom robot ska arbeta i ett område kan det vara bra att känna till dess omgivning, men i många fall vet inte den autonoma roboten hur dess omgivning ser ut. Om området dessutom inte är statiskt så kan det hända att objekt flyttas i ett område där roboten ska köra. Detta medför att roboten måste kunna anpassas till det nya området. En lösning är att använda sig av "Online-Search-Agents" algoritmer.[27] Syftet med dessa algoritmer är att de ska kunna användas för att utforska okända områden och därefter lära sig området den just utforskat. Roboten lär sig själv hur den ska anpassa sig till det nya området utan hjälp.

2.4.1.1 Online-DFS-Search

Det finns väldigt många olika algoritmer för hur ett område ska utforskas. Några av de mest använda är Breadth-First-Search (BFS) samt Depth-First-Search (DFS), men dessa är tillämpade för områden som redan känns till. DFS-algoritmen fungerar på följande vis: Givet en startpunkt, en nod, vill algoritmen besöka alla sina grannoder och sedan göra samma sak för alla de grannoder man besöker. Algoritmen registrerar vilka noder som är besökta, så att dessa inte besöks igen. Algoritmen börjar med att gå till en grannod och för den grannoden så väljer den att göra samma sak för dess grannar. När den senare inte har några grannoder kvar, så går den tillbaka till föregående nod och besöker en av den föregående nodens granne som är obesökt. Detta upprepar sig tills den har besökt alla noder.

```
1 DFS (Graph, vertex) : % where vertex is start of the search;
2 Stack =  Stack.append(vertex) for every vertex u:    set visited[u] = unvisited
    while Stack is not empty do
3   |  node = Stack.pop() if node is not visited then
4   |  |  visited[node] = visited
5   |  |  for unvisited adjacent in node do
6   |  |  |  Stack.append(adjacent)
7   |  |  end
8   |  end
9 end
```

Algorithm 1: En simpel Depth-First-Search psuedokod som använder sig utav en stack [28]

Online-DFS-Search, jämfört med DFS, behöver inte ha någon fördefinierad graf eller karta på området. Det är för att syftet med algoritmen är att utforska ett oupptäckt område och sedan producera en karta av området.

```
1  Online-DFS-Agent(s') returns      : an action;
   Input             : s', a percept that identifies the current state
   persistent        : result, a table indexed by the stat and action, initially empty
                       untried, a table that list for each state, the actions not yet
                       tried unbacktracked, a table that list, for each state, the
                       backtracks not s, a, the previous state and action, initially null
2  if Goal-Test(s') then
   |  returns        : stop
3  end
4  if s' is a new state and not in untried then
5  |  untried[s'].ACTIONS(s')
6  end
7  if s is not null then
8  |  result[s,a] ← s'
9  |  add s to the front of unbacktracked[s']
10 end
11 if untried[s'] is empty then
12 |  if unbacktracked[s'] is empty then
   |  |  returns      : stop
13 |  else
14 |  |  a ← an action b such that result[s',b] = Pop(unbacktracked[s'])
15 |  end
16 else
17 |  a ← Pop(untried[s'])
18 end
19 s ← s'
   returns           : a
```

Algorithm 2: " An online search agent that uses depth-first exploration. The agent applicable only in state spaces in which every action can be 'undone' by som other action" [27, sida.150]

2.4.1.2 LRTA*

Learn Real Time A-star är också, som Online-DFS-Search, en form av Online-Search-algoritm. Precis som Online-DFS-Search så skapar den en karta av området den har besökt. Skillnaden ligger i att LRTA* alltid väljer den mer gynnsamma vägen som sitt nästa val istället för att blint söka sig djupt. Den kan göra det genom att lära sig av området den har kört på innan.

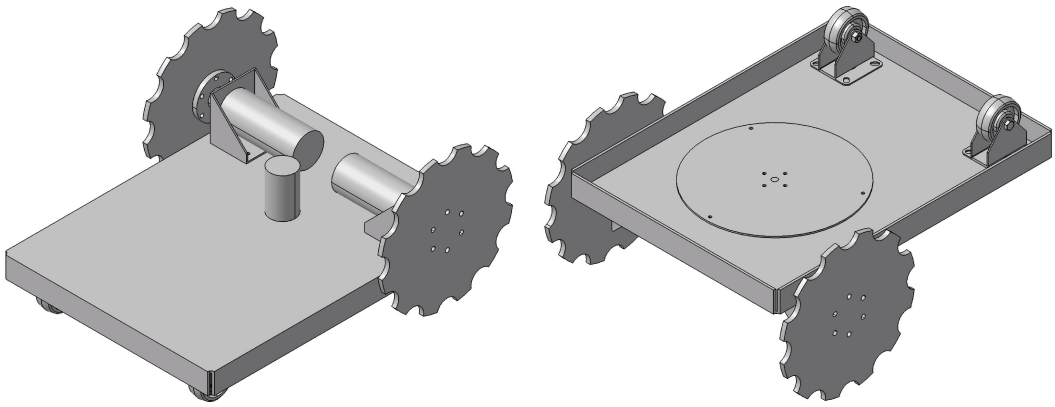
```
1  LRTA*(s') returns      : an action;
   Input          : s', a percept that identifies the current state
   persistent     : result, a table indexed by the state and action, initially empty
                    H, a table of cost estiates index by state, initially empty s, a,
                    the previous and action, initially null
2  if Goal-Test(s') then
   |  returns     : stop
3  end
4  if s' is a new state and not in H then
5  |  H[s'] ← h(s')
6  end
7  if s is not null then
8  |  result[s,a] ← s'
9  |  H[s] ← min LRTA*-Cost(s,b,result[s,b],H)
10 |     b € Actions(s)
11 end
12 a ← an action b in ACTIONS(s') that minimizes LRTA*-Cost(s',
     b,result[s',b],H)
13 s ← s'
   returns      : a
```

Algorithm 3: "LRTA*-Agent selects an action according to the values of neighboring states, which are updated as the agent moves about the state space" [27, sida.152]

3
Konstruktion och tillverkning

3.1 Gräsklippare

Gräsklipparen som konstruerades och tillverkades består av många olika komponenter. Varje komponent kan i sin tur delas in i någon av grupperna: chassi, drivning och styrning, samt klippaggregat. I följande avsnitt beskrivs hur dessa delar konstruerades och tillverkades. I figur 3.1 visas en CAD-modell av prototypen.



Figur 3.1: CAD-modell av prototypen

För en verklig bild av gräsklipparen, se avsnitt 3.1.6. Här presenteras också de ingående komponenterna i gräsklipparen.

3.1.1 Chassi

Efter en del överväganden bestämdes det att chassit skulle konstrueras utan sidoväggar och tak. Anledningen till detta var att det ansågs onödigt att (åtminstone till en början under utvecklingen) ha höga sidoväggar på gräsklipparen, då dessa skulle göra de inre delarna av gräsklipparen mer svåråtkomliga. Istället konstruerades och tillverkades en bottenplatta (ritning i Appendix A.2) där de utstickande kanterna bockades ner. Därav erhölls en skyddande inneslutning av klippaggregatet. Bottenplattan skars ut i vattenskärare och tillverkades i 3 mm aluminiumplåt då hög styvhet eftersträvades och det var den grövsta aluminiumplåt som fanns att tillgå i prototyplabbet. Under tillverkningen av bottenplattan skars det också ut hål för montering av klippaggregatet. Dimensionerna för chassit sattes främst med avseende på att klippaggregatet var tvunget att omslutas, men det beaktades också

att framhjulen skulle monteras på bottenplattan. Därav sattes längden till något längre än bredden på chassit. Vidare konstruerades två små motorfästen (ritning i Appendix A.3) att fästa på bottenplattan. Dessa tillverkades på liknande sätt; de skars ut i vattenskärare och bockades senare till. Materialet som användes var aluminiumplåt med 2 mm tjocklek. Motorfästena monterades på bottenplattan med skruvförband. Det faktiska chassit och motorfästena finns att se i figur 3.5.

3.1.2 Enkortsdatorer

För att roboten skulle kunna ta sig fram utan att köra in i hinder, för att kunna navigera, för att göra olika beräkningar och för att kunna ge feedback behövdes enkortsdatorer. För de "enklare" uppgifterna, som att kommunicera med motorerna och sensorerna, ansågs en Arduino UNO vara det bästa valet. Arduino är mer specifikt ett mikrokontroller-kort som anses vara mycket simpelt att använda och fungerar på så sätt att den kör ett program om och om igen. Detta tillsammans med att Arduino är bra på realtid och analoga system gör att den fungerar utmärkt för enklare uppgifter, som att skicka feedback från sensorer och hantera PWM-signaler. Arduino UNO-kort är dessutom förhållandevis billiga i jämförelse med andra mikrokontroller-kort [29].

Vidare förutsattes det att det skulle krävas en något starkare enkortsdator för att sköta positioneringssystemets beräkningar. Valet föll på en Raspberry Pi 3. Raspberry Pi är ett avskalat datorsystem som oftast använder operativsystemet Linux. Med Linux kan den hålla igång flera program samtidigt, vilket är mycket bra eftersom den då kan köra flera beräkningar samtidigt. I och med dess relativt höga processorprestanda lämpar sig dessutom Raspberry Pi betydligt bättre än Arduino för större beräkningar [30], vilket var viktigt för prototypen.

Genom att använda både en Raspberry Pi och en Arduino erhölls både enkelheten som fås med en Arduino och prestandan som fås med Raspberry Pi. De två korten kopplades samman med USB för att kunna kommunicera med varandra. I kommunikationen fungerar Raspberry Pi:n som huvudkort och skickar kommandon till Arduinon, som i sin tur skickar realtidsdata från sensorerna tillbaka till Raspberry Pi:n.

3.1.3 Batteri

Elmotorerna och enkortsdatorer som finns på klipparen behöver en energikälla för att kunna förflytta sig. Valet föll på batterier då det är ett enkelt och lättillgängligt val av energikälla. Valet av batterier skedde väldigt tidigt i konstruktionen av klipparen och begränsade i sin tur valet av motorer efter tillgänglig spänning. Anledningen till att batterier valdes innan övrig elektronik var för att möjligheten fanns för att låna Balanduinobatterier från institutionen. Batterierna var dock tvungna att passa med projektets ändamål. Informationen om batterierna var begränsad. Då det information som fanns tillgängligt var det som stod på batteriet och återfinns i tabell 3.1.

Tabell 3.1: Data för Balanduino-batteri av LiPo-typ

Parameter	Värde
Kapacitet	4200 mAh
Nominell spänning	11,1 V
Urladdningsström	$20 \cdot C$ A

Batteriet är av LiPo-typ vilket är en fördel genemot olika nickel baserade motsvarigheter gällande vikt och kapacitet [31]. LiPo har dock kortare livslängd gällande laddningscykler. Ett LiPo-batteri består av en eller flera likadana battericeller. En LiPo-cell har en nominell spänning på 3,7 V vilket visar att Balanduino-batteriet består av tre st seriekopplade celler. Liksom på andra batterier förändras spänningen som batteriet kan ge med urladdningsgraden och ett tre cells LiPo-batteri kan ligga i spannet 9-12,8 V [31]. Kapaciteten hos batteriet innebär att batteriet kan leverera 4,2 Ah. Till exempel 1 A under 4,2 h eller 4,2 A under en 1 h. Mer intressant är däremot urladdningsströmmen som enligt [31] beräknas utifrån

$$\text{Urladdningsström} = 20 \cdot C \tag{3.1}$$

Där C är kapaciteten i ampere. Detta ger

$$\text{Urladdningsström} = 20 \cdot 4,2 \text{ A} = 84 \text{ A} \tag{3.2}$$

Denna urladdningsström är den ström batteriet maximalt kan ge ifrån sig utan att ta skada. Denna ström som batteriet klarar av att leverera som max är summan av alla strömmar för motorer, sensorer och enkortsdatorer. De inkopplade enheterna till batteriet måste också hålla sig en bra bit under denna ström om driftstiden ska vara någorlunda bra då kapaciteten är begränsad.

3.1.4 Drivning och styrning

Detta avsnitt beskriver gräsklipparens konstruktion för att förflytta sig. Därför tas valet av drivmotorer upp i nästkommande avsnitt, för att sedan beskriva motorernas styrning. Avsnittet avslutas sedan med en beskrivning av de hjul som togs fram.

Gräsklipparen konstruerades med fyra hjul. Alternativet att istället ha tre hjul övervägdes, men det antogs att en konstruktion med fyra hjul skulle bli stabilare. Vidare konstruerades gräsklipparen med två motorer (en vid varje bakhjul) för drivning och differentiell styrning. Valet av antal hjul och styrning gjordes utifrån det faktum att många andra robotgräsklippare använder sig av samma teknik, exempelvis Ardumower[32] och modeller tillverkade av Husqvarna[7]. Dessutom ansågs det vara en lösning som skulle vara enkel att implementera på prototypen.

3.1.4.1 Drivmotorer

I projektet valdes det att arbeta med borstade likströmsmotorer för att de är relativt billiga. En annan viktig anledning är att det finns enkla billiga styrkort förberedda för “differential drive” som tas upp i nästa avsnitt.

Borstade motorer är framförallt enklare och billigare att styra än borstlösa likströmsmotorer[33], [34]. Nackdelen med borstade motorer är att de däremot har slitagedelar och högre tröghetsmoment, men det är inget som är en nackdel för att projektet ska kunna konstruera prototypen och utföra tester[35]. Då konventionella borstade likströmsmotorer har högt varvtal krävs att de har en växellåda för att få ett varvtal lämpligt till storleken på hjulen. Typen av växellåda i den motor som ska väljas ska vara av typen planetväxel. Detta då då denna typ av växel är mer robust och tål högre moment och tvärkrafter jämfört med en kuggväxel[36].

Valet av motor för drivning gjordes enligt beräkningarna i teori- avsnittet "Modell för beräkning av motorstorlek" 2.1.1. Där parametrar för den färdiga modellen antogs enligt tabell 3.2 för att få reda på vilket moment motorn ska klara av att ge.

Tabell 3.2: Tabell med antagna värden på parametrar för att beräkna vilken motorkapacitet som krävs

Parameter	Värde
Massa, m	10 kg
Hjulradie, R	0.05 m
Lutning, α	10°
Hastighet, v	0.5 m/s
Acceleration, a	0.2 m/s^2

Den tilltänkta massan approximerades från flera liknande projekt och att klipparen skulle tillverkas som en fyrkantig aluminiumlåda. Radien på hjulen sattes efter vad gruppen ansåg var ett rimligt värde innan tillverkning. Önskad hastighet valdes först genom approximation av observationer av olika robotgräsklippare och bekräftades av [37]. Från [37] sattes också accelerationen till ungefär halva hastigheten av klipparen. En lutning uppför lades också till för att kompensera för den förenklade modellen och garantera att prototypen skulle klara av plattmark eller förändringar i de antagna parametrarna.

Beräkningar med dessa parametrar resulterade i att det skulle krävas ett moment större än 0,48 Nm per motor. Varvtalet för den växlade motorn ges från sambandet

$$\omega = \frac{v}{r} \cdot \frac{60}{2\pi} \approx 95 \text{ rpm} \tag{3.3}$$

Motorn som valdes var en borstad 12 V planetväxlad likströmsmotor från Lynxmotion. Motorns prestanda presenteras i tabell 3.3

Tabell 3.3: Data för vald motor från Lynxmotion [38]

Parameter	Värde
Utväxling, i	$26,9:1$
Nominellt varvtal, ω	105 rpm
Moment, max verkningsgrad, $T_{max\ effiency}$	0,70 Nm
Nominellt Moment, T	3,53 Nm
Nominell spänning U	12 V
Nominell ström I	10 A
Ström, Max verkningsgrad $I_{max\ effiency}$	1,61 A

Motorn som valdes har som synes ett högre moment vid maximal verkningsgrad än det beräknade för att kompensera för den förenklade modellen. Önskad hastighet kan sedan varieras med styrkortet för motorn.

3.1.4.2 Styrkort

För att styra motorn valdes att köpa ett färdigt styrkort istället för att tillverka ett eget. Anledningen till detta var för att spara tid samt andra fördelar med det inköpta kortet som inbyggd elektronik som skydd för djupurladdning av batteri och höga strömmar. Kortet som köptes var ett Sabertooth Dual 2×12 A [39] som klarar av kontinuerliga strömmar på just 12 A per kanal. Anledningen till att detta kort valdes var på grund av den höga nominella strömmen i drivmotorn. Drivmotorns ström ska aldrig ligga en längre tid på en så hög ström om det inte uppstår fel. 12 A kortet valdes därför lite av försiktighet då steget nedåt var ganska långt och skulle kunna hamna nära driftsströmmen. Det här kortet är byggt så att det har en funktion för "Differential drive" avsnitt 2.1.3 som gör att den är förberedd för denna typ av styrning. Den utför styrningen med hjälp av en H-brygga avsnitt 2.1.5. Kortet har också regenerativ laddning vilket innebär att batteriet laddas upp då klipparen körs i nedförsbacke eller vid inbromsning. Vilket har möjlighet att förlänga batteritiden. I fallet för just gräsklippare kanske det är marginellt märkbart men kan vara praktiskt i andra fall men det kontrolleras inte i denna rapport.

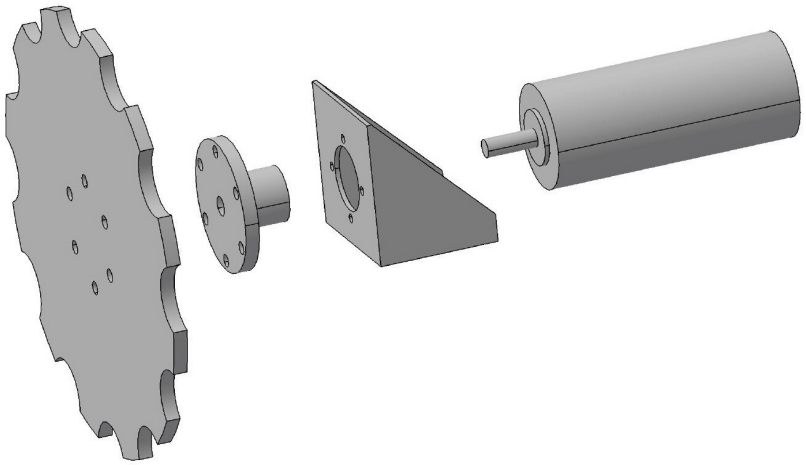
Tillsammans med styrkortet köptes även en PID-kontroller-krets Kangaroo x2 [40] för att spara tid för att lägga på ultraljudskretsen. PID-kontroll är ett återkopplat reglersystem som går att läsa om i avsnitt 2.1.4. I avsnittet "Differential drive" 2.1.3 nämns att svårigheten med "Differential drive" är att få roboten att köra rakt. Kretsen tar därför in varvtalet som en återkoppling och talar om för Sabertooth-kortet om det ska leverera högre eller lägre spänning för att reglera varvtalet till önskat värde. Kangaroon känner automatiskt av dynamiken i systemet då ett testprogram körts. Därefter kan PID-kretsen se till att hjulen roterar lika fort och med en bestämd hastighet.

Systemet sätts ihop genom att Kangaroo kortet kopplas ihop med Sabertooth kortet. Kangaroo kortet har möjlighet att ta in olika typer av styrningar från användaren. Som tidigare nämnts tar Kangaroo kortet in varvtalet som återkoppling. För kommunikation med styrdatorn används seriell-kommunikation mellan Arduino

och Kangaroon. Med seriell-kommunikation och PID-kretsens Arduino-bibliotek kan användaren via Arduinon skicka signaler som “kör framåt x cm”, eller “sväng y grader” [40]. Sabertooth-kortet tar in batterispänningen och styr via PWM (2.1.2) ut en pulsmodulerad signal på de två utgångarna till motorerna för att variera varvtalet. En uppkoppling av systemet visas i Appendix D.1.

3.1.4.3 Hjul

De främre hjulen köptes in med åtanke på total höjd, samt att de skulle kunna rotera fritt. Valet föll på ett par länkhjul med en höjd som stämde väl överens med bakhjulen. Länkhjulen fästes på undersidan av chassit med skruvförband. Konstruktionen av bakhjulen krävde mer arbete. För att enkelt kunna fästa bakhjulen på motorerna och samtidigt hålla ned kostnaderna tillverkades egna hjul i Chalmers prototyplabb. Hjulens design inspirerades av Ardumower, alltså tunna diskar med ett mönster i form av halvcirklar. Ritningen för de hjul som togs fram kan ses i Appendix A.1. Detta mönster antogs ge hjulen ett tillräckligt bra grepp för att kunna driva gräsklipparen på en gräsmatta. Hjulen, som även de är i 3 mm aluminiumplåt, skars ut i vattenskärare och fästes sedan var och ett på ett nav med fyra skruvar. Hålen på hjulen anpassades efter de inköpta naven. Varje nav kunde sedan enkelt fästas på motorns D-formade axel med en stoppskruv. Denna enhet monterades på chassit med hjälp av tidigare nämnda motorfästen. I figur 3.2 visas hur ett hjul monteras tillsammans med nav, motorfäste samt motor.



Figur 3.2: Modell av ett hjul och hur det monteras med en motor

3.1.5 Klippaggregat

Klippaggregatet som tillverkades består av en motor, vars axel är monterad på en disk med ett nav emellan de två delarna. På disken fästes sedan de inköpta robotgräsklipparknivarna med medföljande skruvar, se figur 3.3. En ritning för disken finns under Appendix A.4. Designen med disken och knivarna är utformad så att knivarna slås “inåt” mot disken om de slår emot något hårt. Knivarna är alltså inte helt fixerade av den medföljande skruven, utan kan rotera kring den. Denna konstruktion finns till för att knivarna endast ämnas klippa gräs, och inte andra saker

som råkar hamna under gräsklipparen. Denna lösning inspirerades av andra robotgräsklippare som Ardumower[32] och Husqvarna Automower[7]. Disken tillverkades av 2 mm aluminiumplåt som skars ut i vattenskärare i prototyplabbet.

Figur 3.3: Robotgräsklipparkniv med medföljande skruv

Navet som förbinder motorn med den roterande disken tillverkades på egen hand i prototyplabbet utifrån ritningen som kan hittas under Appendix A.5. Navet tillverkades i aluminium för att bibehålla en så låg vikt som möjligt, och formades med svarv och borr. Navet fästes i disken med skruvförband, och motorns axel fixerades i navet med en stoppskruv. I figur 3.4 visas motorn, navet och klippdisken, samt hur de monteras ihop.

Figur 3.4: Modell av klippaggregatet

3.1.5.1 Motor för klippning

Motor för klippagregat valdes genom att titta på andra liknande projekt. Här utfördes inga momentberäkningar då dessa hade krävt att gruppen utförde tester eller göra uppskattningar för vilket moment som friktionen mellan disken och gräset skulle bidra med. Bidrag till vilket moment motorn skulle behöva klara av består i huvudsak av tröghetsmoment vid start av disken och friktionen mot gräset samt dess egna tröghetsmoment. Efter att ha diskuterat med Göran Stigler i prototyplabbet samt kollat på Huskvarnas lösning som består av två roterande diskar där den ena sitter fast på motor axeln och den andra sitter fast i den första disken via kullager för att

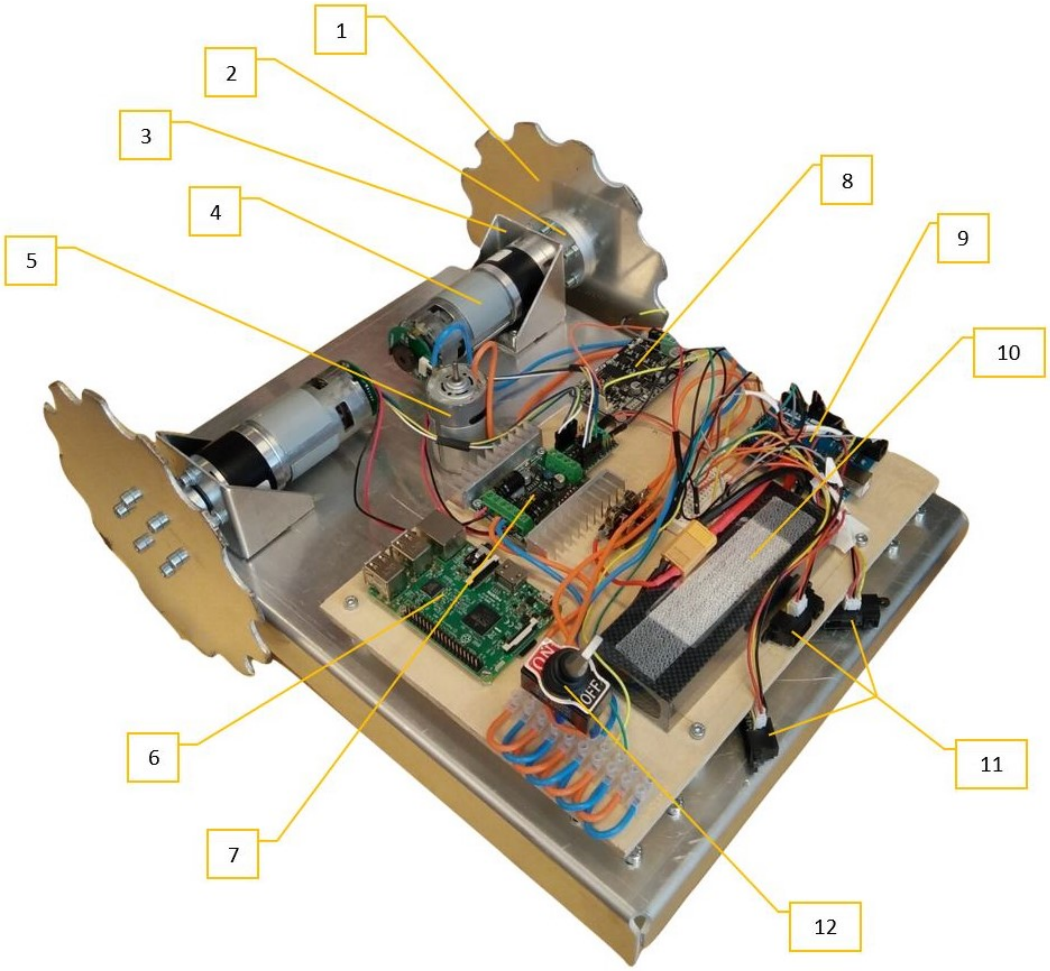
rotera fritt. Detta för att minska friktionen mot gräset som ska vara väldigt märkbar enligt Stigler. Denna lösningen ser till att moment blir lägre vilket är bra för en kommersiell gräsklippare där batteritiden är viktigt. Moment är nämligen proportionellt mot strömmen enligt [41]. I detta projektet är effektiviteten av klippaggregatet inte huvudmålet utan det är mer intressant om det fungerar.

Enligt [42] bör klipparen ha ett varvtal kring 3300 rpm för att klippa gräset. Vidare använde sig [42] av en motor med 45 mm diameter och 5 mm axel till klingan och motorn kunde leverera 15 W. Vid högre gräs rekommenderas en motor med 60 mm i diameter och därav kraftigare axel [42]. Ardumover-gräsklipparen använder sig av en motor som ligger kring 3000 rpm och har en axel med diametern 8 mm[43], [44].

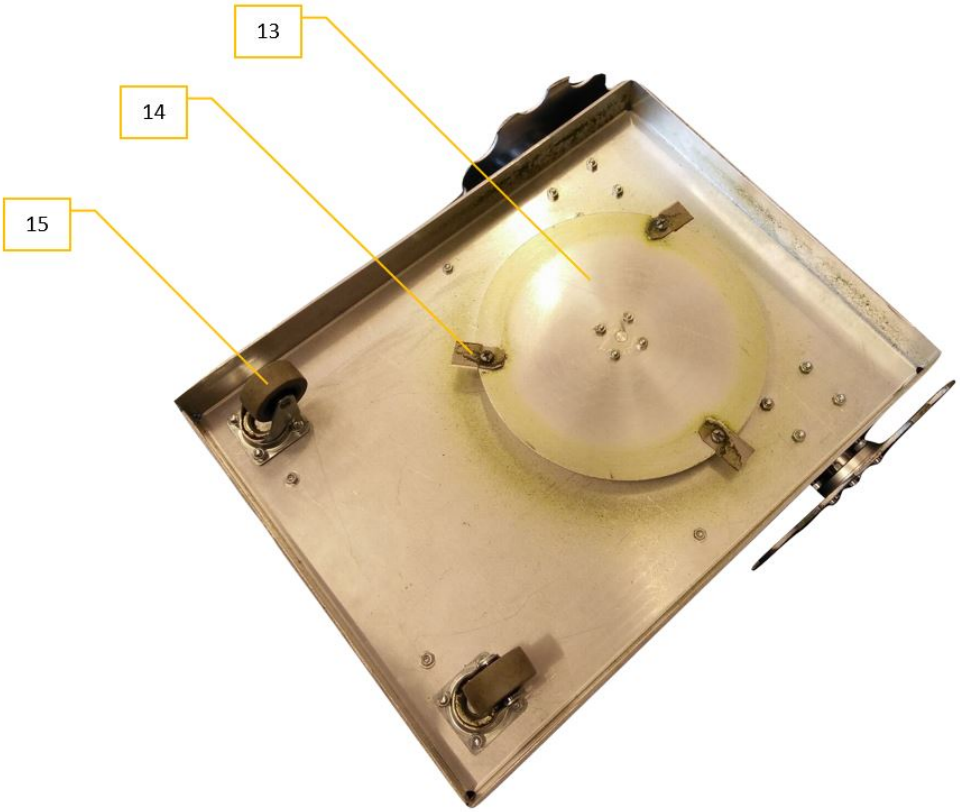
Då komponenterna för motorerna beställdes från en leverantör föll valet av klippmotor på RS-755 som hade specifikation på 5400 rpm, 5 mm axel och effekt på 25,8 W för att den låg bra till i pris [45]. Denna motor är samma som för projektets hjulmotorer fast utan växellåda. Tyvärr var denna motor slut i lager och en mindre RS-555 beställdes istället då den var så pass billig att den kunde beställas för test. RS-555 hade specifikationer enligt 6100 rpm, 3 mm axel och 12 W [46]. Då varvtalet var för högt köptes också ett billigt PWM-styrkort (2.1.2) med H-brygga (2.1.5) för att reglera varvtal och riktning. Riktning är dock inte så intressant i detta fallet. Kortet som köptes var ett Cytron MD10C som är ett kort för att driva en borstad likströmsmotor med en kontinuerlig ström på upp till 13 A [47]. Inkoppling av motorn visas i Bilaga D.1.

3.1.6 Sammanställning

Monteringen av de olika delarna gjordes genomgående med skruvar och skruvförband. I figur 3.5 och 3.6 visas den färdiga robotgräsklipparen, inklusive elektronik.



Figur 3.5: Färdigmonterad robotgräsklippare sedd ovanifrån



Figur 3.6: Färdigmonterad robotgräsklippare sedd underifrån

Notera att klippnavet är skymt i båda dessa bilder, och har därför inte kunnat numreras.

Tabell 3.4: Gräsklipparens komponenter

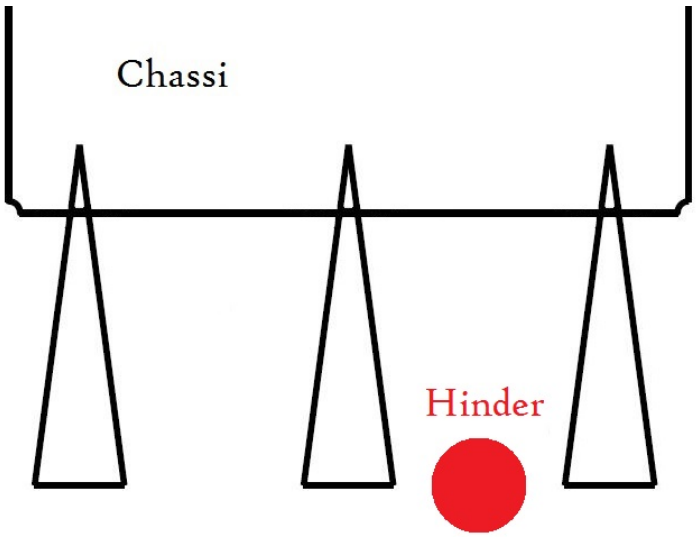
Nr	**Komponent**
1	Hjul
2	Hjulnav
3	Motorfäste
4	Drivmotor
5	Klippmotor
6	Raspberri Pi
7	Sabertooth Dual och Kangaroo x2
8	Cytron MD10C
9	Arduino Uno
10	Batteri
11	IR-sensorer
12	Strömbrytare
13	Klippdisk
14	Kniv
15	Länkhjul
-	Klippnav

3.2 Hinderdetektering

En robotgräsklippare måste kunna detektera hinder för att fungera i den miljö som den är tänkt att vistas i. Detta kan göras med flera olika typer av sensorer. I denna konstruktion används IR-sensorer. Anledningen till detta var att IR-sensorn är en relativt billig sensor, som är enkel att styra med en Arduino. En annan typ av sensor som övervägdes (då den också var billig och enkel att styra med Arduino) var ultraljudsensorn. Ultraljudsensorer har i jämförelse med IR-sensorer, ett större detekteringslängd. Denna fördelaktiga egenskap ansågs dock inte ha någon större betydelse för gräsklipparen då IR-sensorns detekteringslängd på 10-80 cm är fullt tillräcklig. Den övervägande anledningen till att valet föll på IR-sensorn var för att positioneringssystemet arbetar med ultraljud, och en ultraljudsensor skulle då eventuellt kunna störa positioneringssystemet. Vidare togs beslutet att prototypen skulle ha tre stycken IR-sensorer. Detta för att få en bra täckning framåt.

3.2.1 Placering av IR-sensorer

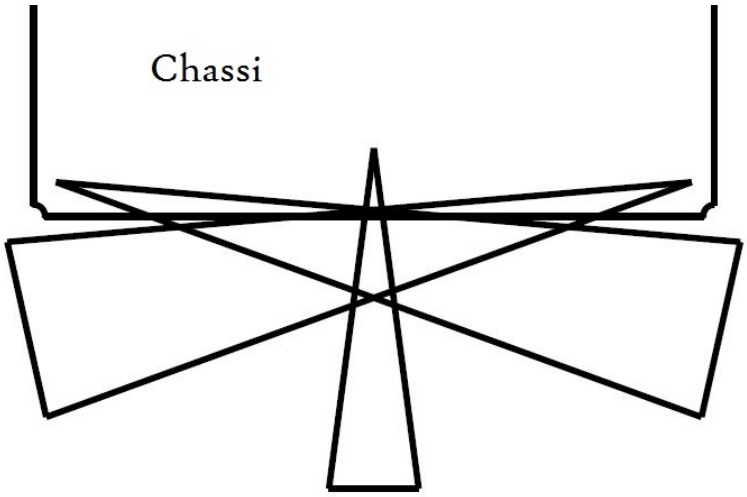
Placeringen av sensorerna krävde en hel del tester och utvärdering. Olika vinklar och positioner testades för att få bästa möjliga resultat. En placering som testades var att sätta en i centrum, en i vänster hörn och en i höger hörn. Här testades dessutom olika vinklar för sensorerna på kanterna, alltifrån att låta dem peka något utåt, till att låta dem korsa varandra. Testerna visade att korsning av sensorerna gav bättre resultat. Det primära problemet som uppstod i annat fall var att smala hinder (exempelvis bordsben) lätt hamnar i en "död vinkel" mellan sensorerna. Detta illustreras i figur 3.7. Anledningen till detta är att sensorernas synvinkel endast är ca 15°. Alternativet för att lösa detta skulle vara att förlänga sensorernas räckvidd, men detta skulle göra att gräsklipparen uppfattar hinder på ett för långt avstånd, och därmed inte lyckas klippa gräset tillräckligt nära.



Figur 3.7: Illustration av "döda vinklar" vid placering av IR-sensorer

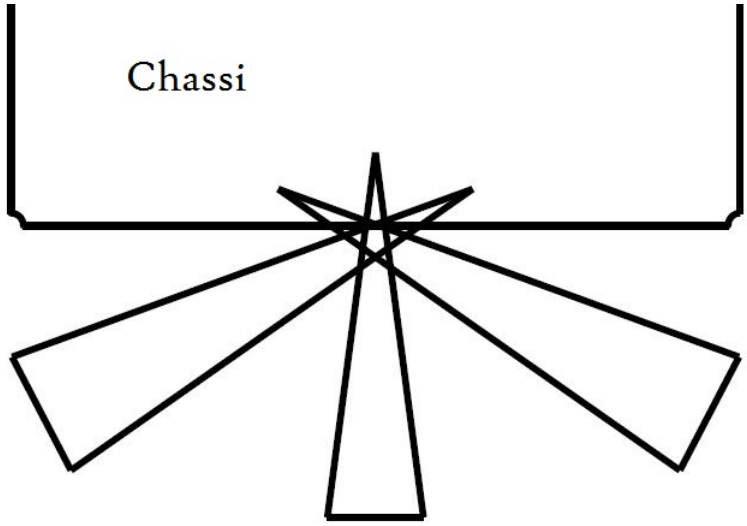
Korsning av sensorerna visade sig som sagt vara ett bättre alternativ då de "dö-

da vinklarna" kunde undvikas. En illustration av detta ses i figur 3.8. Med denna placering uppstod dock ett annat problem, nämligen att om sensorerna på sidorna riktades inåt för mycket så gav de utslag av själva gräsklipparen. Samtidigt kunde de inte riktas allt för långt utåt från gräsklipparen då detta också kräver en längre räckvidd för att "täcka" hela gräsklipparens front.



Figur 3.8: Testad placering av IR-sensorer med korsning

Lösningen på detta, som också blev den slutgiltiga, var att flytta in de två sensorerna från kanterna, samt rikta dem något mer utåt från gräsklipparen. En illustration av detta kan ses i figur 3.9. Den slutgiltiga vinkeln för de två sensorerna på sidorna blev 50° (i förhållande till den mellersta sensorns vinkel).



Figur 3.9: Slutgiltig placering av IR-sensorer

3.2.2 Mjukvara

De tre IR-sensorerna kopplades till den Arduino som finns på gräsklipparen. Arduinon programmerades så att gräsklipparen skulle backa och vända då den stötte på ett hinder. Dessutom skrevs det in en funktion som beräknar medianen av de 25 senaste mätningarna från sensorerna. Anledningen till detta var att sensorerna ibland gav enstaka felaktiga mätningar. Koden för detta kan ses i Appendix C.3

3.3 Positioneringssystem

Detta avsnitt börjar med en motivering av ultraljud, följt av en genomgång av positioneringssystemet. Efter det gås designprocessen av ultraljudskretsen igenom, för att sedan avslutas med algoritmen för trilateration på matrisform.

3.3.1 Motivering av ultraljud

Anledningen till att valet föll på just ultraljud till positioneringssystemet var flera. Dels krävdes en signal som färdas med så hög hastighet att gräsklipparen inte hinner förflytta sig märkvärt innan positionen har beräknats. Samtidigt krävs det att hastigheten är så pass låg att tiden det tar för signalen att gå från sändare till mottagare kan mätas med hjälp av "Time of Arrival". Med dessa två krav kunde signaler som radiosignaler och olika ljussignaler tas ur diskussionen, då dessa färdas för snabbt. Vidare övervägdes användning av Wi-Fi och Bluetooth, men då dessa tidigare visat sig ge en allt för dålig precision (med avvikelser i storleksordningen meter) [48], föll valet på ultraljud. Värt att notera är också att ultraljudsmottagare- och sändare är relativt billiga. Ultraljud används idag främst i applikationer inomhus. En anledning till detta är att ultraljud är temperaturkänsligt, då det antar olika hastigheter för olika temperaturer. En annan nackdel med ultraljud är att det är relativt känsligt för buller [49]. Trots dessa nackdelar ansågs ultraljud vara det bästa tillgängliga alternativet.

3.3.2 Positioneringssystem baserat på ultraljud

I början av konstruktionsprocessen var det tänkt att positioneringssystemet skulle skicka ut en startsignal till samtliga fyrar via radiofrekvens och sedan invänta svar från ultraljud och beräkna positionen. Istället har det blivit så att gräsklipparen frågar om avståndet från en fyr i taget. Detta för att enkelt veta vilket avstånd som mätts istället för att logiskt sortera värden. Systemet består av fyra stycken fyrar bestyckade med en radiofrekvensmottagare (RF-mottagare), en ultraljudshögtalare med tillhörande elektronik och en Arduino UNO. På gräsklipparen sitter ett system som består av en RF-sändare, en ultraljudsmottagare och en Arduino UNO. Principen av hur systemet fungerar blev som följer:

1. Gräsklipparen börjar med att skicka RF-signal till fyrarna och startar samtidigt en tidtagning. Då radiosignaler färdas i ljusets hastighet (300 000 km/s)[50], är tiden RF-signalen färdas försumbar.

2. Fyrarna tar emot RF-signalen. Sedan svarar bara den specifika fyr som var tillfrågad, med en ultraljudsignal. Denna rör sig med ljudets hastighet, vilket är tillräckligt långsamt för att tiden ska kunna mätas. Valet av fyr sker genom att fyren utvärderar meddelandet från RF-sändaren med en if-sats.

3. Ultraljudsmottagaren på gräsklipparen tar emot signalen och stoppar den totala tiden från det att RF-signalen skickades. Tiden det tog kan sedan användas för att beräkna avståndet till fyren enligt teoriavsnittet "Time of Arrival" 2.3.1.

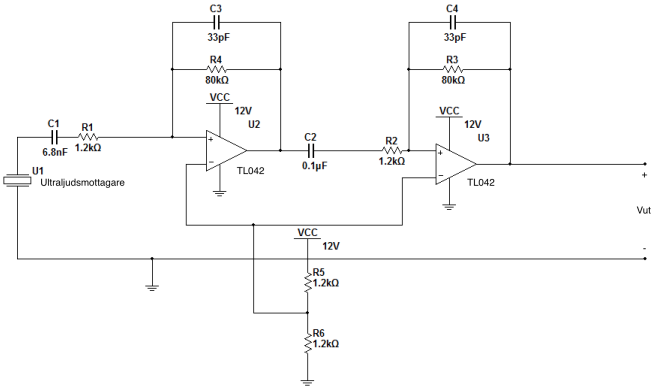
4. Steg $1-3$ upprepas för de andra fyrarna och när fyra resultat är mottagna kan positionen beräknas med trilateration. Detta beskrivs senare i detta kapitel.

3.3.3 Design av ultraljudskrets

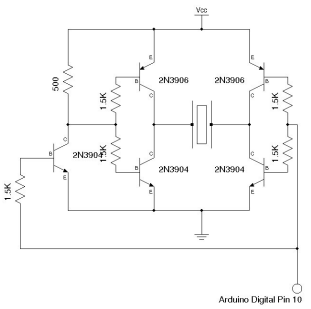
För att designa kretsen undersöktes det mycket om hur andra har gjort tidigare. En vanlig lösning är att använda ultraljud som en "sonar". Det vill säga att den har sändare och mottagare på samma ställe och mäter avstånd till närmsta föremål. Efter att ha kollat på flera olika lösningar märktes flera likheter. Ultraljudsmottagaren producerar, för en ultraljudssignal, en låg sinusformad spänning. Denna spänning behöver då förstärkas och filtreras innan den behandlas för att ta ut den intressanta information ur signalen. Den första lösningen från [51] är en analog lösning som beskrivs i nästa avsnitt.

3.3.3.1 Analog krets

Kretsarna som användes som grund för kretsdesignen (enligt [51]), visas i figur 3.10. Kretsarna designades av komponenter lånade från institutionen för signaler och system för att kunna testa kretsarna. Först kopplades sändarkretsen som visas i figur 3.14b. För att testa den sändarkretsen användes kod från [51] för att skapa en fyrkantssignal med frekvens 40 kHz och 5 V amplitud från Arduinon. Då denna signal kopplades in i kretsen mättes det med oscilloskop att spänningen på signalen vid högtalaren var cirka 8 V och frekvensen densamma som tidigare.



(a) Något omgjord mottagarkrets från [51]. Operationsförstärkare TL042 sitter båda två i samma kapsling



(b) Sändarkretsen uppbyggd som en transistorförstärkare [51]. Den rektangelformade symbolen med två raka sträck på sidorna representerar högtalaren för att skicka ultraljud

Figur 3.10: kopplingsschemat för sändar och mottagarkretsen

För att testa funktionen hos sändaren så kopplades en enkel krets upp med mottagaren i serie med en resistor in till oscilloskopet. Genom att kolla på både sänd och mottagen signal i oscilloskopet gick det att se att amplituden hos mottagen signal förändras med avståndet och amplituden låg i millivoltsområdet.

Mottagarkretsen i figur 3.10a består först av ett aktivt bandpassfilter som i det ideala fallet har 66 gångers förstärkning eller 36 dB och släpper igenom frekvenser mellan 20-60 kHz [52]. Det första filtret är sedan kaskadkopplat med ytterligare ett aktivt bandpassfilter med samma förstärkning men med bandpassfrekvenser mellan 1,3-60 kHz. Eftersom kretsen drivs av ett batteri drivs operationsförstärkarna som "single rail". När konstruktionen sker på detta vis går det att införa en virtuell jord, "virtual ground". Den virtuella jorden läggs till hälften av V_{cc} genom två lika stora resistanser som kopplats till den icke-inverterade ingången på operationsförstärkaren som i figur 3.10a. Den virtuella jorden skapar en ny nivå som utgör en nollnivå som gör att det går att producera en "negativ" spänning med en positiv spänningskälla [53]. Batteriet för med sig mer begränsningar när operationsförstärkare används. Batteriet som används är på 12 V och flera operationsförstärkare tål att matas med 30 V uppdelat i ± 15 V även om de kan matas med lägre spänning. Storleken på spänningsintervallet avgör hur stor upplösning det går att få för en viss insignal, exempelvis om insignalen har en amplitud på 0,1 V och avbildas på ett intervall som är 12 V eller 30 V blir en liten förändring av insignal olika stor på på de olika intervallen.

Efter det att kretsen i figur 3.10a satts ihop utfördes tester då ultraljudssända-

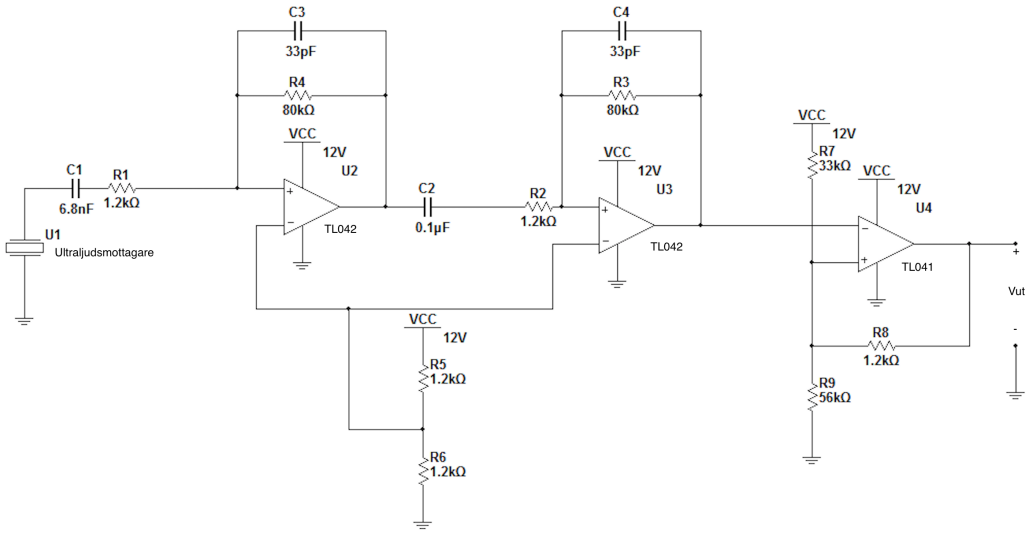
ren sände en kontinuerlig signal på samma sätt som tidigare. På oscilloskopet gick det då att observera att kretsen fungerade och förstärkte signalen. Det gick också se att på korta avstånd var förstärkningen så hög att signalen mättades. Det vill säga att den sinusformade spänningens toppar kapades av. I detta skedet bedömdes det att det inte påverkar testerna på ett negativt sätt. Det går att anpassa värden på de olika komponenterna för att få en annan förstärkning och undvika mättnad senare om det skulle behövas.

Genom att placera sensorerna och mäta på ett avstånd som undviker mättnad fortsatte testerna. För att kunna tillämpa "time of arrival" (avsnitt 2.3.1) behöver det skickas en kort signal och sedan mäta tiden tills den tas emot. Denna korta puls skapas på samma sätt som den kontinuerliga signalen som användes tidigare fast under en kort period kring 200-300 µs vilket vid 40 kHz ger 8-12 perioder. Stommen till Arduino-koden kom ifrån [51]. Den fungerade så att ultraljudssignalen skickades ut och väntade sedan "tillräckligt länge" för att sedan köra en "for loop" för att mäta den analoga signalen och spara undan den tidpunkt då det högsta ekot togs emot. Den tidpunkt som sparades undan skulle sedan användas för att beräkna avståndet. Tester på detta utfördes flera gånger med varierat resultat det vill säga utan repeterbarhet. Mycket arbete lades ner på kretsen och koden då felet antogs ligga där. Felet förmodades sedan ligga i Arduinons hårdvara och den tid det tar att läsa analoga signaler. Enligt [54] är Arduinons hastighet för att läsa ett analogt värde cirka 100 µs. Då ekot av signalen har ungefär samma varaktighet så innebär det att max 2-3 mätningar hinner utföras. Då samplingsfrekvensen är för låg går det inte vara säker på om man har mätt in topparna eller någonstans däremellan.

3.3.3.2 Digital krets

För att slippa byta plattform från Arduinon gjordes ett test att använda en digital ingång istället. Arduinon har möjlighet att läsa av en digital ingång snabbare än den analoga enligt flertalet forum, tillverkaren specificerar själva inga siffror som den gjorde för det analoga fallet. Enligt denna forumkällan [55] kan Arduinon läsa den digitala ingången på 4,78 µs inklusive iterationsloop. Vilket är runt 20 gånger snabbare än den analoga läsningen.

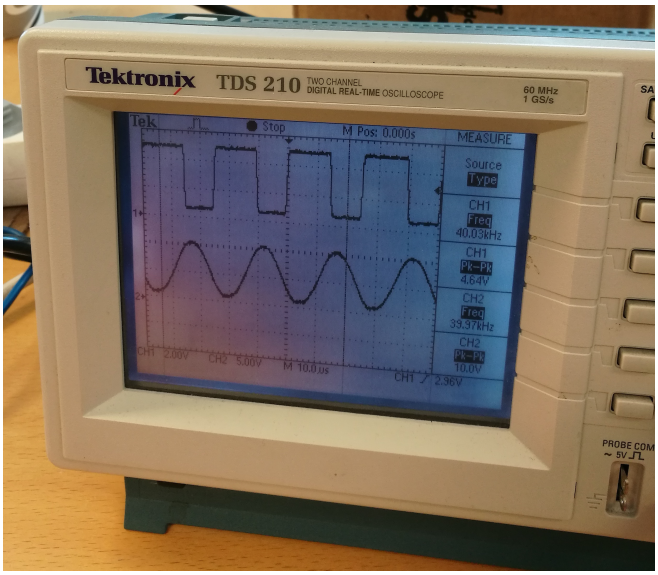
För att göra om kretsen från analog till digital gjordes via en schmittrigger. Schmittriggern går att bygga på olika sätt. I det här projektet valdes det att bygga schmittriggern av en operationsförstärkare då komponenter fanns lättillgängligt för test. Schmittriggern kopplades som "osymmetrisk schmittrigger via operationsförstärkare med 'single power supply'" och visas i figur 3.11 [56].



Figur 3.11: Figur 3.10a kompletterad med en "osymmetrisk schmittrigger via operationsförstärkare med 'single power supply'"

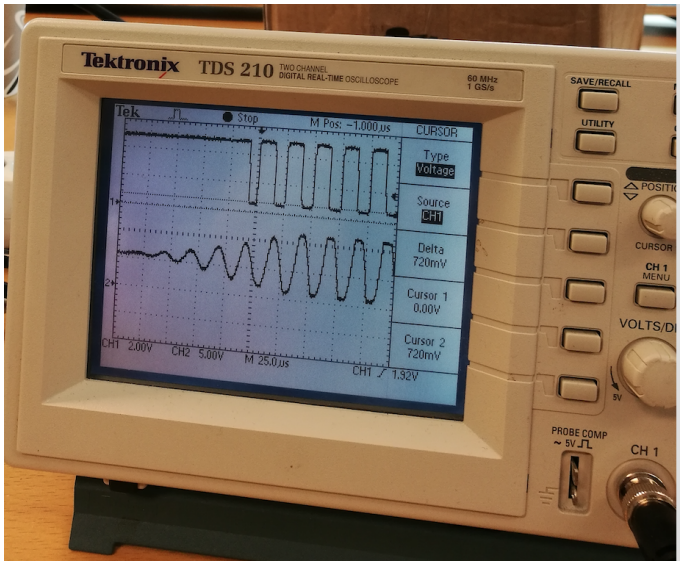
För att förklara vad en schmittrigger är går det förenklat säga att den kan göra om en sinusformad spänning till en fyrkantsformad spänning. Mer specifikt är att om insignalens spänning går över den som är det övre tröskelvärdet hos schmittriggern blir utspänningen från kretsen densamma som matningsspnningen. Omvänt om den passerar den lägre tröskelgränsen blir utspänning densamma som den negativa matningspänningen, i "single rail" fallet innebär det jord. Viktigt att poängtera är att det finns inverterade och icke-inverterade schmittriggers och då blir utspänningen densamma som maxspäninningen om spänningen på ingången passerar det lägre tröskelvärdet och vice versa. Att schmittriggern är av osymmetrisk typ innebär att gränserna för bägge tröskelspänningarna är positiva och att de går att placera efter önskat värde.

Räknaren på [57] användes för att bestämma resistanser till figur 3.11. Genom att använda $R_7 = R_8 = 33$ kΩ och $R_9 = 56$ kΩ får övre tröskelspänningen = 9,3 V och undre tröskelspäningen = 4,6 V för 12 V. Resultatet för detta visas i figur 3.12 där den nedre sinuskurvan är den analoga spänningen som oscillerar kring hälften av matningsspänningen som är kring $6V$. Denna spänning ger upphov till fyrkantspulsen överst i bild och är utsignalen från schmittriggern som sedan är spänningsdelad för att få ner spänningen till Arduinons logiknivåer. Arduinons logiknivåer för spänningarna är att 3-5 V är logisk etta och 0-1,5 V är logisk nolla [58]. Genom att granska figur 3.12 går det att se att logiknivåerna ligger väl innanför Arduinons gränser. Fyrkantspulsen är inte helt jämn men anses vara tillräckligt bra.



Figur 3.12: Mätresultat av analogsignal efter förstärkning och digital signal efter schmittrigger för en kontinuerligt utsänd ultraljudssignal på 40 kHz

Figur 3.12 är för en kontinuerligt utsänd ultraljudssignal och som tidigare diskuterats behövs en kort signal som består av ett antal pulser. Figur 3.13 visar en ögonblicksbild av ett test för en sådan signal. För att ta bilden används ett tillfälligt triggvilkor och när det är uppfyllt pausar oscilloskopet bilden. Detta är nödvändigt för att kolla på en signal som inte är periodisk. Triggvilkoret för oscilloskopet går att se överst i bild på kanal 1 och är den lilla pilen till höger i bild. I samma figur går det att se att den analoga signalen nederst på oscilloskopet går det att se att det tar en stund innan triggvilkoret för schmittriggern uppfylls. Detta skulle gå att arbeta bort större delen av detta genom att förändra schmittriggerns tröskelspänningar. I dessa tester är det dock viktigast att se till att det triggar någorlunda jämnt och att det går att repetera tidtagningarna.



Figur 3.13: Ögonblicksbild av kort pulssignal med hjälp av digitalt oscilloskop

För att mäta tiden användes Arduinon. Den programmerades på flera olika sätt. Det första sättet byggde på den tidigare nämnda "for loop" varianten från [51]. Andra försök gjordes med Arduinons avbrottsrutiner till exempel en funktion "pulseIn()" som kontrollerar och tar tid från det att en ingång går från låg till hög och sedan tillbaka till låg. Resultaten av dessa tester, oavsett programmering, ger samma resultat som för det analoga fallet. Det fås alltså inte någon typ av repererbarhet eller mönster i mätvärdena. I ett försök att kolla nämre på detta utfördes tester genom att läsa av bakgrundsbruset på ingången och skriva ut resultaten i datorn, men det lästes enbart in konstanta ettor, Istället skulle det dykt upp enstaka nollor för att falsk triggning skulle ske. Då det inte blev någon ordning på detta gick arbetet vidare med att en färdig krets köptes och gås igenom i nästa avsnitt.

3.3.3.3 Slutgiltig krets

För att komma framåt i arbetet med positioneringssystemet köptes istället färdiga ultraljudssensorer som fungerar som en sonar. För att utföra test fanns det tillgängligt två stycken kretsar från tillverkaren Parallax. För att få ihop tillräckligt många sonarkretsar köptes ytterligare kretsar in av en annan modell som heter HC-SR04 som säljs från många olika tillverkare. Dessa två är snarlika i utförande och funktion och redovisas i figur 3.14. Parallaxen har dock 3 anslutningar och HC-SR04 har 4.

(a) HC-SR04 avståndsmätare [59] **(b)** Parallaxs avståndsmätare [60]

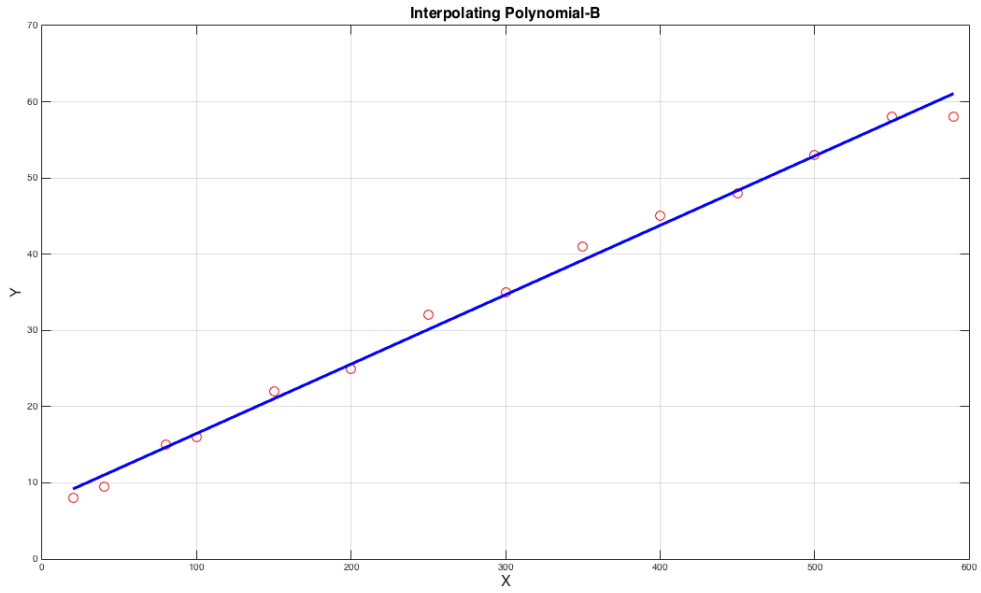
Figur 3.14: De två avståndsmätarna från olika fabrikat

Båda sensorerna är tillverkade så att de först skickas en kort logisk etta på ett par µs. När sensorn får denna signal startar en tidräkning hos sensorn och en ultraljudssignal skickas. Sensorn väntar sedan på en reflekterad ljudsignal, dock max i 18,5 ms för Parrallax.

I det här projektet används istället sensorerna inte som en sonar. Istället används de som det beskrevs i punktlistan i avsnitt 3.3.2 i början av kapitlet. För att få till denna delade funktion skickas det en startsignal för att starta tidtagningen i enheten på klipparen och RF-signalen skickar iväg startkommandon till fyrarna. För att

sensorn på klipparen inte ska skicka iväg ultraljud är den övertäckt med en fasttejpad skummgummibit. På fyrsidan är det istället så att RF-signalen tas emot och ultraljudet startas på samma sätt genom en kort logisk etta. På fyrsidan behöver inte mottagaren för ultraljudet täckas för, då Arduinon inte är programmerad för att göra något med den signalen. Koden som användes för detta var ganska simpel och kretsade primärt kring Arduinos kommando "pulseIn()". Koden för mottagaren finns redovisad i Appendix C.4, och koden för fyrarna finns redovisad i Appendix C.5

Med denna uppsättning fungerade systemet direkt för att mäta avstånd med ultraljud synkroniserat via RF-signal. För att testa hur bra mätningarna var skapades en testbana genom att avstånd mättes upp och markerades med tejpbitar. Avstånden mellan tejpbitarna var 20 cm i intervallet [0 cm, 100 cm] och 50 cm i intervallet [100 cm, 600 cm]. För att kolla om mätningarna gav rätt värde mättes avståndet till varje punkt flera gånger. Avståndet som mättes var inte helt korrekt men det gick att få upprepade värden. För att korrigera mätresultaten togs medelvärdet av fem mätningar på varje avstånd. Mätningarna visade god repeterbarhet och resultat på de fem mätningarna låg inom 4 cm på de längsta avstånden. Avvikelsen var mindre för de kortare avstånden. Något som noterades däremot var att felet i sig däremot ökade med ökat avstånd. För att åtgärda problemet användes MATLAB för att passa ett polynom med minsta kvadratmetoden. Lösningen för minsta kvadratmetoden med ett första ordningens polynom visas i figur 3.15. Avvikelsen från det verkliga mätvärdet går att se längs Y-axeln och det mätta avståndet på X-axeln.



Figur 3.15: Passning av polynom med hjälp av polyfit i MATLAB. Avikelsen från verkligt värde är på Y-axeln och uppmätt avstånd på X-axeln. Mätpunkter markerade med röda cirklar

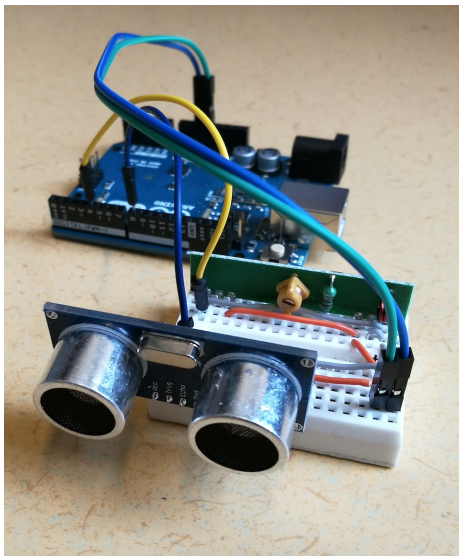
Avstånden som mättes blir inte längre än cirka 6 m. Detta är för att Parallaxen som nämnts tidigare hade en max tid på 18,5 µs och det är så långt ljudet hinner på den tiden plus fördröjningar i kod och RF-signal. Polynomet i figur 3.15 blev $Y = 0,09 \cdot X + 7,37$ (den blåmarkerade linjen) där X som beskriver felet som måste adderas till det uppmätta värdet. Testerna återupprepades och mätningarna blev bättre och centrerades kring det verkliga avståndet.

För att konstruera positioneringssystemet så kopplades tre ytterligare fyrar upp. Test utfördes för att se om felet som precis diskuterats var individuellt mellan fyrarna och nya polynom skulle behöva passas för de övriga fyrarna. Det visade sig att det första polynomet var likadant för samtliga fyrar och därför användes samma polynom på alla fyrar. Nu när fyra stycken fyrar var uppkopplade och mottagaren på gräsklipparen var klar kunde positioneringssystemet kopplas upp och testas.

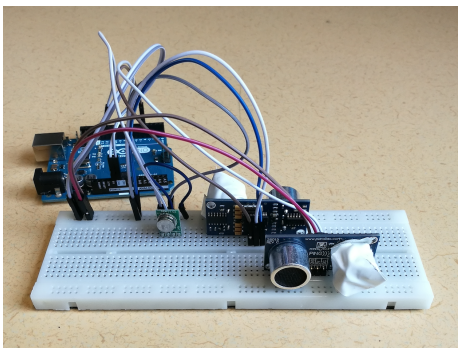
För att bygga positioneringssystemet behövde också ultraljudets sändar- och mottagarvinklar testas. Från början var det tänkt att mikrofonen på högtalaren skulle placeras så den var vänd uppåt. Ultraljudet skulle sedan ledas rätt genom att de placerades en bit ovanför mottagaren riktad lätt nedåt. Tester för spridningsvinkel och mottagningsvinkel visade att ultraljudet var smalare än tänkt både hos mottagare och sändare. Eftersom sonarkretsarna kunde mäta avstånd till föremål kring 20° förväntades att ljudet skulle sprida sig mer och eftersom ljudet inte behöver reflekteras i projektets uppställning förväntades vidare vinkel.

Egna tester visade att mottagaren kunde ta emot ljud från en cirkelbåge med 90° vinkel. Då högtalaren hela tiden var riktad mot mikrofonen och rörde sig längs cirkelbågen. För att testa hur brett ljudet utstrålades från sändaren sattes de upp mitt emot varandra och högtalaren vinklades lite grann bortifrån mottagaren och kunde därför passera mottagaren. Resultat för sändarvinkel var cirka 10°. Om positioneringssystemet skulle konstrueras på detta sättet hade högtalarna på fyrarna behövt höjas upp högt över mottagaren. Om fyrarna placeras för högt upp leder det till att avståndet som mäts som hypotenusan av en triangel blir för lång, längre än de 6 m som är det längsta avståndet som går att mäta med nuvarande uppsättning. Sättet som användes för att lösa problemet var att placera sändare och mottagare på samma höjd och låta mottagaren kolla i flera riktningar samtidigt.

Vid konstruktionen fanns det tillgängligt en extra sonarsensor som placerades rygg mot rygg med den andra mottagaren. Detta gjorde inte så att det var möjligt att få en 360° lyssningsvinkel men det blev möjligt att testa systemet. Då det bara användes två mottagare gick det att förflytta de kombinerade mottagarna så att de var riktade mot två fyrar vardera. Om fyrarna var placerade tätt ihop. Samma metod hade också gått och användas för att placera fler högtalare per fyr. Eller för att lägga till ännu fler mikrofoner hos mottagaren. Men det utfördes inte för att tiden höll på att ta slut. Den sändare och de fyrar som användes visas i sitt kompletta utförande med radiokomponenter i figur 3.16.



(a) Fyren för att ta emot radiosignal och sända ultraljud

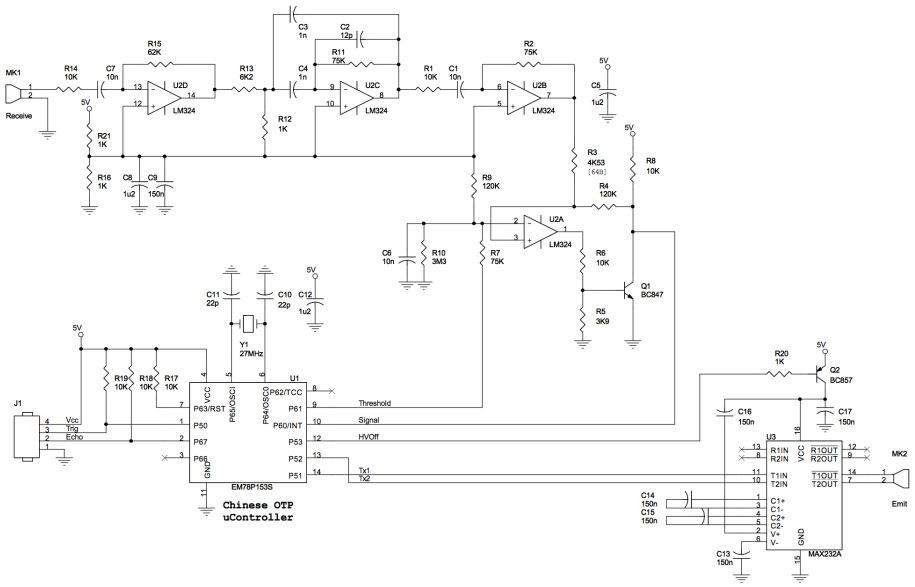


(b) Sändare av radiosignal och mottagare för ultraljud

Figur 3.16: Sändar- och mottagarkrets för att mäta avstånd

Fortsättningsvis anpassades mätdatan för att kunna föras över till Raspberry Pi för att där utföra trilaterationsberäkningar. Mätdatan organiserades i Arduinon på så sätt att mätdatan skulle skickas från samtliga fyra fyrar i varje överföring. Då det i enstaka fall blev så att en datapunkt mättes felaktigt (exempelvis om mottagaren tappade signalen från en sändare) mättes det in tre värden per fyr innan datan överfördes. Då det fanns tre avståndsmätningar per fyr togs medianvärdet av dessa mätningar i förhoppning av att det inte skulle uppstå fler än ett fel per fyr. Skulle det likväl bli så att ett felaktigt avstånd kvarstod efter medianberäkningen hade Arduinon också programmerats på så sätt att den skickar det föregående avståndet om det nya avståndet är mer än 30 cm ifrån det föregående. På så sätt skulle det krävas flera felaktiga mätningar på rad för att göra flera mycket felaktiga positionsuppdateringar. Arduinons kapacitet att skicka dessa värden till Raspberryn var två per sekund. Vilket i sin tur innebär att Arduinon, som mäter tre värden per fyr, gör 24 avståndsmätningar per sekund.

För att kort reflektera över att de kommersiella avståndsmätarna fungerar och inte de egenkonstruerade kretsarna följer här en kort utläggning av skillnaderna i konstruktionen. Från [61] finns en beskrivning av hur en HC-SR04 krets är tillverkad. Kopplingsschemat för kretsen visas i figur 3.17 och jämförs med den egentillverkade kretsen i figur 3.10 med tillägget att figur 3.10a har schmittriggerkretsen från figur 3.11 kopplad till utgången. I jämförelsen mellan kretsarna går det att se både likheter och olikheter. Bland likheterna går det se aktiva förstärknings- och filtreringssteg överst i figur 3.17. Utsignalen från dessa steg går sedan in i en komparator som här fyller samma funktion som en schmittrigger.



Figur 3.17: En “open source” variant av HC-SR04 från [61]

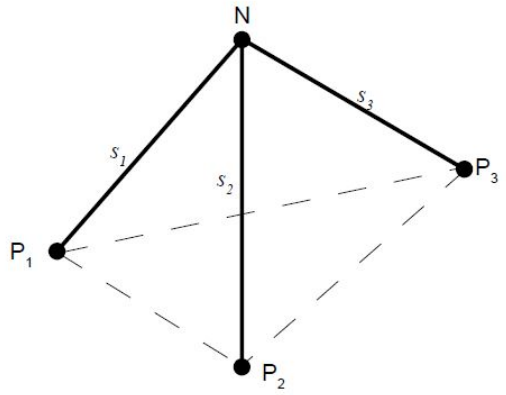
När det kommer till olikheterna däremot så har [61] krets en egen mikrocontroller samt en kristalloscillator. Kristalloscillatorn är en klockkrets som ser till att mikrocontrollern arbetar med 27 kHz till skillnad mot Arduninon som används i projektet som arbetar med 16 kHz. I nedre högerkant av figur 3.17 är kretsen för att sända ut ultraljud. Enligt [61] gör denna krets så att högtalaren matas med en spänning på 20 V och denna del av figuren bör jämföras med figur 3.14b. I kretsschemat i figur 3.17 går det också se att det finns fler kondensatorer kring biaseringen av operationsförstärkarna.

Med denna utläggning om krets tillverkning och skillnader så kommer dessa fortsatt diskuteras i slutsatsen om eventuella förbätringsarbeten. Härnäst följer hur implementeringen av trilateration gick till.

3.3.4 Trilaterationsalgoritm

Den algoritm som togs fram använder sig av trilateration som tidigare beskrevs i avsnitt 2.3.2. Trilaterationsalgoritmen som prototypen använder sig av är en modifierad version av en algoritm skriven av Abdelmoumen Norrdine [62]. Den ursprungliga algoritmen är skriven i MATLAB. Då Raspberry Pi inte har tillräckligt hög prestanda för att klara av systemkraven för MATLAB [63],[64], så var algoritmen tvungen att översättas till ett språk som kan användas av Raspberry Pi. Språket som valdes var Python. Den färdigredigerade koden som användes kan ses i Appendix C.1. Algoritmen är utformad för trilateration i 3D. Beräkningarna skiljer sig något åt beroende på hur många referenspunkter som är tillgängliga. Nedan beskrivs beräkningarna för de två olika fallen enligt Abdelmoumen Norrdine [62].

Lösning med tre referenspunkter
Referenspunkternas positioner $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$ och $P_3(x_3, y_3, z_3)$ är givna. Givet är också de uppmätta avstånden mellan referenspunkerna och den sökta punkten N, dessa kallas s_1, s_2 och s_3 (se figur 3.18).



Figur 3.18: Trilaterationsproblemet [65]

Att bestämma koordinaterna (x, y, z) för punkten N är ekvivalent med att hitta lösningarna till systemet av andragradsekvationer (3.4).

$$\begin{cases} (x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2 = s_1^2 \\ (x-x_2)^2 + (y-y_2)^2 + (z-z_2)^2 = s_2^2 \\ (x-x_3)^2 + (y-y_3)^2 + (z-z_3)^2 = s_3^2 \end{cases} \tag{3.4}$$

(3.4) kan skrivas som

$$\begin{cases} (x^2+y^2+z^2) - 2x_1x - 2y_1y - 2z_1z = s_1^2 - x_1^2 - y_1^2 - z_1^2 \\ (x^2+y^2+z^2) - 2x_2x - 2y_2y - 2z_2z = s_2^2 - x_2^2 - y_2^2 - z_2^2 \\ (x^2+y^2+z^2) - 2x_3x - 2y_3y - 2z_3z = s_3^2 - x_3^3 - y_3^2 - z_3^2 \end{cases} \tag{3.5}$$

Eller på matrisform

$$\begin{bmatrix} 1 & -2x_1 & -2y_1 & -2z_1 \\ 1 & -2x_2 & -2y_2 & -2z_2 \\ 1 & -2x_3 & -2y_3 & -2z_3 \end{bmatrix} \begin{bmatrix} x^2+y^2+z^2 \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_1^2 - x_1^2 - y_1^2 - z_1^2 \\ s_2^2 - x_2^2 - y_2^2 - z_2^2 \\ s_3^2 - x_3^3 - y_3^2 - z_3^2 \end{bmatrix} \tag{3.6}$$

Vidare kan (3.6) skrivas på den bekanta formen

$$A_0 \cdot \mathbf{x} = \mathbf{b_0} \tag{3.7}$$

Med kravet att: $\mathbf{x} \in E$
Där $E = \left\{(x_0, x_1, x_2, x_3)^T \in \mathbb{R}^4 / x_0 = x_1^2 + x_2^2 + x_3^2\right\}$

Fall 1: P_1, P_2 och P_3 ligger inte på en rak linje
Då är $\text{Rang}(A_0) = 3$ (där $\text{Rang}(A)$ definieras som det maximala antalet linjärt

oberoende kolonner i matrisen A) och nollrummet består endast av den triviala nollvektorn. Den generella lösningen till (3.7) är då

$$\mathbf{x} = \mathbf{x_p} + t \cdot \mathbf{x_h} \tag{3.8}$$

Med den reella parametern t. Där $\mathbf{x_p}$ är en partikulärlösning till (3.7) och $\mathbf{x_h}$ är en lösning tll det homogena systemet $A_0 \cdot x = 0$. Vektorerna $\mathbf{x_p}$ och $\mathbf{x_h}$ kan beräknas med Gausselimination. Partikulärlösningen $\mathbf{x_p}$ kan också bestämmas genom att använda Moore–Penroses pseudoinvers av matrisen A_0. Pseudoinversen ger lösningen med minsta möjliga norm.

För att bestämma parametern t sätts $\mathbf{x_p} = (x_{p0}, x_{p1}, x_{p2}, x_{p3})^T$, $\mathbf{x_h} = (x_{h0}, x_{h1}, x_{h2}, x_{h3})^T$ och $\mathbf{x} = (x_0, x_1, x_2, x_3)^T$. Med insättning i (3.8) fås

$$\begin{cases} x_0 = x_{p0} + t \cdot x_{h0} \\ x_1 = x_{p1} + t \cdot x_{h1} \\ x_2 = x_{p2} + t \cdot x_{h2} \\ x_3 = x_{p3} + t \cdot x_{h3} \end{cases} \tag{3.9}$$

Genom att använda kravet att $\mathbf{x} \in E$ fås

$$x_{p0} + t \cdot x_{h0} = \Big(x_{p1} + t \cdot x_{h1}\Big)^2 + \Big(x_{p2} + t \cdot x_{h2}\Big)^2 + \Big(x_{p3} + t \cdot x_{h3}\Big)^2$$

Och således

$$t^2\Big(x_{h1}^2 + x_{h2}^2 + x_{h3}^2\Big) + t\Big(2 \cdot x_{p1}x_{h1} + 2 \cdot x_{p2}x_{h2} + 2 \cdot x_{p3}x_{h3}\Big) + x_{p1}^2 + x_{p2}^2 + x_{p3}^2 - x_{p0} = 0 \tag{3.10}$$

Detta är en andragradsekvation på formen $at^2 + bt + c = 0$ med lösningarna

$$t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3.11}$$

Lösningarna till ekvationssystem (3.7) fås då som

$$\begin{aligned} \mathbf{x_1} &= \mathbf{x_p} + t_1 \cdot \mathbf{x_h} \\ \mathbf{x_2} &= \mathbf{x_p} + t_2 \cdot \mathbf{x_h} \end{aligned} \tag{3.12}$$

Om trilaterationsproblemet inte kan lösas (för korta avstånd), så finns det inga reella lösningar. I detta fall används den realdelen som en approximation till lösningen. Med denna approximation uppfylls inte kravet att $\mathbf{x_{1,2}} \in E$. Följaktligen fås skillnaden

$$d = x_0 - \Big(x_1^2 + x_2^2 + x_3^2\Big) \tag{3.13}$$

som ett mått på lösbarheten för trilaterationsproblemet. Här är x_0, x_1, x_2 och x_3 koordinaterna till lösningen $\mathbf{x}$, till ekvationssystem (3.7). Lösningarna till trilaterationsproblemet är punkterna $N_1 = \mathbf{x_1} \cdot I_0$ och $N_2 = \mathbf{x_2} \cdot I_0$, där

$$I_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fall 2: P_1, P_2 och P_3 ligger på en rak linje
Detta betyder att Rang$(A_0) = 2$ och att det finns en lösning utöver den triviala lösningen nollvektorn. Den generella lösningen till (3.7) är

$$\mathbf{x} = \mathbf{x_p} + t \cdot \mathbf{x_{h1}} + k \cdot \mathbf{x_{h2}} \tag{3.14}$$

Här är t och k reella parametrar, $\mathbf{x_p}$ är en partikulärlösning till ekvationssystem (3.7), och $\mathbf{x_{h1}}$ och $\mathbf{x_{h2}}$ är två lösningar till det homogena systemet $A_0 \cdot x = 0$. Dessa är linjärt oberoende lösningar och utgör därför en bas till nollrummet av A_0. Då det bara finns en begränsande ekvation har trilaterationsproblemet oändligt många lösningar.

Lösning med fyra eller fler referenspunkter
Med adderade referenspunkter P_4, P_5,..., P_n, samt avstånd s_4, s_5,..., s_n, fås följande ekvationssystem på matrisform

$$\begin{bmatrix} 1 & -2x_1 & -2y_1 & -2z_1 \\ 1 & -2x_2 & -2y_2 & -2z_2 \\ 1 & -2x_3 & -2y_3 & -2z_3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & -2x_n & -2y_n & -2z_n \end{bmatrix} \begin{bmatrix} x^2+y^2+z^2 \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_1^2 - x_1^2 - y_1^2 - z_1^2 \\ s_2^2 - x_2^2 - y_2^2 - z_2^2 \\ s_3^2 - x_3^3 - y_3^2 - z_3^2 \\ \vdots \\ s_n^2 - x_n^3 - y_n^2 - z_n^2 \end{bmatrix} \tag{3.15}$$

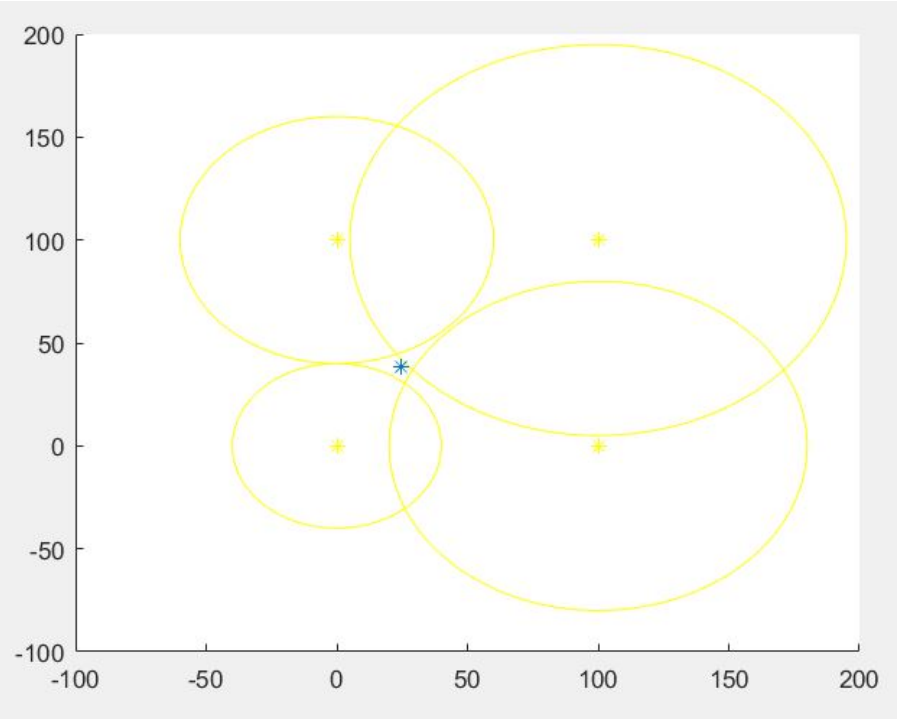
Vilket kan skrivas på den kända formen

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{3.16}$$

Med kravet att $\mathbf{x_{1,2}} \in E$ fås lösningen $\hat{\mathbf{x}}$ till (3.16) med minsta kvadratmetoden

$$\hat{\mathbf{x}} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b} \tag{3.17}$$

I figur 3.19 visas resultatet av ett test av algoritmen i MATLAB. De gula stjärnorna representerar referenspunkternas positioner. Cirklarna representerar de uppmätta avstånden mellan referenspunkterna och den sökta positionen. Den blå stjärnan representerar den approximerade positionen.



Figur 3.19: Illustration av trilaterationsalgoritm

3.4 Ruttplanering med hinderhantering

Ruttplaneringen är baserad på de teoretiska "Online-search-agents"-algoritmerna där roboten är omedveten om området den befinner sig i. Algoritmen gör då så att roboten kan utforska området och även lära sig området genom att spara en karta av arean den besökt. Huruvida det är en optimal lösning för robotgräsklipparen beror på gräsmattan den ska klippa, därför har det gjorts ruttplaneringsalgoritmer anpassade efter olika gräsmattor. Skillnaden mellan de olika algoritmerna är lärandet av området. I vissa fall anses det mer praktiskt att inte spara en karta över området, då omgivningen i en trädgård inte är statisk. Detta innebär dock att det hela tiden behöver utföras mer beräkningar på området, då robotgräsklipparen hela tiden måste utforska arean den klipper "på nytt".

Ruttplaneringen tar hänsyn till de olika objekt som robotgräsklipparen märker av och stöter på. Under utvecklingen togs det fram två algoritmer som som löser detta på olika sätt (Algoritm A och Algoritm B). Lösning A försöker alltid hitta en väg för att gå runt objekten, medan lösning B tar en annorlunda väg då den stöter på ett hinder, genom att tex. svänga 180°. De två algoritmerna skiljer sig dessutom åt gällande avsökningsmetod. Efter en analys drogs slutsatsen att båda algoritmerna hade sina fördelar. Utgången av detta blev att en ny algoritm (Algoritm C) togs fram. Algoritm C är en hybrid av Algoritm A och Algoritm B, samt influerad av LRTA* (se avsnitt 2.4.1.2).

Algoritmerna använder sig av den teori som behandlas i avsnitt 2.4. Samtliga algoritmer är skrivna i Python för att enkelt kunna köras av en Raspberry Pi. Samtliga algoritmer finns i Appendix C.2. I följande avsnitt beskrivs de tre algoritmerna som har tagits fram.

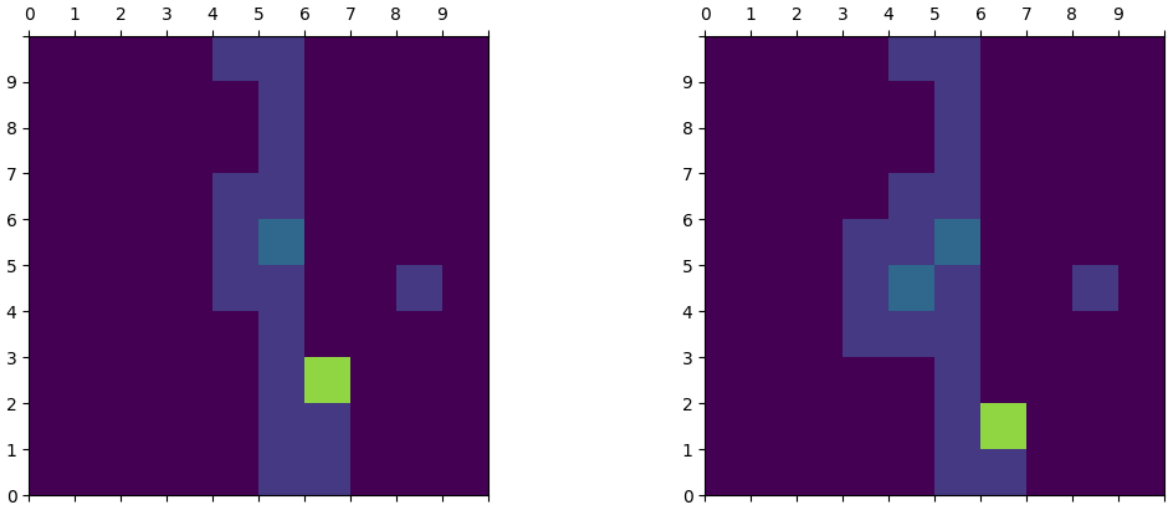
3.4.1 Algoritm A

Algoritm A bygger på en Online-search-agent (se avsnitt 2.4.1). Algoritmen arbetar sig framåt i ett "sicksack-mönster". Då den stöter på ett hinder kör den runt hindret och återvänder sedan till den ursprungliga kursen. När det förekommer många näraliggande hinder, alternativt väldigt stora hinder, så kan det vara en fördel att "undersöka" de närliggande objekten. Detta för sedan köra runt allihopa och återgå till tidigare kurs. Sedan så fortsätter den att röra sig "sicksack" genom området. Denna algoritm passar bäst på områden med små objekt nära varandra, då den alltid åker runt hinder för att undvika dessa.

```
1  Initialize ():
2     queue
3  CheckObjects (x, y)
4  if grid[x][y] == obstacle then
5  |  for every adjacent tiles that is not obstacles:
6  |     add adjacent tiles to the top of queue
7  end
8  OnlineAlgoritm: (queue)
9  while queue > 0 do
10 |  node = queue.popleft()
11 |  x = node[0]
12 |  y = node[1]
13 |  if grid[x][y] is not visited then
14 |  |  grid[x][y] = visited
15 |  end
   |  CheckObjects: (x,y)
16 end
```

Algorithm 4: Online-Search-Algoritm som har en fördefinerad rutt som går "sicksack" genom området. Samtidigt som den känner av objekt och tar handlingar som gör att den åker runt objektet. För att sedan fortsätta sin bana

I figur 3.20a, som är en simulering av Algoritm A, kan man se att den gröna markören (symbol för gräsklipparen) har åkt uppifrån och ner mot ett hinder. Lösningen i detta fallet var att runda hindret från den högra sida, ur gräsklipparens perspektiv. Algoritmen håller ordning på vilken riktning gräsklipparen borde ha, och kan därav undvika hinder baserat på vilken riktning gräsklipparen har gentemot hindret. Algoritm A kan även hantera närliggande och ihopsatta objekt som i figur 3.20b. Här åker den tänkta gräsklipparen runt alla hinder som har kontakt med det första hindret den stöter på.



(a) Algoritm A med ett hinder

(b) Algoritm A med två hinder

Figur 3.20: Algoritm A som hanterar hinder genom att åka runt hindret innan den fortsätter sin "sicksack-rutt"

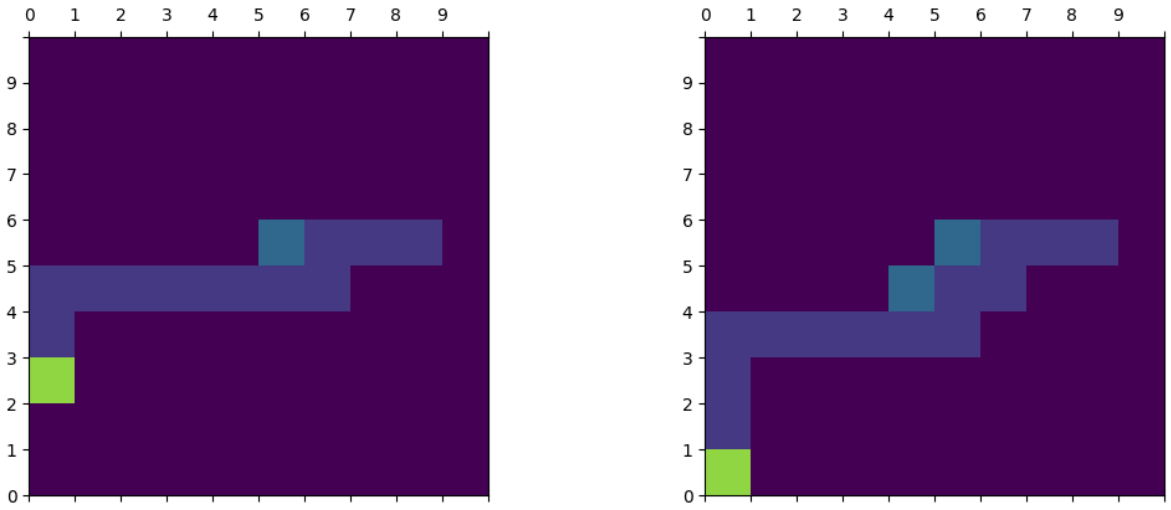
3.4.2 Algoritm B

Denna algoritm är baserad på en så kallad Online-DFS-Search-algoritm, se avsnitt 2.4.1.1. Grunden till algoritmen bygger på att försöka åka så långa sträckor som möjligt, vilket medför att roboten svänger minimalt. Detta kan jämföras med Algoritm A som alltid rör sig runt hindren och sedan återvänder till ursprunglig kurs. Jämför man svängar med att åka rakt fram, så tar svängar mer energi och även mer tid. Denna algoritm är bäst anpassad för ett område där det inte förekommer många objekt tätt inpå varandra.

```
1  Online-DFS: (x, y);
   Input      : Point X and Y on the plane
2  if grid[x][y] is not visited then
3  |  grid[x][y] = visited
4  |  if y > 0 then
   |  |  Online-DFS: (x,y-1) % left
5  |  end
6  |  if x > 0 then
   |  |  Online-DFS: (x-1,y) % down
7  |  end
8  |  if x < the bound in x-axis then
   |  |  Online-DFS: (x+1,y) % up
9  |  end
10 |  if y < the bound in y-axis then
   |  |  Online-DFS: (x,y+1) % right
11 |  end
12 end
```

Algorithm 5: Rekursiv Online-DFS-algoritm som inte lär sig området. Algoritmen antar att den startar på 0,0 och sedan itererar genom området med prioritet på vänster och neråt

Algoritm B är utformad för att inte slösa energi på att försöka runda hindret, utan istället fortsätta vidare. I figur 3.21a har den den gröna markören rört sig förbi ett hinder från den högra sidan. Detta är för att algoritmen fungerar på så sätt att den prioriterar vänster och neråt först. Den tänkta gräsklipparen svänger av när den kommer fram till ett hinder och fortsätter sedan vänster och neråt. I figur så ser vi att den använder samma logiska lösning med två hinder.



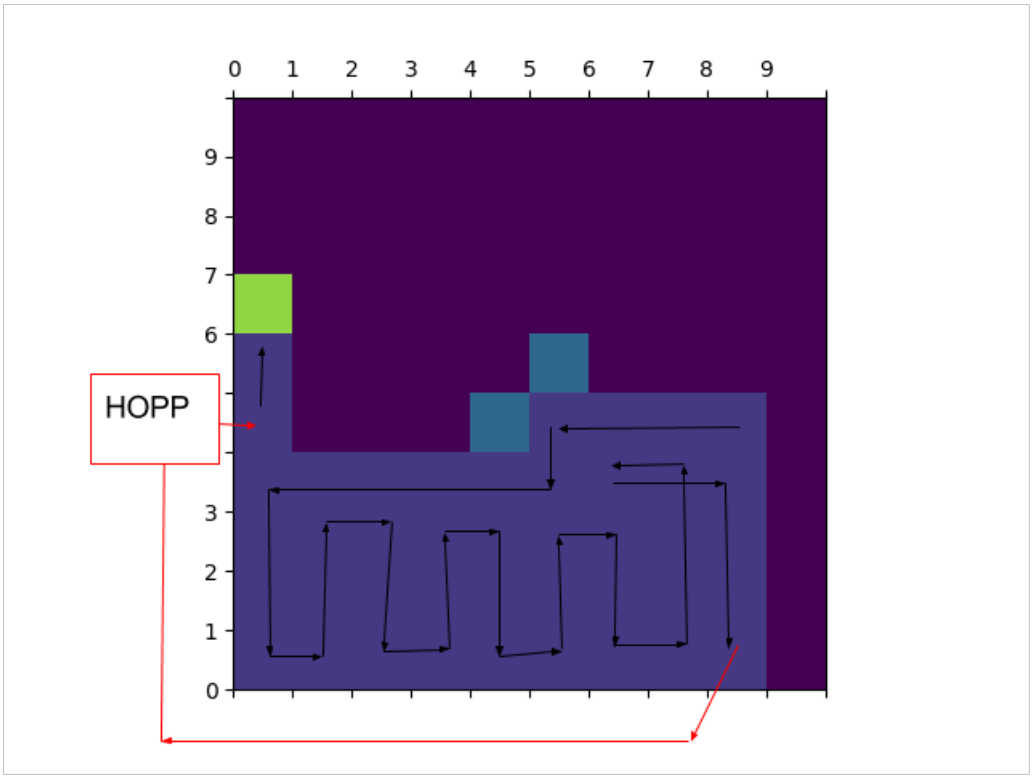
(a) Algoritm B med ett hinder

(b) Algoritm B med två hinder

Figur 3.21: Algoritm B som hanterar hinder genom att prioritera vänster och neråt. Den åker inte runt hindret utan svänger av och fortsätter sin väg

3.4.3 Analys av Algoritm A och Algoritm B

Algoritm B kan inte integreras direkt med gräsklipparen då den endast ger gräsklipparen vilka noder som gräsklipparen ska söka. Den ger alltså inte alltid en fullständig väg för gräsklipparen att följa. Algoritmen "hoppar" fram till områden där gräsklipparen ska klippa härnäst. Detta beror på att konstruktionen av algoritmen bygger på en lista, där den sparar gräsklipparens nästa punkter att klippa i området. Algoritmerna är dessutom bäst optimerade för att starta nerifrån vänster, speciellt Algoritm A då den alltid vill röra sig i ett "sicksack-mönster". Algoritm B väljer alltid neråt och vänster före uppåt och höger. Detta fortsätter ända tills den inte kan gå sina prioriterade vägar, då den vänder tillbaka till det område som den tidigare valde bort. Detta är orsaken till "hoppen" som förekommer. En illustration av detta finns att se i figur 3.22. På grund av dessa nackdelar utvecklades en tredje algoritm som och blev den slutgiltiga, Algoritm C.



Figur 3.22: Algoritm B som bara ger nästa punkt att klippa och inte vägen dit, den “hoppar”

3.4.4 Algoritm C

Algortim C är en vidareutveckling av Algoritm A och Algoritm B med inslag av algoritmen LRTA*. Den rör sig runt föremål på samma sätt som Algoritm A, och utforskar området på samma sätt som Algoritm B. Algoritm A och Algoritm B gav endast robotgräsklipparen nästa punkt att besöka, och alltså inte vägen till nästa punkt. På grund av detta kompletterades den med delar av LRTA*. Resultatet av detta är att den tar hänsyn till att vissa områden kan ha orörliga objekt och hinder. Detta löses genom att spara en karta av området. Med ett antal beräkningar kan sedan en effektiv rutt läras in. Tanken är att algoritmen först ska få åka en “visningsrunda” och då lära sig alla fasta objekt. Sedan utgår den från detta minnet och undviker hinder såsom Algoritm A gör.

```
1  learnMemory (memorylist);
   Input      : is a list with walked sequences
2  Check through the steps in memory and see where it "jumps", and then take
     the sublist of the "jumping path" and remake the "memory" to not make
     the jump by walking that path before the main path.

3  LRTA*hybrid (x, y, memory);
   Input      : Point X and Y on the plane
   Output     : memory of the steps that has been walked

4  stack = ([]);
5  visited = ([]);
6  prev = ([x,y])
7  while Stack not empty do
8  |  current = stack.pop()
9  |  if current in visited then
10 |  |  skip to next iteration
11 |  end
12 |  x = current.x
13 |  y = current.y
14 |  //checks if it is jumping and walks back if it is.
15 |  if abs(prev.x - x) > 1 or abs(prev-y - y) > 1 then
16 |  |  shortestPath from prev(x,y) to (x,y)
17 |  end
18 |  prev = ([x,y])
19 |  visited.append([x,y])
20 |  grid[x][y] = visited
21 |  memory.insert([x,y])
22 |  if (x,y+1) not in visited and inside boundaries then
23 |  |  stack.append([x,y-1])
24 |  end
25 |  if (x+1,y) not in visited and inside boundaries then
26 |  |  stack.append([x-1,y])
27 |  end
28 |  if (x-1,y) not in visited and inside boundaries then
29 |  |  stack.append([x+1,y])
30 |  end
31 |  if (x,y-1) not in visited and inside boundaries then
32 |  |  stack.append([x,y-1])
33 |  end
34 end
35 return(memory);
```

Algorithm 6: En icke-rekursiv Online-DFS-algoritm som lär sig området. Vilket gör den till an hybrid av LRTA* och Online-DFS-algoritm som anpassas till robotgräsklipparen. Den returnerar ett slags minne av alla sina steg

4
Resultat

4.1 Gräsklippare

I detta avsnitt tas resultaten för de olika delarna av gräsklipparen upp.

4.1.1 Chassi

Gräsklipparens chassi utför sin uppgift på ett bra sätt. På grund av dess stabilitet, höga styvhet och hårdhet har chassit inga problem med att bära upp komponenterna eller att stå emot eventuella kollisioner. Det problem som upptäckts är att de främre delarna av chassit ibland slår i ett hinder/vägg då gräsklipparen roterar. Detta kan exempelvis förekomma då gräsklipparen kör längs med en vägg i riktning mot ett hörn. Om den då stannar och roterar mot den sida där väggen är finns det en risk att den främre delen av gräsklipparen slår i väggen. Detta för att rotationscentrum ligger mellan bakhjulen, och den främre delen av gräsklipparen roteras då i en ganska stor radie, ca 30 cm.

4.1.2 Drivning och styrning

Gräsklipparen rör sig överlag på ett mycket bra sätt. I och med den inbyggda PID-kontrollern håller gräsklipparen en mycket rak kurs på plana partier. Ojämnheter medför dock att den avviker något. Gräsklipparen är kort testad för backkörning och påvisade även där mycket goda resultat. Gräsklipparen körde upp för tre meters höga backar med upp till 20° lutning på ett bra sätt. En viss kursavvikelse förekom. Tester gjordes även i backar med en lutning på upp till 30°. Gräsklipparen fick då problem där marken var ojämn, fronten fastnade i marken och hjulen började slira.

4.1.3 Klippaggregat

För att testa klippaggregatet gjordes ett antal testkörningar utomhus på en gräsmatta. Gräsmattan innehöll både delar som hade högt och tjockt gräs, samt delar som hade tunt och ganska kort gräs. Gräsmattan innehöll också varierande terräng, i form av plan mark, lutningar och gropar. Klippresultatet blev mycket bra, den klippta ytan håller genomgående en höjd på 3 cm. Klippaggregatet körde fast en gång på grund av för högt och tjockt gräs. Gräsets höjd var då nära 20 cm högt och stod mycket tätt. Gräsklipparen hanterade gräs med höjder en bit över 10 cm utan några som helst problem. I figur 4.1 visas resultaten av två körningar genom högt och tjockt gräs. Det syns här väldigt tydligt vad som är klippt och inte.

Figur 4.1: Resultat av testkörning med klippaggregat

Gräsklipparen klippte mycket bra på plan mark och i lutningar. Den hade dock svårt när den kom fram till en grop (ungefär lika stor som gräsklipparen), klippaggregatet körde då fast i samband med att det slog i marken.

Testkörningarna gjordes under en kort tidsperiod, ungefär 20 min. Under denna tid fungerade knivarna mycket bra och den lilla massa gräs som fastnade på knivarna hade ingen märkbar inverkan. Några långtidstester har inte genomförts, främst på grund av tidsbrist.

4.2 Hinderdetektering

Den slutgiltiga placeringen av sensorerna (se avsnitt 3.2.1) fungerar till stor del på ett bra sätt. För att komma fram till detta utfördes flera testkörningar. Testkörningarna gjordes både utomhus och inomhus och med flera olika hinder så som väggar, stolar, träd, buskar med mera. Under de utförda testkörningarna upptäcktes hindren i tid i de absolut flesta fallen. Gräsklipparen kunde sedan hantera detta genom att backa tillbaka en bit för att sedan rotera och fortsätta i en ny riktning. Det förekom dock att gräsklipparen inte lyckades detektera mycket smala hinder (exempelvis smala bordsben) i tid, utan körde på hindret lätt innan den reagerade. Detta fenomen förekom då gräsklipparen närmade sig ett mycket smalt hinder på ett sådant sätt så att hindret hamnade mellan sensorernas detekteringsvinklar. I figur 3.9 visas hur sensorernas detekteringsvinklar täcker hela gräsklipparens front, men om hindret precis undviker den centrala sensorn så upptäcks den väldigt sent. I praktiken visar sig detta vara så sent att gräsklipparen inte hinner stanna förrän den träffar hindret. Det ska också sägas att gräsklipparen inte reagerade vid precis samma avstånd från hindren varje gång. I de flesta fallen reagerade gräsklipparen då avståndet var mellan 2-6 cm, men ibland stannade den uppemot 10 cm från hindret, och i sällsynta fall hann den som sagt inte stanna innan kollision.

För att få en mer exakt överblick på hinderdetekteringens precision gjordes tester utomhus. Testerna utfördes på så sätt att först placerades en pinne, med en diameter på ca 2-3 cm, framför robotgräsklipparen. Pinnen trycktes ner i marken så den stod lodrätt upp och illustrerade då ett mycket smalt träd. Därefter kördes robotgräsklipparen rakt mot denna pinne för att ta reda på hurvida roboten detekterade hindret eller inte. Resultatet av detta test gav att om robotgräsklipparen kör rakt fram mot pinnen, med pinnen placerad så att den träffar robotens främre hörn så detekterades hindret i 90 % av fallen. Utöver detta detekterades pinnen, om den skulle träffat de centrala delarna av robotens framsida, med 100 % säkerhet.

4.3 Positioneringssystem

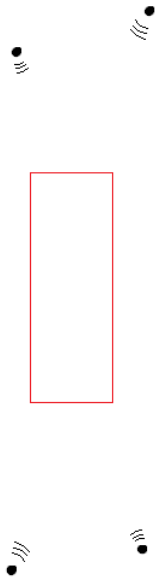
Följande avsnitt beskriver hur testet för positioneringssystemet gick till och hur resultatet av testet blev.

4.3.1 Beskrivning av test

För att testa positioneringssystemets precision utformades ett test med ett par varianter. Testet som gjordes på positioneringssystemet gick ut på att ta reda på hur mycket dess position avviker från den verkliga positionen, då roboten följer en rektangulär bana.

Detta test utformades på så sätt att det först placerades ut fyra stycken fyrar i en fyrkant. Fyrarnas positioner noterades. Eftersom ultraljudet fungerade relativt dåligt på marken, placerades fyrarna en bit upp från marken. Testet skedde inomhus, dels för enkelhetens skull, men främst för att det skulle ge bäst resultat då inte parametrar som vind, temperatur och liknande behövde tas i beaktning. Samtliga fyrar är riktade in mot samma punkt, så att en yta i mitten täcks av samtliga fyrars spridningsvinkel. Positioneringssystemets mottagare plockades därefter bort från roboten, så att rörelserna kunde ske manuellt med en högre noggrannhet. Positioneringssystemet kunde därmed positionera mottagaren som förflyttades för hand.

Testet gick sedan till på så sätt att först kontrollerades storleken på ytan för det gemensamma “synfältet”. Detta gjordes för att få en förståelse för hur stor yta positioneringssystemet kan ge en position inom, innan det tappar kontakten med någon av fyrarna. I denna yta markerades därefter en rektangel ut (se figur 4.2), där positionerna för rektangelns hörn var bestämda. Mottagaren flyttades därefter längs den utmarkerade rektangelns sidor samtidigt som de uppmätta positionerna ritades upp i en graf. Då den utmarkerade rektangelns position var bestämd kunde de uppmätta positionerna och den verkliga rektangeln ritas upp i samma graf. I grafen gick det sedan enkelt att se hur mycket mottagarens position avviker från den verkliga positionen. Detta test utfördes flera gånger för att få en mer översiktlig bild av storleken på avvikelserna och repeterbarheten hos systemet.



Figur 4.2: Förenklad modell över testuppsättning

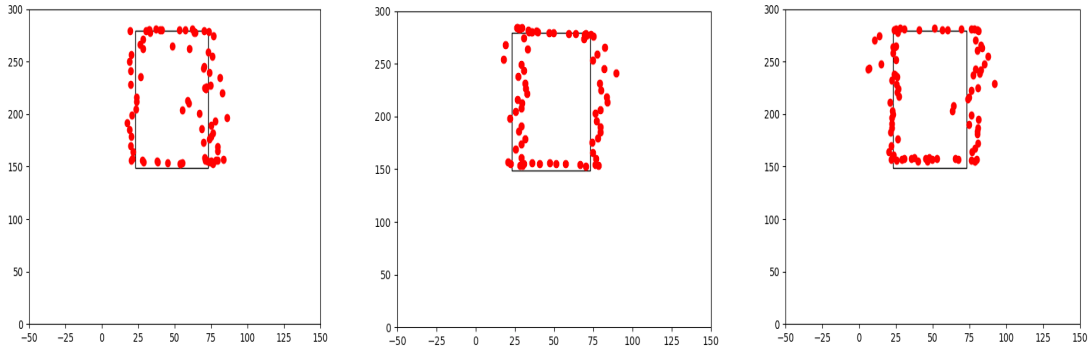
Samma test utfördes också då fyrarna flyttades till andra positioner riktades med andra vinklar. Detta för att ge ett mer tillförlitligt resultat. Det ska noteras att mottagarens kod kräver avstånden från alla fyra fyrar (vilket beskrivs i avsnitt 3.3.3.3) för ett bra resultat.

4.3.2 Testresultat

Storleken på ytan för det gemensamma synfältet uppmättes i det första testet till ungefär 1×2 m^2. När fyrarna placerades ut uppmärksammades det att "fyrparen" (de fyrar med liknande position i y-led) var tvungna att inneha liknande positioner i y-led. "Fyrparen" kunde flyttas, men de var alltså tvungna att flyttas parvis för att inte erhålla betydligt försämrade resultat. Det ska också sägas att resultaten blev mycket försämrade vid andra uppsättningar än vid de rektangelliknande. En annan observation som gjordes var att då ett av "fyrparen" flyttades utåt i y-led, så blev synfältet längre, men smalare. Detta gjorde att synfältet ungefär blev 2 m^2 oavsett om avståndet mellan "fyrparen" var tre, fyra eller fem meter. Då ultraljudssändarnas spridningsvinkel är väldigt begränsad gick det heller inte att ha "fyrparen" allt för nära varandra, eftersom detta minskade ytan för det gemensamma synfältet markant.

Vid första testuppsättningen placerades fyrarna på positionerna (0, 0), (0.95, 0.30), (0.10, 3.90) och (1.10, 3.95). Origo är här placerat i nedre vänstra hörnet och avstånden är i meter. Detta gjorde att rektangeln, som positioneringssystemet skulle föras utefter, hade dimensionen $0,5 \times 1,3$ m^2. Det var möjligt att göra rektangeln större, men bra mätvärden erhölls i hela rektangeln och därför behövdes inte dimensionen på rektangeln ändras. I övriga testuppsättningar flyttades fyrarna endast ett fåtal decimeter och vinklades så att de återigen täckte rektangelns yta med sitt synfält. Detta gör att det ungefär är samma storleksordning på samtliga mått.

Figur 4.3 visar hur mycket den uppmätta positionen avviker från den verkliga positionen, vid tester för tre olika placeringar och vinklingar av fyrarna. Den svarta rektangeln är uppritad i grafen enligt de koordinater den verkliga utmarkerade rektangeln hade. De röda punkterna är de positioner positioneringssystemet registrerade då mottagaren fördes längs rektangeln. För att få många punkter fördes mottagaren långsamt längs rektangeln. Notera att skalan på y-axeln är mycket större än skalan på x-axeln.



(a) Testuppsättning 1 **(b)** Testuppsättning 2 **(c)** Testuppsättning 3

Figur 4.3: Figuren visar grafer på hur avvikelserna (i centimeter) ser ut vid tre olika placeringar av fyrarna

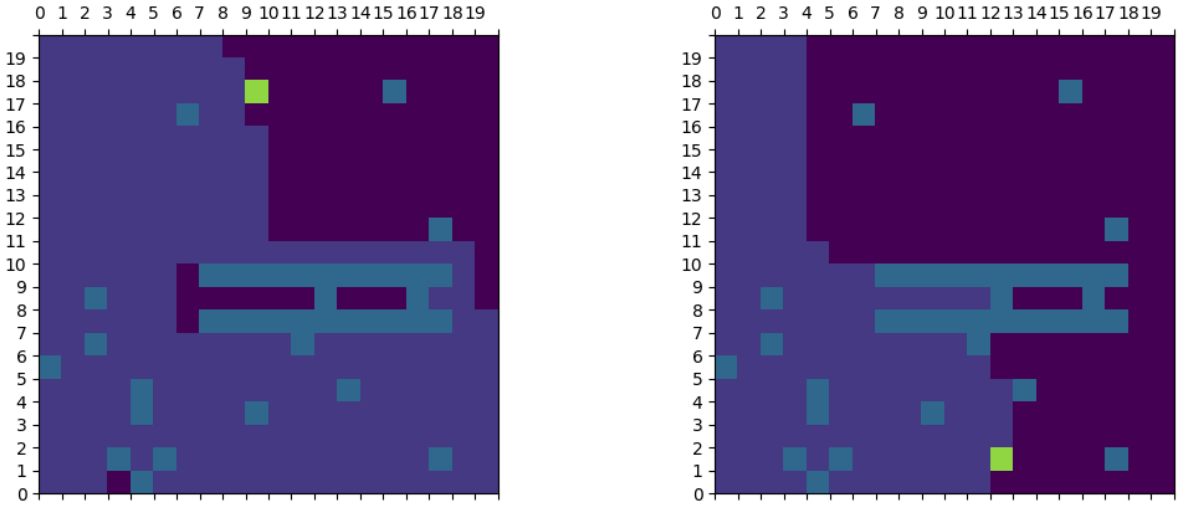
Figur 4.3a visar att den vänstra, översta och nedersta delen hade en ganska bra precision eftersom att de flesta punkter träffar - eller är mycket nära - den svarta linjen. De flesta punkter har här en avvikelse på centimeternivå. Vad gäller den högra sidan är det många fler punkter som avviker från linjen, och flera av dess avvikelser är dessutom i storleksordningen två decimeter. Detta berodde på att mottagaren tappade signalen från en eller flera av sändarna. För denna testuppsättning utfördes tre tester, samtliga grafer finns under Appendix B.1.

Figur 4.3b visar att det finns rätt många avvikande punkter. Dock är de uppmätta positionerna förskjutna lite åt höger. Bortsätt från förskjutningen så avvikelserna jämföras med avvikelserna jämföras med de från föregående stycke. Samtliga grafer från denna testuppsättning finns under Appendix B.2.

Figur 4.3c visar att den höll sig mycket bra på linjen förutom ett fåtal avvikelser på vänstra sidan, då ultraljudsmottagaren tappade signalen från en eller flera sändare en kort stund. Punkterna på högra sidan är förskjutna åt höger. Avvikelserna uppgår till decimeternivå. Samtliga grafer från denna testuppsättning finns under Appendix B.3.

4.4 Ruttplanering med hinderhantering

Den slutgiltiga algoritmen (Algoritm C) fungerar bra för dess ändamål. Det är en självlärande algoritm som utforskar ett område första gången den söker sig igenom det. Därefter sparas en karta över hur den har rört sig under sin första runda. Rutten optimeras sedan genom att algoritmen analyserar sitt minne och programmerar om det så att den tar den mer lönsamma vägen, då den har flera val av vägar att gå. I figur 4.4a visas det att den istället för att svänga in i gränden, valde att fortsätta rakt fram. I figur 4.4b har den lärt sig av sitt misstag, och har då ändrat i sitt minne vilken rutt den ska välja.

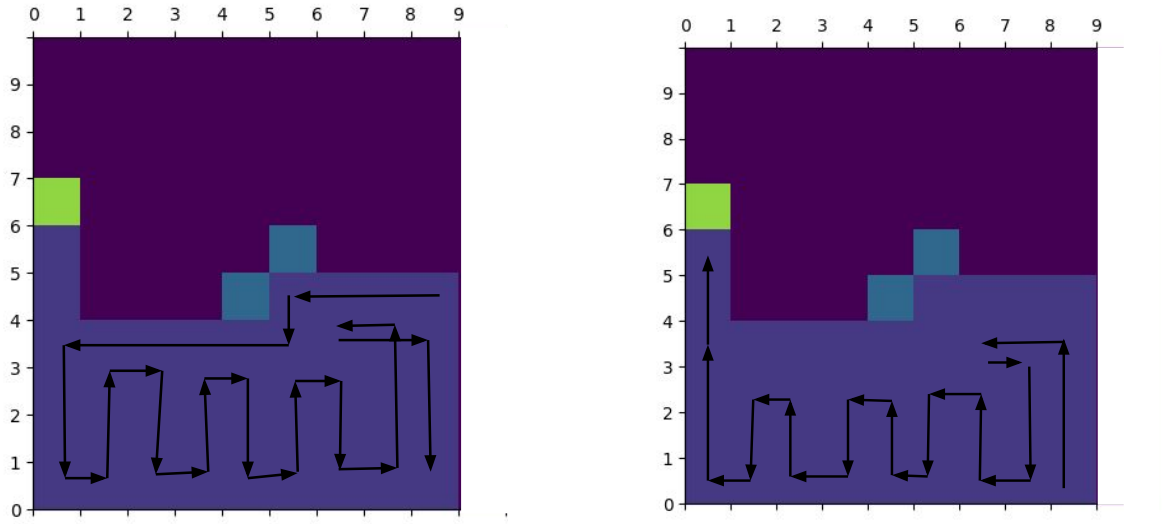


(a) Algoritm C kör en testrunda

(b) Algoritm C effektiv planering

Figur 4.4: Algoritm C som utforskar området och sedan sparar en karta som dessutom är optimerad så att den tar en smartare väg nästa gång

Algoritm C är dessutom implementerbar på robotgräsklipparen, i och med att den inte "hoppar". Den är dock inte optimerad. Då algoritmen kommer till en plats, som den av någon anledning vill "hoppa" ifrån (exempelvis en återvändsgränd), så löser den det genom att analysera sitt minne. Den undersöker hur den kom till sin nuvarande plats. Sedan undersöker den om något av de steg som den har rört sig, är granne med den punkt som den vill "hoppa" till. Om så är fallet så går den samma väg, fast tillbaka, för att komma till den nya punkten. Figur 4.5 illustrerar detta exemplet.



(a) Prioritering av vänster och neråt **(b)** Går samma väg tillbaka till

Figur 4.5: Algoritm C som går till en ny punkt genom att gå samma väg tillbaka till den nya punkten

4.5 Kostnad

Tabell 4.1 visar kostnaden för projektets samtliga komponenter, med undantag av det material som erhölls av Chalmers prototyplab. Kostnaden för inköpta komponenter uppgick till 4969 kr. För det material som erhölls är det svårt att ta fram ett exakt värde, men det uppskattade värdet av materialet är 300 kr. I materialet från prototyplabbet inkluderas skruvar, muttrar, distanser och aluminiumplåt. Det ska dock noteras att aluminiumplåten står för den absolut största delen av kostnaden. Den totala materialkostnaden för prototypen (inklusive den uppskattade materialkostnaden) uppgick således till 5269 kr.

Tabell 4.1: Tabell över kostnaderna för projektets komponenter

Produkt	**Antal**	**Kostnad/st (kr)**
Lynxmotion växlad drivmotor DC	2	379,93
Sabertooth styrkort till drivmotorer	1	587,03
Kangaroo styrkort till drivmotorer	1	190,83
Cytron MD10C styrkort till klippmotor	1	124,33
RS-555 klippmotor DC	1	41,86
Axelnav drivmotorer	2	87,09
Frakt Robotshop	1	195,26
Batterieliminator till Raspberry Pi	1	111,20
Minneskort till Raspberry Pi	1	127,20
Raspberry Pi 3	1	318,40
Radiomottagare	4	71,20
IR-sensorer	3	119,20
Ultraljudsändare	4	31,20
Ultraljudmottagare	1	31,20
Kabel för IR-sensorer	3	15,20
Radiosändare	1	63,20
Frakt electro:kit	1	79,20
Kopplingsplatta liten	3	49,90
Luxorparts Start-kit för Arduino	1	199,90
Strömbrytare	1	29,90
Kopplingsplatta stor	1	99,90
Säkring	10	2,99
Hållare till säkring	2	19,90
Framhjul	2	26,90
Arduino Uno R3	5	149,90
Total kostnad		**4968,95**

5
Slutsats

Genom att summera resultaten av projektet kan det konstateras att det har åstadkommits en fullt fungerande robotgräsklippare (på inhängnat område) som är jämförbar med de enklaste robotgräsklipparna på marknaden, samt "bygg-själv-robotgräsklippare" som Ardumower. Det har konstruerats ett fungerande positioneringssystem som använder sig av trilateration med ultraljud. Prestandan är dock inte tillräcklig för att systemet ska kunna appliceras direkt på en robotgräsklippare. Det har dessutom utvecklats en ruttplanerande algoritm som är applicerbar på robotgräsklipparen. Algoritmen tillsammans med positioneringssystemet skulle göra att robotgräsklipparen drar mycket mindre energi än dagens robotgräsklippare.

Det går dock inte säga att det har skapats en robotgräsklippare med ruttplanering. Anledningen till att detta mål inte nåddes var dels tidsbrist, men också att det positioneringssystem som togs fram inte har den prestanda som behövs för en robotgräsklippare. Detta gäller både positioneringssystemets ytkapacitet och dess precision.

Slutligen kan sägas att om det tas fram ett positioneringssystem med erforderlig prestanda, så skulle konceptets sista pusselbit vara på plats.

Fortsättningsvis följer en mer detaljerad slutsats om de olika delarna.

5.1 Gräsklippare

I detta avsnitt analyseras den färdiga gräsklipparens olika delar och dess resultat.

5.1.1 Chassi

Chassit fungerar till stor del mycket bra. Problemet med att den främre delen slår i hinder/väggar vid rotation är svårt att lösa. Det som kan göras är att placera de drivande hjulen i centrum på chassit, och/eller minska längden på chassit så mycket som möjligt. En central placering av drivhjulen skulle innebära en försämrad stabilitet men kan vara värd att överväga. Att korta chassits längd skulle förbättra chassits rotationspotential men skulle samtidigt minska ytan för att placera elektronik.

Det ska också beaktas att chassit inte har någon överdel. Det finns alltså ingenting som skyddar den mot regn eller annat som kan tänkas träffa gräsklipparen uppifrån. För kontinuerlig användning borde detta tillverkas.

5.1.2 Drivning och styrning

I inledningen sas det att "ingen tid kommer att spenderas på att få gräsklipparen att köra upp för backar", vilket det heller inte har gjorts. Trots detta klarar gräsklipparen att köra upp för rejäla lutningar, och samtidigt hålla en skapligt rak kurs. Då inga utförliga tester har gjorts för backkörning går det inte att ge några exakta siffror på hur stora backar som gräsklipparen klarar av på en gräsmatta. Den slutsats som går att dra av de enkla tester som gjorts, är att gräsklipparen inte bör ha några problem gräsmattor innehållande backar upp till 20° lutning, som samtidigt inte är högre än tre meter i fallhöjd. Med detta sagt ska det beaktas att gräsklipparen inte kommer att kunna bibehålla denna prestanda på en ojämn gräsmatta (exempelvis innehållande mindre stenar och rötter).

Angående den avvikelse för förekommer i kurs, beror denna troligtvis på att de drivande hjulen inte alltid har perfekt grepp mot underlaget. PID-kontrollern ser till att varvtalen på de två utgående motoraxlarna är desamma, men om ett av hjulen slirar något uppstår en avvikelse i kurs. Med tanke på att konceptet med detta projekt var att styra gräsklipparen med hjälp av ett positioneringssystem, så gjordes inga övriga försök för att reglera kursen. Tanken var att positioneringssystemet skulle hjälpa gräsklipparen att hålla kursen, och därav hade ett ytterligare reglersystem för att bibehålla kursen varit överflödig.

5.1.3 Klippaggregat

De testkörningar som gjordes gav överlag ett mycket bra resultat. Klippaggregatet fick endast problem vid två tillfällen; vid väldigt högt och tjockt gräs, samt då den fastnade i en grop. Detta är en viktig punkt att åtgärda i framtiden. För att lösa detta går det att implementera ett system som detekterar om klingan är blockerad då motorn samtidigt försöker att rotera. Annars kan strömmen bli hög och komponenter kan ta skada. Gräsklipparen klippte annars genomgående gräset till en höjd på 3 cm. Detta resultat är tillräckligt bra för att klippa många gräsmattor. Att den körde fast i väldigt högt och tjockt gräs anses inte vara ett betydande problem då en robotgräsklippare är avsedd att kontinuerligt klippa gräset, och därmed hålla det kort. Gräset bör aldrig bli så högt som 20 cm, och kommer därför alltid kunna klippas av gräsklipparen utan problem. Det ska dock noteras att det finns en risk för att gräsklipparen kör fast om gräsmattan innehåller gropar. Då det inte har gjorts något långtidstest går det dessutom inte med säkerhet att säga att gräsklipparen klarar att klippa en hel gräsmatta utan problem. Ett problem som eventuellt skulle uppstå är att det fastnar så mycket gräs på knivarna så att de inte kan klippa gräset på ett bra sätt. För att garantera att detta inte uppstår måste ett långtidstest göras.

5.2 Hinderdetektering

Gräsklipparen hanterade i de absolut flesta fall hinder på ett bra sätt. Det hände som sagt att den inte lyckades detektera ett hinder i tid, och därav körde på det innan den stannade och backade. Utifrån de tester som genomförts så bedöms dock sannolikheten för att gräsklipparen skulle köra fast i hinder, som mycket liten. Även om hindret inte alltid upptäcks i tid så verkar det alltid upptäckas till slut. Den sammanlagda slutsatsen av detta är att systemet fungerar tillräckligt bra för att fungera på en gräsmatta som inte innehåller ömtåliga föremål, då dessa i annat fall skulle kunna skadas. Utöver detta får gräsmattan inte innehålla för branta sluttningar, då dessa i så fall skulle kunna uppfattas som hinder. Även genomskinliga ytor såsom glasrutor och liknande kunde försämra precisionen på IR-sensorerna. Det ska också beaktas att föremål med en höjd på under 8 cm riskerar att inte bli detekterade alls, och därmed överkörda.

Med hänsyn till ovanstående observationer kan slutsatsen dras om att detta systemet inte har tillräckligt hög precision för att användas på en robotgräsklippare. Det skulle därmed gå att använda andra system, exempelvis sådana som använder sig av en “bumper”. Alltså att de vänder först när de kör in i något.

5.3 Positioneringssystem

Positioneringssystemet fungerade bra till viss del. Det gav ofta rätt position med en bra precision, vilket tyder på att systemet faktiskt fungerade. Inom den lilla yta där positioneringssystemet fungerade gav det precisa positioner - så länge mottagaren inte tappade kontakten med någon av fyrarna. Anledningen till den mycket begränsade ytan, samt de förekommande avvikelserna, kan i första hand härledas till de dåligt presterande ultraljudssändarna. Ultraljudssändarna hade en mycket snäv spridningsvinkel, cirka 10°. Anledningen till att de ändå ansågs intressanta att testa, var att den specificerade vinkeln antogs gälla för ultraljudssensorer. Det antogs alltså att spridningsvinkeln för en ultraljudssändare skulle vara större än den vinkeln som specificeras för ultrajudssensorer, då signalen inte behöver reflekteras på vägen till mottagaren. Så var dock inte fallet, och i efterhand hade en djupare litteraturstudie av ljuds spridning varit användbar. Även ultraljudsmottagarna hade en mindre lyssningsvinkel än vad som hade förväntats. Ultraljudsmottagarnas lyssningsvinkel uppmättes till ungefär 90°. Detta faktum försämrar inte positioneringssystemet i lika stor utsträckning som ultraljudssändarna, men är ändå under förväntan. Det skulle alltså innebära att minst fyra stycken ultraljudsmottagare skulle behövas för att få en lyssningsvinkel på 360°.

Angående testerna bör det tilläggas att det inte oproblematiska utförandet kan ha inneburit felkällor. Testerna utfördes genom att först mäta upp alla mått för hand med ett måttband och sedan flyttades positioneringssystemet för hand över rektangeln. Det faktum att så mycket gjordes för hand, kan mycket väl ha försämrat de framtagna resultaten.

Det befintliga positioneringssystemet skulle kunna förbättras genom att öka antalet sändare och mottagare. Med flera sändare, samt ytterligare två mottagare skulle den nu mycket begränsade ytan kunna förstoras avsevärt. En annan lösning skulle kunna vara att konstruera ett system där sändarna och mottagaren monteras på axlar som roterar i lagom hastighet med hjälp av servomotorer. På detta sätt så skulle mottagaren kunna ta emot signaler i 360°, och sändarna skulle kunna sända över en större vinkel. Alternativt skulle sändarna kunna riktas mot gräsklipparen med hjälp av servomotorerna. Detta borde medföra att sändarnas begränsade spridningsvinkel inte längre får någon större inverkan. Vidare skulle kraftigare ultraljudssändare eventuellt kunna förbättra positioneringssystemet, eftersom att de ultraljudssensorer som användes endast kunde arbeta vid relativt låga spänningar. Eventuellt skulle de två misslyckade kretsarna (se avsnitt 3.3.3) ha gett en större anpassningasbarhet för spänningen till ultraljudssändaren. De skulle även ha kunnat ge ökade möjligheter för filtrering och förstärkning av den mottagna signalen.

I och med bristande tid, gjordes inga utomhustester för positioneringssystemet. Antagligen skulle positioneringssystemet ge sämre resultat utomhus på grund av vind och varierande temperatur. Genom att komplettera positioneringssystemet med en termometer, så skulle det kunna beräkna ljudets hastighet för olika temperaturer. Den varierande utomhustemperaturen skulle därav inte längre vara ett problem. Även vinden som förekommer utomhus skulle kunna kompenseras för med en vindsensor.

Det är inte endast hårdvaran som kan förändras för att få täckningssytan för positioneringssystemet att öka, utan även programmeringskoden kan utvecklas så systemet blir bättre. Genom en smart placering av ultraljudssändare och med en smart programmeringkod, skulle det gå att ha vissa mindre delar av gräsmattan utanför fyrarnas spridningsvinkel. Robotgräsklipparen kommer då att tappa signalen när den kör in i en sådan del av gräsmattan, men vet då istället sin position genom att den vet riktningen den körde i när den körde in i ytan. Den kan då också kommunicera med motorerna och genom det veta om den ändrar riktning och så den vet vilken hastighet den har. Med dessa parametrar kan den då själv räkna ut var den befinner sig. När gräsklipparen sedan återfår signalen från ultraljudssändarna uppdateras dess position till den verkliga (om den möjligtvis skulle få något fel på positionen i ytan utanför fyrarnas spridningsvinkel, exempelvis att den slirar). En lösning lik denna skulle eventuellt vara tillräcklig för att gräsklipparen ska lyckas klippa en hel gräsmatta på ett bra sätt.

En förbättring som är relativt enkel att implementera är en redigering av mottagarens kod. Då koden, för att få ett bra resultat, kräver kontakt med alla fyra sändare, är systemet mycket känsligt. Trilaterationsalgoritmen kan i sig hantera alltifrån kontakt med tre sändare, och fler. En redigering av mottagarens kod, som gör att den inte alltid skickar parametrar från fyra fyrar, skulle därför förbättra resultatet.

Gällande fyrarna är de inte riktigt färdigkonstruerade. För att de ska kunna an-

vändas på ett enkelt sätt behövs någon typ av ställning till dem. Detta för att få fyrarna stabila och för att få upp sändarna en bit från marken. Ställningarna bör också konstrueras för att klara av flera olika väderförhållanden såsom regn, snö och kraftig vind. På grund av tidsbristen hanns inte detta med, utan Arduinon med ultraljudssändarkretsen har endast placerats på stolar och dylikt vid tester. Fyrarna har drivits med ström från stationära datorer, genom USB-kablar. De färdiga fyrarna var tänkta att vara kopplade till var sitt 9 V batteri. Detta skulle göra fyrarna helt trådlösa.

Huruvida ett positioneringssystem med ultraljud bör appliceras på en kommersiell robotgräsklippare är för övrigt något oklart. Ultraljudssignaler är inte optimalt för avståndsmätning i utomhusbruk. Detta eftersom ultraljud kan påverka djurlivet på och runt trädgården. Det har inte funnits några lagar som begränsar detta, men huruvida det är etiskt rätt kan övervägas.

5.4 Ruttplanering med hinderhantering

Algoritm C fungerar till stor del bra och kan i princip implementeras direkt med en robotgräsklippare som har ett positioneringssytem. Den kan hantera hinder och stora områden på ett bra sätt. Det finns dock fortfarande potential till förbättring. För att Algoritm C ska bli mer optimal så måste den kompletteras med en "shortest-pathfinding-algoritm". På det sättet så håller den reda på när den "hoppar", och kan då ta den snabbaste vägen tillbaka - istället för att gå samma väg som den kom ifrån - till nästa punkt.

Då Algoritm C lär sig att ta en effektivare väg andra gången, jämfört med den första, så är det ändå inte säkert att den nya vägen är den mest optimala. Det som kan vidareutvecklas är att algoritmen testar alla möjliga fall den kunnat göra, och sedan jämföra dem med varandra. Jämförelserna skulle då gå ut på att jämföra antal steg i varje fall, och det fall med minst antal steg skulle kunna väljas. Å andra sidan är kanske inte antalet steg det enda som bör värderas. Kanske bör jämförelsen också ta hänsyn till saker som exempelvis antalet svängar.

För att vidareutveckla algoritmen så skulle maskinlärningsystemet kunna kompletteras så att det fortsätter att lära sig och optimera rutten för varje gång robotgräsklipparen körs. På detta sätt skulle en närmast optimal rutt kunna uppnås, till skillnad från vad som fås med nuvarande algoritm som bara förbättrar rutten en gång.

5.5 Kostnad

Då inköpskostnaden av komponenter uppgick till 4969 kr höll sig projektet precis inom den uppsatta budgeten på 5000 kr. Även det material som erhölls av prototyplabbet höll sig inom den separata budgeten på 2000 kr då materialets uppskattade värde uppskattades till endast 300 kr.

Det ska också benämnas att Arduinos lånades av institutionen. Dessa Arduinos var betydligt dyrare än vad som hade behövts för projektet. Därför budgeterades det med billigare Arduinokopior med liknande prestanda.

Ett mål med projektet var att kostnaden för prototypen skulle vara låg i förhållande till andra robotgräsklippare på marknaden. Detta mål nåddes då den totala kostnaden, 5269 kr, är betydligt lägre än priserna för majoriteten av marknadens robotgräsklippare. Något som bör beaktas är dock att utvecklandet av robotgräsklipparen krävde många timmar av arbete. I och med detta är det rimligt att jämföra kostnaden för prototypen med inköpskostnaden för en annan "bygg-själv-robotgräsklippare", exempelvis Ardumower. En byggsats till en Ardumower kostar idag 469,90 €, med dagens valutakurser cirka 4500 kr [66]. Denna kostnad är alltså något lägre än kostnaden för prototypen, men det ska då beaktas att kostnaden för prototypen innefattar kostnaden för ett positioneringssystem som inte följer med Ardumower. Om kostnaden för positioneringssystemet exkluderas, resulterar detta i att kostnaden för prototypen är betydligt lägre än kostnaden för en Ardumower.

Referenser

[1] Illustrerad Vetenskap, "När uppfanns gräsklipparen?", 2012, [Online]. Tillgänglig: http://illvet.se/teknologi/uppfinningar-och-patent-nar-uppfanns-grasklipparen. Hämtad: 2017-01-25.

[2] H. Rådmark, "Så väljer du rätt robotgräsklippare 2016", *smarto.se*, 2016, [Online]. Tillgänglig: http://www.smarto.se/livsstil/sa-valjer-du-ratt-robotgrasklippare-2016/. Hämtad: 2017-01-29.

[3] Robotnyheter, "3 nya robotgräsklippare från Husqvarna", 2013, [Online]. Tillgänglig: http://robotnyheter.se/2013/02/06/3-nya-robotgrasklippare-fran-husqvarna/. Hämtad: 2017-01-29.

[4] Husqvarna AB, *HUSQVARNA AUTOMOWER® 420/430X/450X BRUKSANVISNING*, [Online]. Tillgänglig: http://service.webec.husqvarna.net/documents/HUSO/HUSO2016_EUsv/HUSO2016_EUsv__1157866-01.pdf. Hämtad: 2017-01-29, 2016.

[5] Företagsbladet, "Robotgräsklipparen – Hur prototypen blev en storsäljare", 2014, [Online]. Tillgänglig: http://foretagsbladet.se/articles/view/robotgrasklipparen-hur-prototypen-blev-en-storsaljare. Hämtad: 2017-01-25.

[6] Robotgräsklippare24, "Lutning på robotgräsklippare", [Online]. Tillgänglig: http://robotgräsklippare24.se/lutning-pa-robotgrasklippare/. Hämtad: 2017-01-25.

[7] Husqvarna, "Produktinformation Robotgräsklippare", 2017, [Online]. Tillgänglig: http://www.husqvarna.com/se/produkter/robotgrasklippare/25ejanuari. Hämtad: 2017-01-25.

[8] Wikipedia, "Local positioning system", 2016, [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Local_positioning_system. Hämtad: 2017-03-23.

[9] Prisjakt, "Prisjämförelse Robotgräsklippare", 2017, [Online]. Tillgänglig: https://www.prisjakt.nu/kategori.php?b=s310229786. Hämtad: 2017-01-29.

[10] C. Benson, "Driver motor sizing tutorial", *robotshop.com*, 2012, [Online]. Tillgänglig: http://www.robotshop.com/blog/en/drive-motor-sizing-tutorial-3661. Hämtad: 2017-03-06.

[11] Acquris, “Vad är PWM styrning?”, [Online]. Tillgänglig: http://www.acquris.se/media/index.php?id=36. Hämtad: 2017-03-24.
[12] National Instruments, “What is a Pulse Width Modulation (PWM) Signal and What is it Used For?”, [Online]. Tillgänglig: http://digital.ni.com/public.nsf/allkb/2AC662081E01E1AE86257F5D00011B6D. Hämtad: 2017-03-21.
[13] A. Gobor, “What is PWM and how does it work?”, *ekwb.com*, 2016, [Online]. Tillgänglig: https://www.ekwb.com/blog/what-is-pwm-and-how-does-it-work/. Hämtad: 2017-04-25.
[14] Protostack, “ATMEGA168A PULSE WIDTH MODULATION – PWM”, 2011, [Online]. Tillgänglig: https://protostack.com.au/2011/06/atmega168a-pulse-width-modulation-pwm/. Hämtad: 2017-04-25.
[15] Robotsidan, “Styrning”, [Online]. Tillgänglig: http://www.robotsidan.se/steering.html. Hämtad: 2017-03-27.
[16] J. Klemets och S. Siffrén, “Telepresence Robot”, 2011, Vasa: Yrkeshögskolan NOVIA. [Online]. Tillgänglig: https://publications.theseus.fi/bitstream/handle/10024/37816/Klemets_Jonatan_Siffren_Sebastian.pdf?sequence=1. Hämtad: 2017-03-27.
[17] 42 Bots, “Principle of operation of differential drive”, 2014, [Online]. Tillgänglig: http://42bots.com/tutorials/differential-steering-with-continuous-rotation-servos-and-arduino/. Hämtad: 2017-04-20.
[18] Wikipedia, “Structure of an H bridge (highlighted in red)”, 2017, [Online]. Tillgänglig: https://en.wikipedia.org/wiki/H_bridge. Hämtad: 2017-03-24.
[19] Carnegie Mellon Robot Academy, “What is an IR Sensor?”, [Online]. Tillgänglig: http://education.rec.ri.cmu.edu/content/electronics/boe/ir_sensor/1.html. Hämtad: 2017-03-08.
[20] Electronics Hub, “IR sensor”, 2015, [Online]. Tillgänglig: http://www.electronicshub.org/ir-sensor/. Hämtad: 2017-03-08.
[21] A. Bensky, “Wireless Positioning Technologies and Applications”, 2008, Norwood, USA: Artech House Publishers.
[22] F. Gustafsson och F. Gunnarsson, “Positioning using time-difference of arrival measurements”, 2003, Linköping: Linköping universitet. [Online] Tillgänglig: http://users.isy.liu.se/en/rt/fredrik/reports/03icasspgustafsson.pdf. Hämtad: 2017-03-21.
[23] NMT Electrical Engineering Department, “Time of arrival notes”, 2008, [Online]. Tillgänglig: http://www.ee.nmt.edu/~rison/ee389_spr08/toa.pdf. Hämtad: 2017-03-21.
[24] W. Palm och K. Åhlfeldt, “APS - Akustiskt positioneringssystem”, 2014, Södertälje: Kungliga tekniska högskolan. [Online]. Tillgänglig:

http://www.diva-portal.se/smash/get/diva2:744923/FULLTEXT01.pdf. Hämtad: 2017-03-08.

[25] J. Moritz, "Inomhuspositionering med Bluetooth Low Energy", 2016, Uppsala: Uppsala universitet. [Online]. Tillgänglig: http://www.diva-portal.org/smash/get/diva2:1064632/FULLTEXT01.pdf. Hämtad: 2017-03-08.

[26] GISGeography, "Trilateration vs Triangulation – How GPS Receivers Work", 2017, [Online]. Tillgänglig: http://gisgeography.com/trilateration-triangulation-gps/. Hämtad: 2017-03-21.

[27] S. Russel och P. Norvig, "Artificial Intelligence - A modern approach", vol. Third edition, 2010.

[28] D. Heap, "Depth-first search (DFS)", *Cs.toronto.edu.*, 2012 , [Online]. Tillgänglig: http://www.cs.toronto.edu/~heap/270F02/node36.htm. Hämtad: 2017-05-12.

[29] Arduino, "Getting Started with Arduino and Genuino products", 2017, [Online]. Tillgänglig: https://www.arduino.cc/en/Guide/HomePage. Hämtad: 2017-04-20.

[30] C. Benson, "Arduino vs Raspberry Pi", *robotshop.com*, 2012, [Online]. Tillgänglig: http://www.robotshop.com/blog/en/arduino-vs-raspberry-pi-17699. Hämtad: 2017-03-07.

[31] B. Schneider, "A guide to Understanding LiPo Batteries", *rogershobbycenter.com*, 2017, [Online]. Tillgänglig: https://rogershobbycenter.com/lipoguide/. Hämtad: 2017-04-25.

[32] Ardumower, [Online]. Tillgänglig: http://www.ardumower.de/index.php/en/. Hämtad: 2017-03-01.

[33] John, "Brushed motor v brushless motors", *radiocontroltips.com*, [Online]. Tillgänglig: http://www.radiocontroltips.com/brushed-motor-v-brushless-motors/. Hämtad: 2017-04-27.

[34] C. McGrady, "Brush or Brushless? Which DC Motor Should You Choose?", *arrow.com*, 2016, [Online]. Tillgänglig: https://www.arrow.com/en/research-and-events/articles/which-dc-motor-is-best-for-your-application. Hämtad: 2017-04-27.

[35] Complete Powerelectronics, "BLDC vs DC Motor - Comparison between BLDC Motor and Conventional DC Motor", 2015, [Online]. Tillgänglig: http://www.completepowerelectronics.com/pmbldc-vs-dc-motor-comparison-between-bldc-motor-and-conventional-dc-motor/. Hämtad: 2017-04-27.

[36] J. Marks, "Spur versus planetary gearheads for dc servomotors", *machinedesign.com*, 2000, [Online]. Tillgänglig: http://www.machinedesign.com/archive/spur-versus-planetary-gearheads-dc-servomotors. Hämtad: 2017-04-27.

[37] Ardumover Wiki, “Wheel Motor - www.wiki.ardumover.de”, 2016, [Online]. Tillgänglig: http://wiki.ardumower.de/index.php?title=Wheel_Motor. Hämtad: 2017-04-25.

[38] Robotshop, “Lyxmotion 12V 90 rpm 99.11oz-in 1:26.9 Brushed DC Gear Motor w/ Encoder”, 2017, [Online]. Tillgänglig: http://www.robotshop.com/en/lynxmotion-12v-90-rpm-9911oz-in-1269-brushed-dc-gear-motor-w--encoder.html. Hämtad: 2017-04-27.

[39] Dimension Engineering, “Sabertooth dual 12A motor driver”, 2017, [Online]. Tillgänglig: https://www.dimensionengineering.com/products/sabertooth2x12. Hämtad: 2017-04-27.

[40] Dimension Engineering, “Kangaroo x2 motion controller”, 2017, [Online]. Tillgänglig: https://www.dimensionengineering.com/products/kangaroo. Hämtad: 2017-04-27.

[41] D.-G. Călin, “2 Simple Methods to Choose Motors for Wheel Drive Robots”, *intorobotics.com*, 2013, [Online]. Tillgänglig: https://www.intorobotics.com/2-simple-methods-choose-motors-wheel-drive-robots/. Hämtad: 2017-05-02.

[42] P.-O. Johansson, “Val av motorer”, *pojpoj.se*, 2015, [Online]. Tillgänglig: http://pojpoj.se/klippare%20motorval.html. Hämtad: 2017-05-02.

[43] MediaWiki, “Motor driver”, *wiki.ardumower.de*, 2017, [Online]. Tillgänglig: http://wiki.ardumower.de/index.php?title=Motor_driver. Hämtad: 2017-05-02.

[44] Marotronics, “Likströmsmotor 63 25 24 V, 8mm axel”, 2017, [Online]. Tillgänglig: https://www.marotronics.de/DC-Motor-63-25-24-V-8mm-Welle. Hämtad: 2017-05-02.

[45] Robotshop, “RS-755”, 2017, [Online]. Tillgänglig: http://www.robotshop.com/en/rs-755-12v-4791oz-in-5600rpm-brushed-dc-motor.html. Hämtad: 2017-05-02.

[46] Robotshop, “RS-555”, 2017, [Online]. Tillgänglig: http://www.robotshop.com/en/rs-555-12v-6100-rpm-brushed-dc-motor.html. Hämtad: 2017-05-02.

[47] Robotshop, “Cytron 13A, 5-25V Single DC Motor Controller”, 2017, [Online]. Tillgänglig: http://www.robotshop.com/en/cytron-13a-5-30v-single-dc-motor-controller.html. Hämtad: 2017-05-02.

[48] Jelena Mišić, Bratislav Milovanović, Nikola Vasić och Ivan Milovanović, “An Overview of Wireless Indoor Positioning Systems”, 2015, [Online]. Tillgänglig: http://infoteh.etf.unssa.rs.ba/zbornik/2015/radovi/KST/KST-3.pdf. Hämtad: 2017-04-04.

[49] R. Modir och A. Rydberg, “Inomhuspositionering – En användarstudie med fokus på detaljhandel”, 2014, Stockholm: Stockholms universitet. [Online]. Tillgänglig:

https://people.dsv.su.se/~romo4805/resurser/Examen.pdf. Hämtad: 2017-04-06.

[50] Kavli Institute for Cosmological Physics, "Radio Wave Basics", 2002, [Online]. Tillgänglig: https://kicp.uchicago.edu/education/explorers/2002summer-YERKES/pdfs-sum02/background.pdf. Hämtad: 2017-04-06.

[51] Kerry D. Wong, "A Sensitive DIY Ultrasonic Range Sensor", 2011, [Online]. Tillgänglig: http://www.kerrywong.com/2011/01/22/a-sensitive-diy-ultrasonic-range-sensor/.
Hämtad: 2017-05-05.

[52] Electronics Tutorials, "Active Band Pass Filter", 2017, [Online]. Tillgänglig: http://www.electronics-tutorials.ws/filter/filter_7.html. Hämtad: 2017-05-09.

[53] Professor Erik Cheever, "Single Supply Op Amps", 2017, [Online]. Tillgänglig: http://www.swarthmore.edu/NatSci/echeeve1/Ref/SingleSupply/SingleSupply.html. Hämtad: 2017-05-05.

[54] arduino, "analogRead()", 2017, [Online]. Tillgänglig: https://www.arduino.cc/en/Reference/analogRead. Hämtad: 2017-05-05.

[55] Forum Arduino, "How exactly slow is digitalRead()?", 2015, [Online]. Tillgänglig: https://forum.arduino.cc/index.php?topic=337578.0.
Hämtad: 2017-05-08.

[56] Giorgos Lazaridis, "The Schmitt Trigger", 2009, [Online]. Tillgänglig: http://www.pcbheaven.com/wikipages/The_Schmitt_Trigger/?p=1.
Hämtad: 2017-05-08.

[57] pcbheaven, "Dr. Calculus technical calculators", 2009, [Online]. Tillgänglig: http://pcbheaven.com/drcalculus/index.php?calc=st_nonsym_sp.
Hämtad: 2017-05-08.

[58] arduino, "constants", 2017, [Online]. Tillgänglig: https://www.arduino.cc/en/reference/constants. Hämtad: 2017-05-09.

[59] Kjell Company, "Avståndsmätare för Arduino", 2017, [Online]. Tillgänglig: https://www.kjell.com/se/sortiment/el-verktyg/elektronik/arduino/moduler/avstandsmatare-for-arduino-p87891ProductDetailedInformation.
Hämtad: 2017-05-09.

[60] Parallax Inc., "PING))) Ultrasonic Distance Sensor", 2017
, [Online]. Tillgänglig: https://www.parallax.com/product/28015. Hämtad: 2017-05-09.

[61] Emil, "Making a better HC-SR04 Echo Locator", 2014, [Online]. Tillgänglig: https://uglyduck.ath.cx/ep/archive/2014/01/Making_a_better_HC_SR04_Echo_Locator.html. Hämtad: 2017-05-09.

[62] A. Norrdine, "An Algebraic Solution to the Multilateration Problem", 2012, Aachen: Aachen university. [Online]. Tillgänglig: https://www.researchgate.net/publication/275027725_An_Algebraic_Solution_to_the_Multilateration_Problem. Hämtad: 2017-04-04.

[63] Raspberry Pi Foundation, “Raspberry Pi 3 Model B”, *raspberrypi.org*, [Online]. Tillgänglig: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`. Hämtad: 2017-04-04.

[64] MathWorks, “System Requirements for MATLAB R2017a”, 2017, [Online]. Tillgänglig: `https://se.mathworks.com/support/sysreq.html`. Hämtad: 2017-04-04.

[65] A. Norrdine, “Trilateration problem”, 2012, [Online]. Tillgänglig: `https://www.researchgate.net/publication/275027725_An_Algebraic_Solution_to_the_Multilateration_Problem`. Hämtad: 2017-04-20.

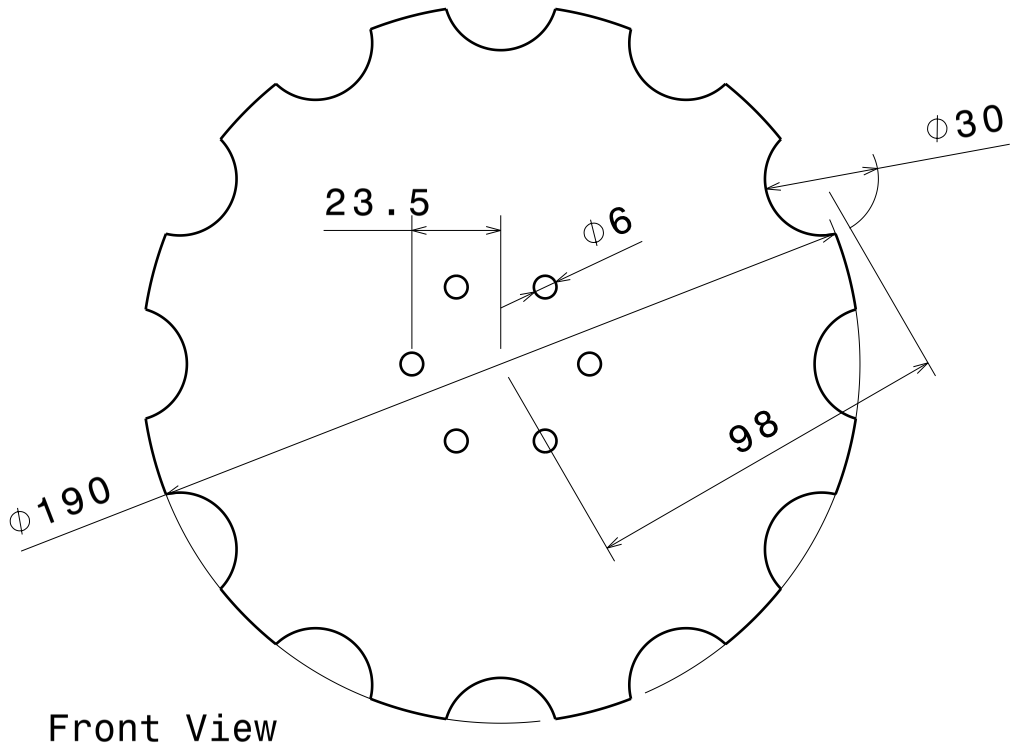
[66] Euroinvestor, “Valutaomvandlare”, *valuta.se*, 2017, [Online]. Tillgänglig: `http://www.valuta.se/`. Hämtad: 2017-04-24.

Appendix

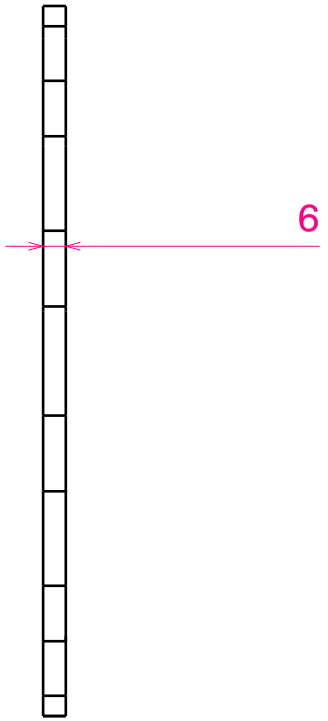
A

Ritningar

A.1 Bakhjul



Front View
Scale: 1:2

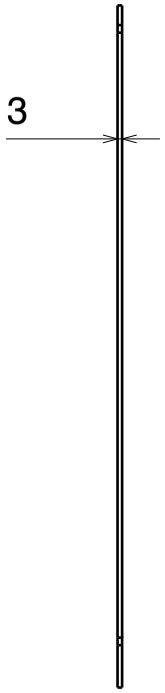


Left view
Scale: 1:2

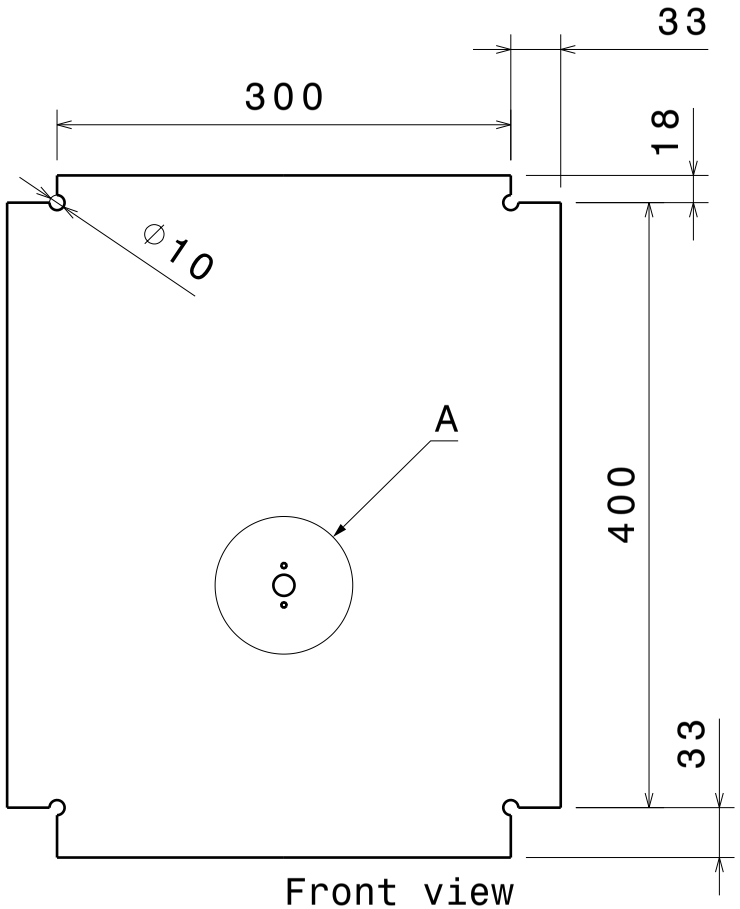
DESIGNED BY: Anton Gustafsson DATE: 2017-03-27		Chalmers Tekniska Högskola	
CHECKED BY: XXX DATE: XXX			
SIZE A4		Robotgräsklippare	
SCALE 1:1	WEIGHT (kg) -	DRAWING NUMBER Bakhjul	SHEET 1/1

I	_
H	_
G	_
F	_
E	_
D	_
C	_
B	_
A	_

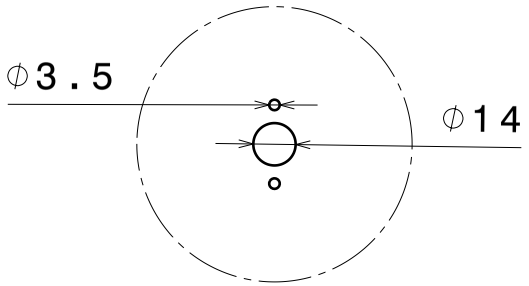
A.2 Bottenplatta



Right view
Scale: 1:5



Front view
Scale: 1:5



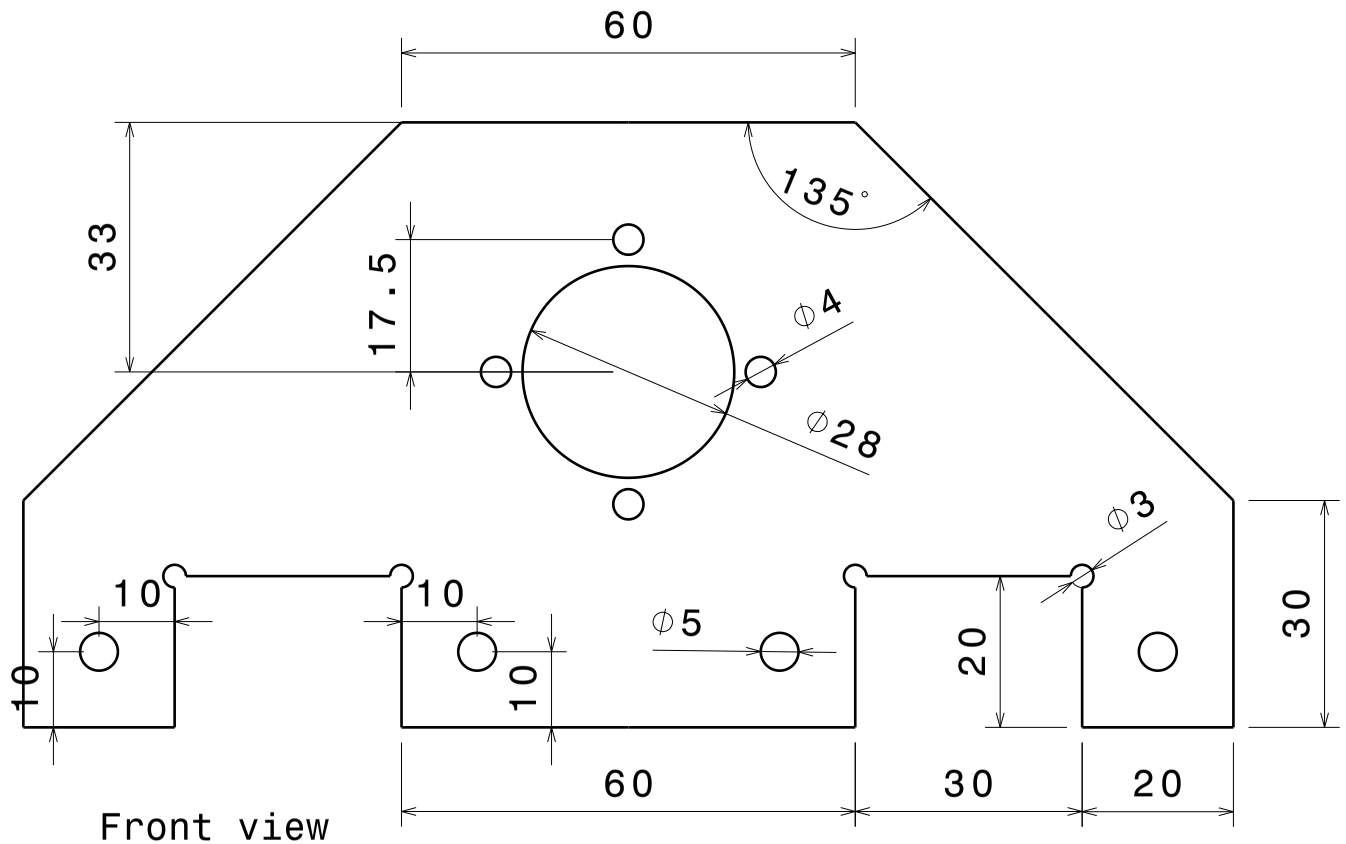
Detail A
Scale: 2:5

DESIGNED BY: Anton Gustafsson DATE: 2017-03-27		Chalmers Tekniska Högskola	
CHECKED BY: XXX DATE: XXX			
SIZE A4		Robotgräsklippare	
SCALE 1:1	WEIGHT (kg) -	DRAWING NUMBER Bottenplatta för bockning	SHEET 1/1

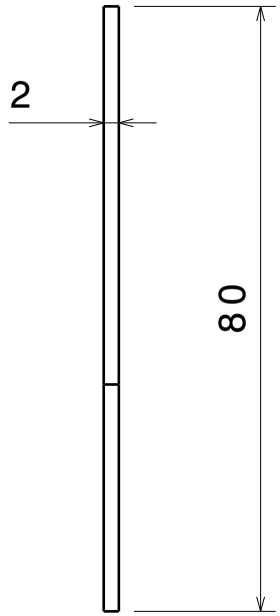
This drawing is our property; it can't be reproduced or communicated without our written agreement.

I	_
H	_
G	_
F	_
E	_
D	_
C	_
B	_
A	_

A.3 Motorfäste



Front view
Scale: 1:1



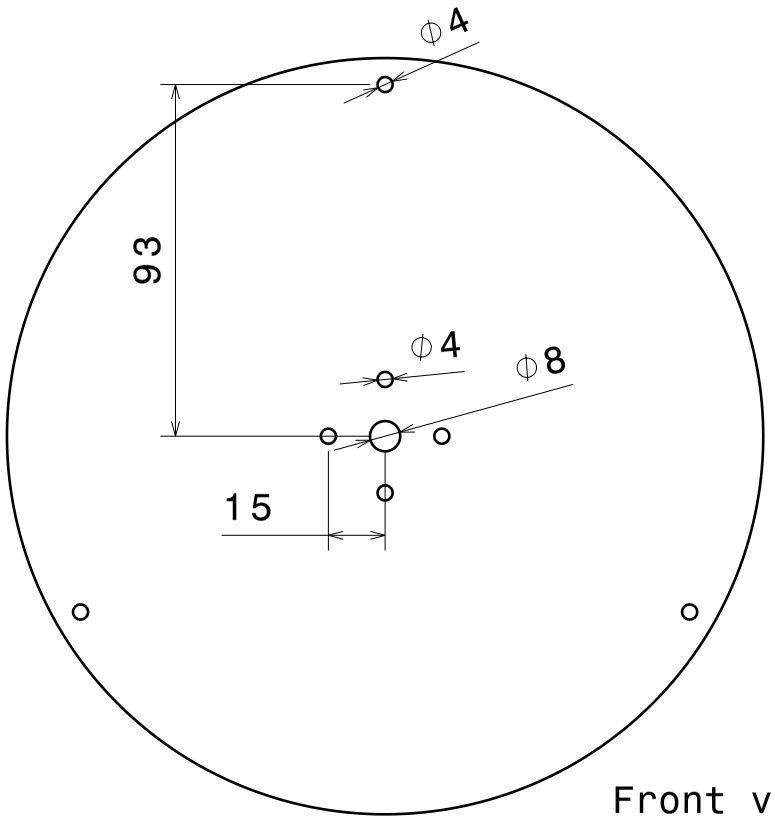
Left view
Scale: 1:1

DESIGNED BY: Anton DATE: 2017-03-27		Chalmers Tekniska Högskola	
CHECKED BY: XXX DATE: XXX			
SIZE A4		Robotgräsklippare	
SCALE 1:1	WEIGHT (kg) -	DRAWING NUMBER Motorfäste för bockning	SHEET 1/1

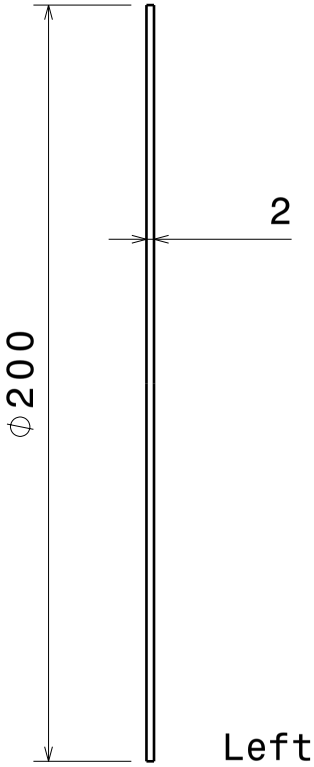
This drawing is our property; it can't be reproduced or communicated without our written agreement.

I	_
H	_
G	_
F	_
E	_
D	_
C	_
B	_
A	_

A.4 Klippdisk



Front view
Scale: 1:2

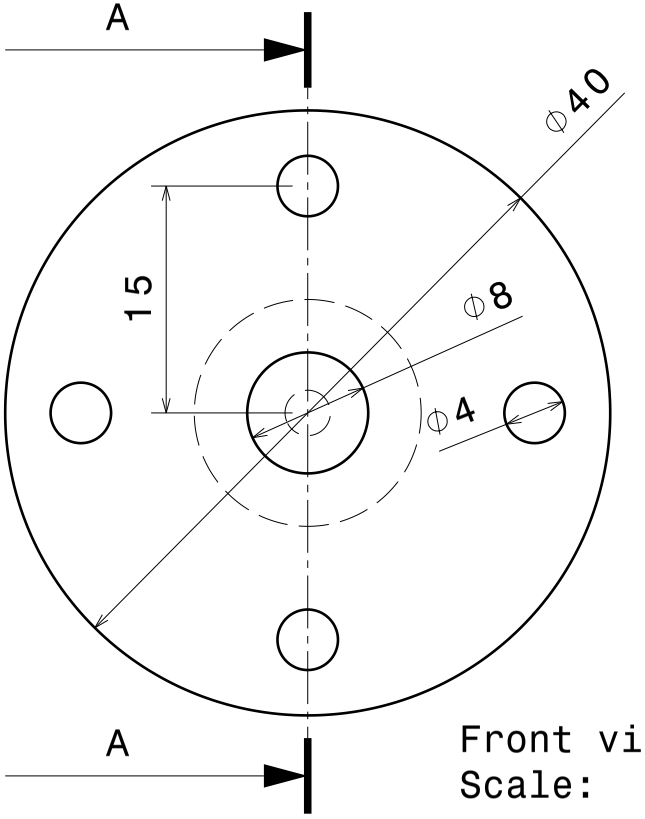


Left view
Scale: 1:2

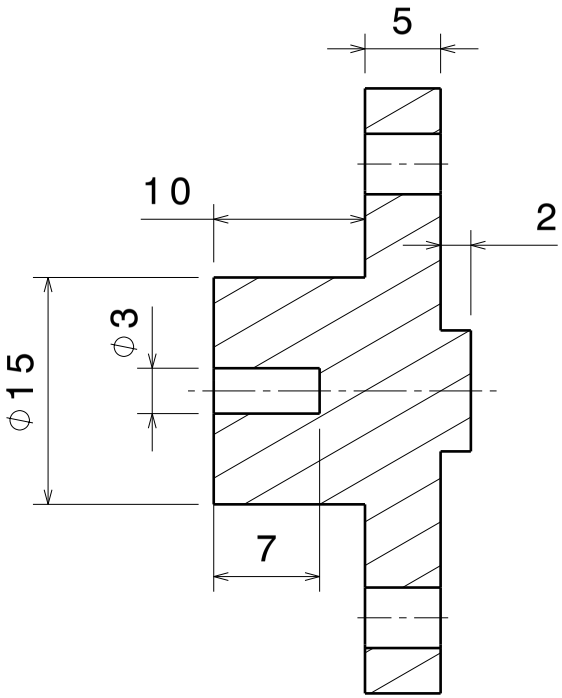
DESIGNED BY: Anton DATE: 2017-03-27		Chalmers Tekniska Högskola	
CHECKED BY: XXX DATE: XXX			
SIZE A4		Robotgräsklippare	
SCALE 1:1	WEIGHT (kg) -	DRAWING NUMBER Klippdisk	SHEET 1/1
This drawing is our property; it can't be reproduced or communicated without our written agreement.			

I	_
H	_
G	_
F	_
E	_
D	_
C	_
B	_
A	_

A.5 Klippnav



Front view
Scale: 2:1



Section view A-A
Scale: 2:1

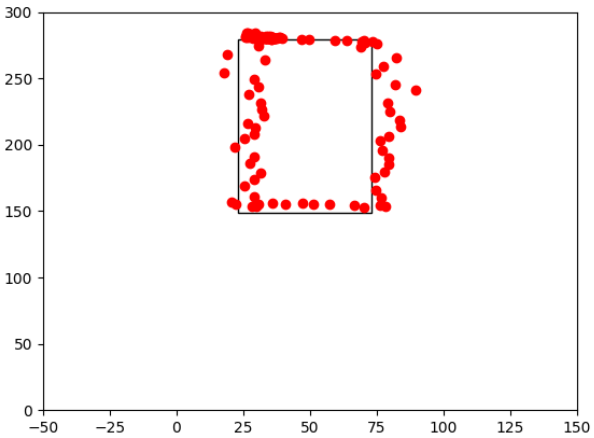
DESIGNED BY: Anton DATE: 2017-03-27		Chalmers Tekniska Högskola	
CHECKED BY: XXX DATE: XXX			
SIZE A4		Robotgräsklippare	
SCALE 1:1	WEIGHT (kg) -	DRAWING NUMBER Klippnav	SHEET 1/1

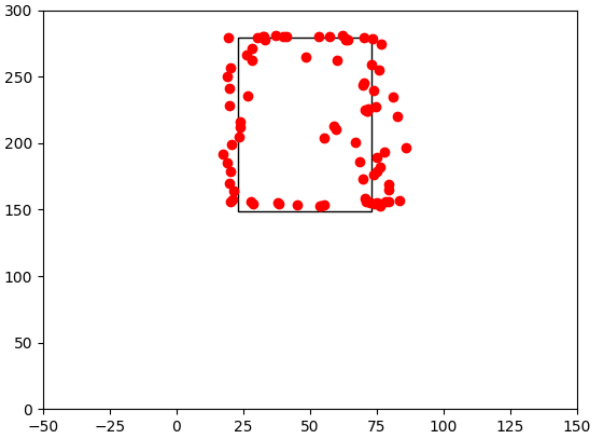
I	_
H	_
G	_
F	_
E	_
D	_
C	_
B	_
A	_

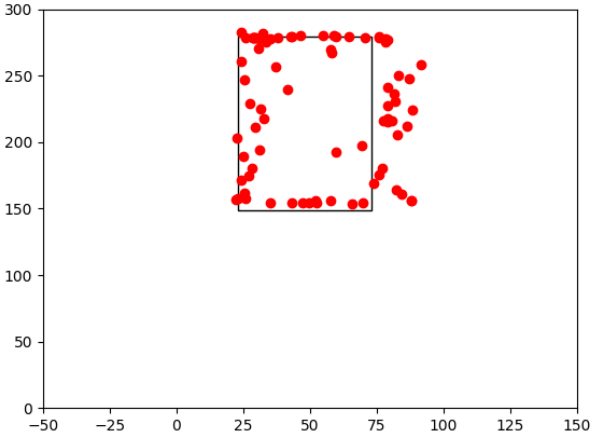
B

Tester

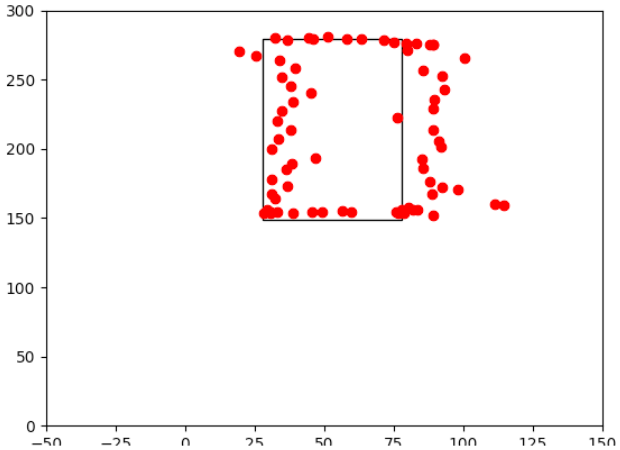
B.1 Positioneringssystem testuppsättning 1

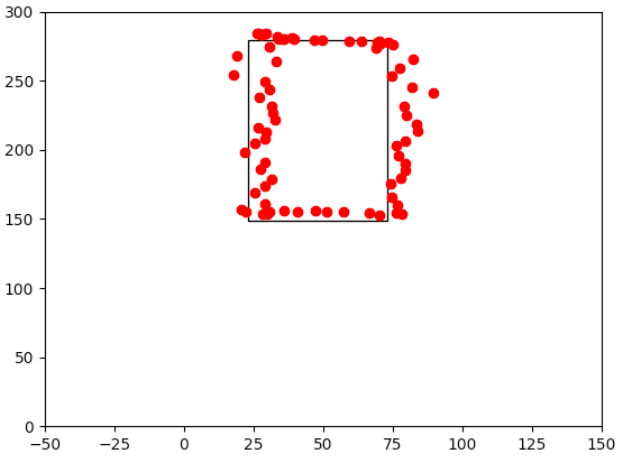


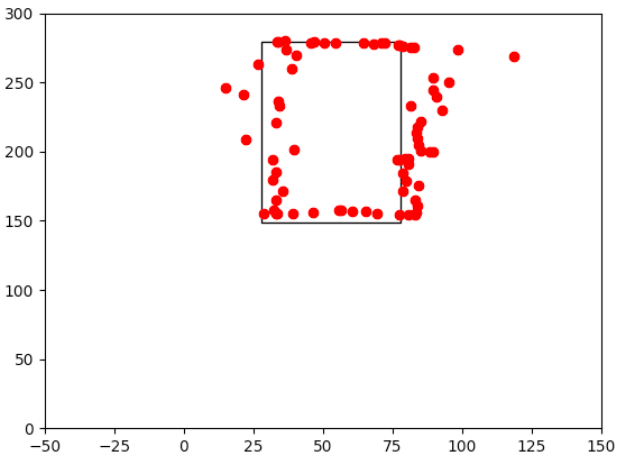




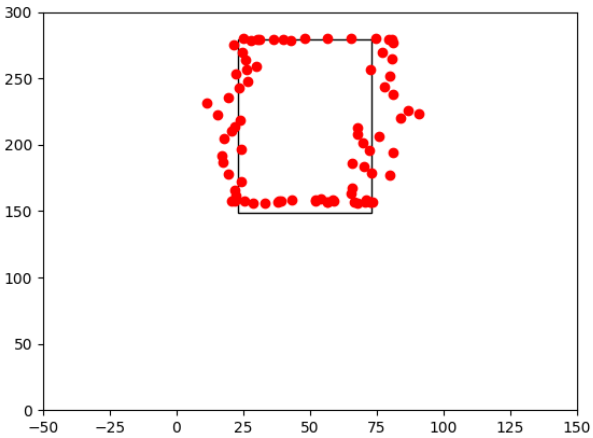
B.2 Positioneringssystem testuppsättning 2

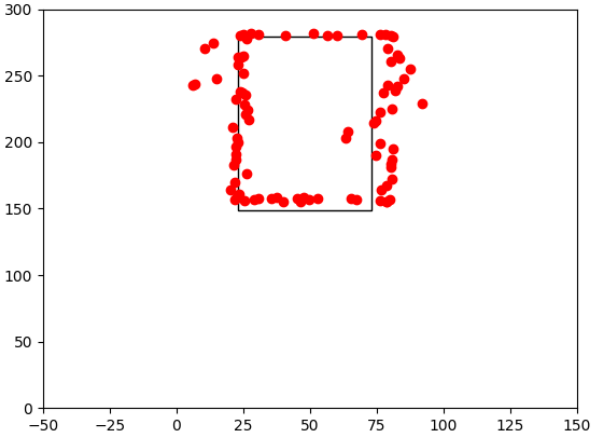






B.3 Positioneringssystem testuppsättning 3





C

Kod

C.1 Trilaterationsalgoritm

```
import numpy as npy
import matplotlib.pyplot as plt
import time as time

"""
Trilateration algorithm (Translated from MATLAB)
paper "An algebraic solution to the multilateration problem"
Author: Norrdine, Abdelmoumen (norrdine@hotmail.de)
Translator: Gustafsson, Anton
https://www.researchgate.net/publication/275027725
   _An_Algebraic_Solution_to_the_Multilateration_Problem
usage: [N1 N2] = Trilateration(P,S,W)
P = [P1 P2 P3 P4 ..] Reference points matrix
S = [s1 s2 s3 s4 ..] distance matrix.
N : calculated solution
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY!! """


def null(A, eps=1e-15):
    """
    http://mail.scipy.org/pipermail/scipy-user/2005-June
       /004650.html
    """
    u, s, vh = npy.linalg.svd(A)
    n = A.shape[1]   # the number of columns of A
    if len(s)<n:
        expanded_s = npy.zeros(n, dtype = s.dtype)
        expanded_s[:len(s)] = s
        s = expanded_s
    null_mask = (s <= eps)
    null_space = npy.compress(null_mask, vh, axis=0)
    return npy.transpose(null_space)

t0 = time.clock()

def getPos(P,S):
    [mp, np] = P.shape
    ns = S.size

    if ns != np:
        print('Number of reference points and distances are
           different')

    x = P[0, 0]
```

```
y = P[1, 0]
z = P[2, 0]

s = S[0]

b = npy.array([s ** 2 - x ** 2 - y ** 2 - z ** 2])
A = npy.array([1, -2 * x, -2 * y, -2 * z])

for i1 in range(1, np):
    x = P[0, i1]
    y = P[1, i1]
    z = P[2, i1]

    s = S[i1]

    A = npy.append(A, [1, -2 * x, -2 * y, 2 * z], 0)
    b = npy.append(b, [s ** 2 - x ** 2 - y ** 2 - z **
        2], 0)

b.shape = (np, 1)
A.shape = (np, 4)

if np == 3:
    Xp = npy.linalg.lstsq(A, b)
    Xp = Xp[0]
    xp = Xp[1:]
    Z = null(A)
    z = Z[1:]
    if npy.linalg.matrix_rank(A) == 3:
        a2 = z[0] ** 2 + z[1] ** 2 + z[2] ** 2
        a1 = 2 * (z[0] * xp[0] + z[1] * xp[1] + z[2] *
            xp[2]) - Z[0]
        a0 = xp[0] ** 2 + xp[1] ** 2 + xp[2] ** 2 - Xp
            [0]
        p = [a2[0], a1[0], a0[0]]
        t = npy.roots(p)

        N = npy.array([Xp + t[0] * Z, Xp + t[1] * Z])
        N = N[0,1:3]
        return N

if np > 3:
    pinvA = npy.linalg.pinv(A)
    Xpdw = npy.matmul(pinvA, b)

    N = npy.array([Xpdw[1], Xpdw[2]])
```

```
        return N

def PlotCircle(x,y,r):
    th = npy.linspace(0, 2 * npy.pi, 101)
    fig = plt.figure(1)
    plt.axis([0, 100, 0, 100])
    ax = fig.add_subplot(1, 1, 1)
    circ = plt.Circle((x, y), radius=r, color='g', fill=
       False)
    ax.add_patch(circ)

"Fyrarnas positioner"
P1 = npy.array([0, 0, 0])
P2 = npy.array([100, 100, 0])
P3 = npy.array([100, 0, 0])
P4 = npy.array([0, 100, 0])

"De uppm tta avst nden"
s1 = 80
s2 = 70
s3 = 110
s4 = 30

P = npy.array([P1, P2, P3, P4])
P = npy.transpose(P)
(m,n) = P.shape

S = npy.array([s1, s2, s3, s4])

N = getPos(P,S)


print('Tid f r ber kning: ')
print(time.clock() - t0)
print('L sning: ')
print(N)


"Plot"
for k in range(0,n):
    PlotCircle(P[0,k],P[1,k],S[k])
plt.plot(N[0],N[1],'ro')
plt.show()
```

C.2 Ruttplaneringsalgoritmer

```
# coding=utf-8
import matplotlib.pyplot as plt
import numpy as np
import time
import py_compile
from collections import deque
from heapq import heappop, heappush
#import serial
#ser = serial.Serial('/dev/tty.usbmodem1411', 9600)

###
# Global Values
###
fig = 1
image = []
xl = 0
yl = 0

def initialize(nrows,ncols):
    global fileExist
    fileExist = False
    global image
    image = np.zeros(nrows*ncols)
    global futurePQ
    futurePQ = []
    image = image.reshape((nrows, ncols))
    setWalls()
    plt.figure()
    global fig
    fig = plt.matshow(image,fignum = 0)
    image[4,4] = 2 #f r att ndra f rgerna f r
       simuleringar
    plt.ylim(plt.ylim()[::-1]) # reverse y-axis
    plt.xticks(range(ncols), range(ncols)) # range g r att
       man g r 0-args
    plt.yticks(range(nrows), range(nrows))
    plt.xticks(np.arange(-.5, ncols, 1))
    plt.yticks(np.arange(-.5, nrows, 1))
    # interactive mode on
    plt.ion()
    # visar plotten
    # pausar so den hinner rita ut
    plt.pause(0.0001)
    global fileMap
```

```
    try:
        with open('yardmap.tex','r'):
            print("file exist")
            fileMap = open('yardmap.tex','r')
            fileExist = True

            pass
    except FileNotFoundError:
        print('File not existing, making a new...')
        fileMap = open('yardmap.tex', 'w')

###
# S tt ut v ggar f r simulering endast!
###
def setWalls():
    image[5,5] = 2
    image[4,4] = 6

def update():
    fig.set_data(image)
    #plt.imshow(image,interpolation='nearest') # f r
       anv ndning av A*
    plt.draw()
    #plt.matshow(image, fignum = 0)
    plt.pause(0.0001)
    #time.sleep(0.1) # F r att delaya plotten

###
# Algoritm B
###
def recDFS(x, y, string, dweg):
    if image[x][y] == 0:
        image[x][y] = 1
        old = image[x][y]
        image[x][y] = 5
        update()
        image[x,y] = old
        futurePQ.append([(x,y)])
        dweg += 1
        if y > 0:
            recDFS(x,y-1,"left",dweg)
        if x > 0:
            recDFS(x-1,y,"down",dweg)
        if x < xl - 1:
            recDFS(x+1,y,"up",dweg)
        if y < yl - 1:
```

```
            recDFS(x,y+1,"right",dweg)
    return(futurePQ)

###
# Kortaste V gen mellan tv  punkter, Astar algoritm
# taget fr n http://bryukh.com/labyrinth-algorithms/
###
def heuristic(cell, goal):
    return abs(cell[0] - goal[0]) + abs(cell[1] - goal[1])


def find_path_astar(maze):
    start, goal = (1, 1), (len(maze) - 2, len(maze[0]) - 2)
    pr_queue = []
    heappush(pr_queue, (0 + heuristic(start, goal), 0, "",
       start))
    visited = set()
    graph = maze2graph(maze)
    while pr_queue:
        _, cost, path, current = heappop(pr_queue)
        if current == goal:
            return path
        if current in visited:
            continue
        visited.add(current)
        for direction, neighbour in graph[current]:
            heappush(pr_queue, (cost + heuristic(neighbour,
               goal), cost + 1,
                                path + direction, neighbour)
                                   )
    return "NO WAY!"

###
# Hittar en ny punkt l ngt ifr n nuvarande
# Genom att leta efter n rmsta grannen i minnet
###
def findPath(tuplez,futurePQ):
    #tuplez[0] = prev
    #tuplez[1] = curr
    px = tuplez[1][0][0]
    py = tuplez[1][0][1]
    for k in [[px,py-1],[px+1,py],[px,py+1],[px-1,py]]:
        if(k[0] >= 0 and k[0] < xl-1 and k[1] >= 0 and k[1]
           < yl-1):
            if(image[k[0]][k[1]] == 1 and [(k[0],k[1])] in
               futurePQ ):
```

```
                indexP = futurePQ.index([(k[0],k[1])])
                indexc = futurePQ.index(tuplez[0])
                return(indexP,indexc)
    print("NOTHING")

###
# Algoritm C - Online-DFS-algoritm anpassad till projektet
###
def DFS2(x,y):
    stack = deque([[(x,y)]])
    visited = []
    index = 0
    prev = ([(x,y)])
    while stack:
        current = stack.pop()
        if current in visited:
            continue
        x = current[0][0]
        y = current[0][1]
        if(abs(prev[0][0] - x) > 1 or abs(prev[0][1] - y) >
           1 or (abs(prev[0][0] - x) >= 1 and abs(prev[0][1]
            - y) >= 1)  ):
            tuplepath = findPath([prev,current],visited)

            reversePath = (visited[tuplepath[0]:tuplepath
                [1]][::-1])
            visited = visited[:tuplepath[1]] + reversePath +
                visited[tuplepath[1]:]
            reversePath = deque(reversePath)
            walkPath(reversePath)
        prev=([(x,y)])
        old = image[x][y]
        image[x][y] = 5
        update()
        image[x,y] = old
        visited.append([(x,y)])
        image[(x,y)] = 1
        if((x,y+1) not in visited and image[x][y+1] == 0 and
            y+1 < yl-1 ):
            stack.append([(x,y+1)])
        if((x+1,y) not in visited and image[x+1][y] == 0 and
            x+1 < xl-1 ):
            stack.append([(x+1,y)])
        if((x-1,y) not in visited and image[x-1][y] == 0 and
            x-1 >= 0 ):
            stack.append([(x-1,y)])
```

```
        if((x,y-1) not in visited and image[x][y-1] == 0 and
            y-1 >= 0 ):
            stack.append([(x,y-1)])
        index += 1

    return visited

###
# G r om en grid till en graf
# taget fr n http://bryukh.com/labyrinth-algorithms/
###
def maze2graph(maze):
    height = len(maze)
    width = len(maze[0]) if height else 0
    graph = {(i, j): [] for j in range(width) for i in range
       (height) if not maze[i][j]}
    for row, col in graph.keys():
        if row < height - 1 and not maze[row + 1][col]:
            graph[(row, col)].append(("S", (row + 1, col)))
            graph[(row + 1, col)].append(("N", (row, col)))
        if col < width - 1 and not maze[row][col + 1]:
            graph[(row, col)].append(("E", (row, col + 1)))
            graph[(row, col + 1)].append(("W", (row, col)))
    return graph

###
# L ser fr n minnet
###
def readPath():
    fileMap.read()
    print(fileMap)

###
# G r en sicksack-sp r t algoritm A
###
def makePath(x,y):
    print("check if fime exist")
    if(fileExist):
        print("map exist, taking from map...")
        readPath()

    print("file does not exist, making a new one")

    queue = deque([(x,y)])
    index = 1
    counter = 1
```

```
    while(counter < (xl*yl)):
        counter += 1
        if y > 0 and y < yl-1:
            y += index
            queue.append(([y,x]))
        elif y == yl-1 and index < 0:
            y += index
            queue.append(([y,x]))
        elif y == 0 and index > 0:
            y += index
            queue.append([y,x])
        else :
            if x >= 0 and x < xl-1:
                x += 1
                index = index * (-1)
                queue.append([y,x])
    return queue

###
# Algoritm A som g r efter sicksack-karta
###
def walkPath(queue):
    print("this is queue", queue)
    xold = 0
    yold = 0
    while len(queue) > 0:

        node = queue.popleft()
        print("this is noode", node)
        try:
            x = node[0]
            y = node[1]
        except IndexError:
            x = node[0][0]
            y = node[0][1]

        old = image[x][y]
        image[x][y] = 5
        update()
        image[x,y] = old
        objectAvoidz(x,y,xold - x, yold-y, queue)
        if(image[x,y] != 2):
            image[x,y] = 1
        xold = x
        yold = y
```

```
###
# Hinderhantering, "rundar" objektet och n rliggande.
###
def objectAvoidz(x,y,xo,yo,queue):
    l = []
    print("avoid",x,y)
    print("is object?", image[x][y],image[x][y] == 2)
    if image[x][y] == 2:
        print("obstacle at ",x,y,xo,yo)
        if xo > 0:
            print("down")
            #drive("forward");
            l = [[x-1,y],[x-1,y-1],[x,y-1],[x+1,y-1]]
        elif xo < 0:
            print("up")
            #drive("forward");
            l = [[x+1,y-1],[x,y-1],[x-1,y-1],[x-1,y]]
        elif yo > 0:
            print("left")
            #drive("left");
            l = [[x,y-1],[x-1,y-1],[x-1,y],[x-1,y+1]]
        elif yo < 0:
            print("right")
            #drive("left")
            l = [[x-1,y+1],[x-1,y],[x-1,y-1],[x,y-1]]

        for i in l:
            if(i[1] > 0 and i[1] < xl-1 and i[0] > 0 and i
               [0] < yl-1):
                print("adding", l)
                if(image[i[0]][i[1]] != 2):
                    queue.appendleft((i[0],i[1]))
                else:
                    objectAvoidz(i[0],i[1],x-i[0],y-i[1],
                       queue)

###
# Optimerar en rutt s  den blir b ttre
###
def learnQueue(futurePQ):
    learnedQueue = []
    for i in range(0,len(futurePQ)):
        if i < len(futurePQ)-1:

            try:
                x = futurePQ[i][0][0]
```

```
                try:
                    y = futurePQ[i][0][1]
                except IndexError:
                    continue
                x + 1

            except TypeError:
                x = futurePQ[i][0][0][0]
                y = futurePQ[i][0][0][1]
            try:
                xn = futurePQ[i+1][0][0]
                try:
                    yn = futurePQ[i+1][0][1]
                except IndexError:
                    continue
                xn + 1

            except TypeError:
                xn = futurePQ[i+1][0][0][0]
                yn = futurePQ[i+1][0][0][1]

            if (abs(x-xn >= 2) and abs(y-yn >= 2)):
                popedNode = futurePQ.pop(i+1)
                px = popedNode[0][0]
                py = popedNode[0][1]

                for k in [[px,py-1],[px+1,py],[px,py+1],[px
                    -1,py]]:
                    if(k[0] >= 0 and k[0] < xl-1 and k[1] >=
                        0 and k[1] < yl-1):
                        if(image[k[0]][k[1]] == 1 and [(k
                            [0],k[1])] in futurePQ ):
                            indexP = futurePQ.index([(k[0],k
                                [1])])
                            futurePQ = futurePQ[:(indexP+1)]
                                + [popedNode] + futurePQ[(
                                indexP+1):]
                            break
            else:
                learnedQueue.append(futurePQ[i])
    return(futurePQ)

###
# Sparar kartan
###
def saveMap():
```

```
    fileMap.write('['+(''.join(str(e) for e in image))+']')

###
# Sparar sina steg
###
def saveQueue(futurePQ):
    futurePQ = learnQueue(futurePQ)
    string = str(futurePQ)
    fileMap.write(string)

###
# G r efter sitt minne av stegen
###
def getQueue():
    queue = deque([(0,0)])

    for i in fileMap.read().strip('[[').strip(']]').split
       (']', ['):
        i = i[1:]
        i = (i.strip('(').strip(')').strip('\n').strip(']]')
           .strip(')').strip('').split(','))
        i = list(map(int, i))
        queue.append(i)
    walkPath(queue)

###
# Huvudkoden
###
def main():
    global xl #hur l ngt i x-axeln
    global yl #hur l ngt i y-axeln
    xl = 10
    yl = 10
    initialize(xl,yl) #g r area av x och y
    #walkPath(makePath(0,0))
    if(fileExist):
        print("get queue")
        getQueue()
    else:
        print("dfs explore!")
        futurePQ = DFS2(0,0)  #Anv nd algoritm C
        #futurePQ = recDFS(0,0,"begin",0) #Anv nd algoritm
           B
        saveQueue(futurePQ)
main()
```

C.3 Arduino-kod för gräsklippare

```
#include <SoftwareSerial.h>
#include <Kangaroo.h>
#include <SharpIR2.h>

#define TX_PIN 8   // S1 orange
#define RX_PIN 7   // S2 brun
#define model 1080

const int cm = 8.1;
const float grad = 50/18;
int mvarde = 3;
int hastighet = 100;
double klipphast = 0.6;   // klipphast = [0 1], MIN = 0, MAX
    = 1

#define RANGE_SENSOR_LEFT      A0
#define RANGE_SENSOR_MID     A1
#define RANGE_SENSOR_RIGHT     A2

SharpIR SharpIRl(RANGE_SENSOR_LEFT, model);
SharpIR SharpIRm(RANGE_SENSOR_MID, model);
SharpIR SharpIRr(RANGE_SENSOR_RIGHT, model);

//SharpIR sensorArray[] = {sharpL, sharpM, sharpR};

// Mixed mode channels on Kangaroo are, by default, 'D' and
   'T'.
SoftwareSerial  SerialPort(RX_PIN, TX_PIN);
KangarooSerial  K(SerialPort);
KangarooChannel Drive(K, 'D');
KangarooChannel Turn(K, 'T');
/*
 * from what I understand of the h file you can call
    constructor w/ no parameters
 * and the library will take care of the magic behind the
    status object
 */
KangarooStatus Feedback;

void setup()
{
  pinMode(10, OUTPUT);

  Serial.println("Test");
```

XXX

```
  Serial.begin(9600);

  SerialPort.begin(9600);
  SerialPort.listen();

  Drive.start();
  Turn.start();

  Drive.si(0);
  Turn.si(0);

}

// Funktion f r IR-sensorer
void getDist(){
  int l=SharpIRl.distance(); // this returns the distance
     to the object you're measuring
  int m=SharpIRm.distance();
  int r=SharpIRr.distance();

  Serial.print("Sensor vanster: ");
  Serial.println(l);
  Serial.print("Sensor mitten: ");
  Serial.println(r);
  Serial.print("Sensor hoger: ");
  Serial.println(m);
  Serial.println();

  int lenght = 15;
  int lrlen = 25;

  if((l <= lrlen) || (m <= lenght) || (r <= lrlen)){
    if(mvarde == 0){
      Drive.pi(-20*cm, hastighet).wait();
      Turn.pi(120*grad,100).wait();
      Serial.println("INNE");
      mvarde = 3;
    }else
      mvarde = mvarde - 1;
  }else
    if(mvarde < 3)
      mvarde = mvarde+1;

}

// Funktioner f r klippmotor
```

```
void stopTransducer()
{
  //cli();
  TCCR1B = 0;
  sei();
  digitalWrite(9,LOW);
  digitalWrite(10,LOW);
}

void startTransducer(float freq, float dutyCycle)
{
  if (dutyCycle > 1.0) dutyCycle = 1.0;
  else if (dutyCycle < 0) dutyCycle = 0;

  cli();
  TCCR1B = _BV(WGM13) | _BV(CS10) | _BV(ICNC1);
  //f0 = fclk / (2 * N * Top)
  long topv = (long) ((float) F_CPU /(freq * 2.0 * 1.0));
  ICR1 = topv;


  OCR1A = (int) ((float) topv * dutyCycle);
  OCR1B = (int) ((float) topv * (1 - dutyCycle));
  DDRB |= _BV(PORTB1) | _BV(PORTB2);
  TCCR1A = _BV(COM1A1) | _BV(COM1B1);
  sei();
}

void loop(){
  ////////////////////////FIRST TEST KANGAROOSTATUS
     ////////////////////////////////////////////////
  /*
   * I believe all the propertis are set to zero in
      intialization....
   * Lets find out!
   */

  Feedback = Drive.getS(KANGAROO_GET_DEFAULT); //Can"t
     understand documenation=>need to play around with the
     getflgs
  Serial.println(Feedback.value());// This returns a int32_t
      => might need some formating/casting before printing
  Serial.print("The Kangaroo is");
  if(Feedback.ok()){
    Serial.println("good to go");
  }
```

```
  else{
    Serial.println("not ok");
  }

  ////////////////////////FIRST TEST KANGAROOSTATUS
     ////////////////////////////////////////////////
  /*
  // Drive.pi(position,speed).wait();
  //
  // Turn.pi(angle,speed).wait();
  */

  getDist();

  startTransducer(10000.0, 1-klipphast);
  delay(1000);

  Drive.pi(100*cm,hastighet);

}
```

C.4 Arduino-kod för positioneringssystem, mottagare

```
#include <VirtualWire.h>

// Robot, S ndare RF, Mottagere UL
// Parralax
long v = 0;
long a,b,c,d = 0;
long a2,b2,c2,d2 = 0;
char inData[50];
char mss[20];
int newmessage = 0;
long A,B,C,D = 0;
long A_2,B2,C2,D2 = 0;
int delayt = 30;

/* Ping))) Sensor

   This sketch reads a PING))) ultrasonic rangefinder and
      returns the
   distance to the closest object in range. To do this, it
      sends a pulse
   to the sensor to initiate a reading, then listens for a
      pulse
   to return.  The length of the returning pulse is
      proportional to
   the distance of the object from the sensor.

   The circuit:
    * +V connection of the PING))) attached to +5V
    * GND connection of the PING))) attached to ground
    * SIG connection of the PING))) attached to digital pin
       7

   http://www.arduino.cc/en/Tutorial/Ping

   created 3 Nov 2008
   by David A. Mellis
   modified 30 Aug 2011
   by Tom Igoe

   This example code is in the public domain.

 */
```

```
// this constant won't change.  It's the pin number
// of the sensor's output:
const int pingPin = 7;
const int pingPin2 = 4;

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  vw_setup(2000);
  vw_set_tx_pin(12);
}

long sendPing() {
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  long duration, cm;
  // The PING))) is triggered by a HIGH pulse of 2 or more
     microseconds.
  // Give a short LOW pulse beforehand to ensure a clean
     HIGH pulse:

  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, HIGH);
  digitalWrite(pingPin, LOW);
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);
  cm = microsecondsToCentimeters(duration);
  return(cm);
}
long sendPing2() {
  long duration2, cm2;
  pinMode(pingPin2, OUTPUT);
  digitalWrite(pingPin2, HIGH);
  digitalWrite(pingPin2, LOW);
  pinMode(pingPin2, INPUT);
  duration2 = pulseIn(pingPin2, HIGH);
  cm2 = microsecondsToCentimeters(duration2);
  return(cm2);
}


 long getDistanceA() {
     inData[0] = 'A';
     sprintf(mss, "%s", inData);
     vw_send((uint8_t *)mss, strlen(mss));
```

```
      vw_wait_tx();
         a = sendPing();
         a = a+0.0910176394749180*a+7.37498534927331;
  return(a);
}

 long getDistanceA2() {
      inData[0] = 'A';
      sprintf(mss, "%s", inData);
      vw_send((uint8_t *)mss, strlen(mss));
      vw_wait_tx();
         a2 = sendPing2();
         a2 = a2+0.0910176394749180*a2+7.37498534927331;
  return(a2);
}


 long getDistanceB() {
      inData[0] = 'B';
      sprintf(mss, "%s", inData);
      vw_send((uint8_t *)mss, strlen(mss));
      vw_wait_tx();
        b = sendPing();
        b = b+0.0910176394749180*b+7.37498534927331;
  return(b);
}

 long getDistanceB2() {
      inData[0] = 'B';
      sprintf(mss, "%s", inData);
      vw_send((uint8_t *)mss, strlen(mss));
      vw_wait_tx();
        b2 = sendPing2();
        b2 = b2+0.0910176394749180*b2+7.37498534927331;
  return(b2);
}

 long getDistanceC() {
      inData[0] = 'C';
      sprintf(mss, "%s", inData);
      vw_send((uint8_t *)mss, strlen(mss));
      vw_wait_tx();
        c = sendPing();
        c = c+0.0910176394749180*c+7.37498534927331;
  return(c);
}
```

```
 long getDistanceC2() {
     inData[0] = 'C';
     sprintf(mss, "%s", inData);
     vw_send((uint8_t *)mss, strlen(mss));
     vw_wait_tx();
       c2 = sendPing2();
       c2 = c2+0.0910176394749180*c2+7.37498534927331;
  return(c2);
}

 long getDistanceD() {
     inData[0] = 'D';
     sprintf(mss, "%s", inData);
     vw_send((uint8_t *)mss, strlen(mss));
     vw_wait_tx();
       d = sendPing();
       d = d+0.0910176394749180*d+7.37498534927331;
  return(d);
}

 long getDistanceD2() {
     inData[0] = 'D';
     sprintf(mss, "%s", inData);
     vw_send((uint8_t *)mss, strlen(mss));
     vw_wait_tx();
       d2 = sendPing2();
       d2 = d2+0.0910176394749180*d2+7.37498534927331;
  return(d2);
}

void loop()
{
  double p[3][4];
  for( int i=0; i <= 2; i++)
  {
  long test = 0;
  // get distance and then checks which side it is,
  // then we check if it mooved more than 30cm, then it is
     wrong and we
  // take the old value. the same goes if it reaches 624 or
     0.
  A = getDistanceA();
  delayMicroseconds(delayt);
  test = getDistanceA2();
    if(A > test){
```

```
      A = test;}
    if(abs(A_2 - A) < 30 || A_2 == 0 || A_2 == 624){
      A_2=A;}
  delayMicroseconds(delayt);

  B = getDistanceB();
  delayMicroseconds(delayt);
  test = getDistanceB2();
    if(B > test){
      B = test;}
    if(abs(B2 - B) < 30 || B2 == 0 || B2 == 624){
      B2=B;}

  delayMicroseconds(delayt);

  C = getDistanceC();
  delayMicroseconds(delayt);
  test = getDistanceC2();
    if(C > test){
     C = test;}
  if(abs(C2 - C) < 30 || C2 == 0 || C2 == 624){
    C2=C;}
  delayMicroseconds(delayt);

  D = getDistanceD();
  delayMicroseconds(delayt);
  test = getDistanceD2();
    if(D > test){
      D = test;}
  if(abs(D2 - D) < 30 || D2 == 0 || D2 == 624){
    D2=D;}
  p[i][0] = A_2;
  p[i][1] = B2;
  p[i][2] = C2;
  p[i][3] = D2;

}
  for( int i = 0; i <= 3 ; i++){
    double a = p[0][i];
    double b = p[1][i];
    double c = p[2][i];

    double min1 = min(a,b);
    double minsta = min(min1,c);
    double max1 = max(a,b);
    double storsta = max(max1,c);
```

```
    if(a > minsta && a < storsta){
      Serial.print(a);
    }
    else if (b > minsta && b < storsta){
      Serial.print(b);
    }
    else {
      Serial.print(c);
    }
      if(i != 3)
    Serial.print(',');
  else
    Serial.println();
  }

}


double microsecondsToCentimeters(long microseconds) {
  return (microseconds/29.15);

}
```

C.5 Arduino-kod för positioneringssystem, sändare

```
#include <VirtualWire.h>

// Beacon, S ndare ultraljud, Mottagere RF
// huvudsakligen HC-SR04

const int ledPin = 7;
const int pingPin = 8;

void setup()
{
delay(600);
Serial.begin(9600);
pinMode(ledPin, OUTPUT);
digitalWrite(ledPin, LOW);//turn LED on
vw_set_rx_pin(4);
Serial.println("Listening");
vw_setup(2000);
vw_rx_start();
}

void sendPing() {
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  long duration, cm;
  // The PING))) is triggered by a HIGH pulse of 2 or more
     microseconds.
  // Give a short LOW pulse beforehand to ensure a clean
     HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);          // 10us f r HC-SR04,
     paralax har kortare men funkar med v ran.
  digitalWrite(pingPin, LOW);
  Serial.println("sended");
}

void loop()
{
byte message[VW_MAX_MESSAGE_LEN];
byte messageLength = VW_MAX_MESSAGE_LEN;
```

```
  vw_wait_rx_max(300);
if (vw_get_message(message, &messageLength))
  {
  digitalWrite(ledPin, HIGH);//turn LED on
  if(message[0] == 'A') {      // Byt bokstav f r respektive
      fyr. a,b,c,d.
    sendPing();
    Serial.println("Fyr A"); // Debug
  }
  for (int i = 0; i < messageLength; i++)
    {
      Serial.write(message[i]);
    }
  Serial.println();
  digitalWrite(ledPin, LOW);//turn LED on
  }
}
```

D

Kopplingsscheman

D.1 Kopplingsschema motorer och styrkort