CHALMERS

Autonom robotgräsklippare

Autonom robotgräsklippare utvecklad för linjemålning och kamning av fotbollsplaner

Kandidatprojekt vid Avdelningen för System- och Reglerteknik
Johan Axelsson
Jonathan Carlson
Gustav Malmström
Jakob Olson
Felix Renberg
Emil Öhlund
INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2023
www.chalmers.se

Kandidatarbete 2023

Autonom robotgräsklippare

Autonom robotgräsklippare utvecklad för linjemålning och kamning av fotbollsplaner

Johan Axelsson
Jonathan Carlson
Gustav Malmström
Jakob Olson
Felix Renberg
Emil Öhlund

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg 2023

Autonom robotgräsklippare
Autonom robotgräsklippare utvecklad för linjemålning och kamning av fotbollsplaner
JOHAN AXELSSON
JONATHAN CARLSON
GUSTAV MALMSTRÖM
JAKOB OLSON
FELIX RENBERG
EMIL ÖHLUND

Handledare: Nikolce Murgovski, Institutionen för Mekatronik
Examinator: Jonas Fredriksson, Institutionen för Mekatronik

Kandidatarbete 2023
Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Husqvarna automower 450x

Skriven i LaTeX
Göteborg 2023

Sammanfattning

I dagens samhälle automatiseras många tjänster och en sådan tjänst är robotgräsklipparen. En robotgräsklippare är begränsad då den inte erbjuder några andra tjänster än gräsklippning. Den har också ineffektiv navigering och otillräcklig lokalisering av GPS-position. Målet med detta kandidatarbete är att få en robotgräsklippare att sköta flera av de arbetsuppgifter som finns kring fotbollsplansskötsel. Detta inkluderar, utöver att klippa gräs, även att måla linjer och kamma gräs med precision. Genom att automatisera tjänster såsom fotbollplansskötsel kan tid och ansträngning sparas, samtidigt som det kan leda till en ökad produktivitet och ekonomiska besparingar.
Till en början användes enbart modellen Husqvarna Automower 450x, men under projektets gång erhölls även modellen Husqvarna Automower 550 EPOS. Vidare utvecklades båda modellerna för att jämföra dessa eftersom dem har olika navigeringssystem. Projektet har utgått från Husqvarnas robotgräsklippare som har modifierats för att kunna genomföra målning och kamning. Ett stort fokus låg på att få robotgräsklipparna att kunna köra efter rutter med bästa möjliga precision. Resultatet av projektet var att robotgräsklipparen av modell 450x kunde köras efter en rak linje, men med en varierande precision på mellan 0,1-0,5 meter.
Cirkelkörning och svängar gick inte att realisera på grund av tidsbrist men fungerade bra i simulation. Kamning och färgmålning fungerar, men gräsklipparen kan inte följa en komplex rutt och därmed kan uppgifterna inte skötas autonomt av gräsklipparen. Med robotgräsklipparen av modell 550 EPOS lyckades inte positionen från basstationen finnas på ett korrekt sätt vilket innebar att den inte gav några resultat kring körning efter en rutt. Slutsatserna från projektet är bland annat att med Kalmanfilter gick det att förbättra precisionen anmärkningsvärt genom att använda odometri och GPS-data tillsammans. Precisionen är inte tillräcklig för att utföra syftet med att måla och kamma gräset autonomt. Huvudanledningen till detta var dålig precision på 450x GPS. Däremot har 550 EPOS potential att lösa tillräckligt nogrann navigering, även om det inte hanns med i detta projekt.

Nyckelord: Automower, 2D robot, reglering, RTK, EPOS, CAD, kalmanfilter, GPS, HRP, ROS, fotbollsplan, autonom, fotbollslinjer, kamning.

Abstract

In todays society, an increasing number of services are being automated, and one such service involves the utilization of robotic lawnmowers. However, a robotic lawnmower is inherently constrained, as it provides no services beyond grass cutting. Moreover, its navigation capabilities are inefficient, and its ability to precisely locate its GPS position is insufficient. The objective of this bachelor's project is to enhance a robotic lawnmower's functionality, enabling it to perform a range of tasks associated with the maintenance of football pitches. These tasks extend beyond mere grass cutting to also include the precise application of line marking and grass combing. By automating services such as football pitch maintenance, considerable time and effort can be saved, potentially leading to increased productivity and financial savings. Initially, only the Husqvarna Automower 450x model was used, but as the project progressed, the Husqvarna Automower 550 EPOS model was also incorporated. Both models were further developed and compared due to their differing navigation systems. The project was based on Husqvarna's robotic lawnmowers, which were modified to enable line marking and grass combing. A significant focus was placed on enabling the robotic lawnmowers to follow routes with the greatest possible precision. The project's outcome demonstrated that the 450x model could follow a straight line, albeit with a variable precision of between 0.1 and 0.5 meters. Circular paths and turns could not be realized due to time constraints, although they performed well in simulations. Both line marking and grass combing functionalities were operational; however, the lawnmower could not follow a complex route, precluding the autonomous execution of these tasks. The 550 EPOS model was unsuccessful in correctly positioning itself relative to the base station, thus providing no results regarding its ability to follow a route. The project's conclusions include the noteworthy improvement in precision achieved through the use of a Kalman filter, which combined odometry and GPS data. Nonetheless, the precision was not sufficient to autonomously execute line marking and grass combing tasks. The primary reason for this was the poor precision of the 450x model's GPS. Nevertheless, the 550 EPOS model possesses the potential to accomplish sufficiently precise navigation, although this was not realized within the scope of this project.

Förord

Vi vill börja med att tacka vår handledare Nikolce Murgovski som hjälpt oss under kandidatarbetet. Vi vill även rikta ett stort tack till Husqvarna Group som skänkt oss en robotgräsklippare av modell Husqvarna Automower 550 EPOS. Tack för all hjälp under mötena vi haft med er. Slutligen vill vi också tacka biblioteket och fackspråk för era tjänster och handledningar gällande rapportskrivande.

Johan Axelsson
Jonathan Carlson
Gustav Malmström
Jakob Olson
Felix Rehnberg
Emil Öhlund
Göteborg, Maj 2023

Akronymer

Listan nedan innehåller akronymer använda i rapporten. Akronymerna förekommer i alfabetisk ordning.

CAD	Computer Aided Design
EPOS	Exact Positioning Operating System
GPS	Global Positioning System
HRP	Husqvarna Research Platform
IMU	Inertial Measurement Unit
LQR	Linear Quadratic Regulator
PID	Proportional Integral Derivative
PWM	Puls Width Modulation
ROS	Robot Operating System
RTK	Real Time Kinematic

Innehåll

Akronymer **ix**

Figurer **xiv**

Tabeller **xvi**

1 Inledning **1**
1.1 Syfte . . . 2
1.2 Problem- och uppgiftbeskrivning . . . 2
1.3 Mål och delmål . . . 3
1.4 Avgränsningar och begränsningar . . . 4
1.5 Samhälleliga- och etiska aspekter . . . 4

2 Teori **6**
2.1 Fotbollsplan . . . 6
2.2 Husqvarna Automower . . . 7
2.2.1 Husqvarna Automower 450x . . . 8
2.2.2 Husqvarna Automower 550 EPOS . . . 9
2.3 RTK teknik . . . 9
2.4 Jetson Nano . . . 10
2.5 Raspberry pi . . . 10
2.6 ROS . . . 11
2.7 SSH . . . 11
2.8 Trilateration . . . 13
2.9 Kalmanfilter . . . 13
2.10 Differentiellstyrning . . . 15
2.11 Regulatorer . . . 16
2.11.1 PID-regulator . . . 16
2.11.2 LQR . . . 17
2.12 Pulsbreddsmodulering (PWM) . . . 18
2.13 Computer Aided Design (CAD) . . . 19

3 Metod **20**

3.1 Styrdator 20
3.1.1 Husqvarna Automower 450x 20
3.1.2 Husqvarna Automower 550 EPOS 21
3.2 Beräkning av position 22
3.2.1 Koordinatsystem 23
3.2.2 Kalmanfilter 27
3.2.3 Husqvarna Automower 550 EPOS 28
3.3 Programmeringskod 29
3.4 Reglering och styrning 31
3.4.1 PID 31
3.4.2 LQR 33
3.4.3 Styrning 34
3.5 Måla linjer 35
3.5.1 Längder av linjer 35
3.5.2 Ruttplanering 36
3.5.3 Färgspruta 38
3.6 Kamning 41
3.6.1 Ruttplanering 41
3.6.2 Kam 43

4 Resultat **44**
4.1 Navigering 44
4.1.1 Navigering 450x 44
4.1.2 Navigering 550 EPOS 46
4.2 Simulering 47
4.2.1 PID 47
4.2.2 LQR 48
4.3 Målning av linjer 49
4.4 Kamning av gräs 50
4.5 Kostnader 52
4.6 Måluppfyllnad 52

5 Diskussion **53**
5.1 Styra gräsklipparna 53
5.2 Navigera i sin omgivning 53
5.3 Reglering 55
5.4 Kamning och målning 55

6 Slutsats **56**

Litteraturförteckning **58**

A Programmeringskod **I**
A.1 main.py I
A.2 calc_velocities VII
A.3 kalman.py XI
A.4 coord_sys_trans.py XIII

A.5 paint.py XV
A.6 comb_cut.py XXVII
A.7 README.md XXVIII

Figurer

2.1 Figuren visar svenska fotbollsförbundets regler för dimensioner för en elvamannaplan [17]. 7
2.2 Figuren visar Husqvarna Automower 450x [19]. 8
2.3 Figuren visar Husqvarna Automower 550 EPOS [22]. 9
2.4 Figuren visar en Jetson Nano [26]. 10
2.5 Figuren visar en Raspberry pi 4 [29]. 11
2.6 Figuren visar signal från satellit 1, 2 och 3 [35]. 13
2.7 Figuren visar hur uppskattning av positionen i Kalmanfilter beräknas [38]. 15
2.8 Figuren visar en differentiellstyrning av en robot som kör olika riktningar beroende på hjulens rotation [41]. 15
2.9 Figuren visar en graf över hur referensvärdet följs beroende på parametrarna K_p, K_i och K_d [43]. 16
2.10 Figuren visar ett blockschema över en PID-regulator i en återkopplingsslinga. r(t) är det önskade processvärdet eller börvärdet (SP), och y(t) är det uppmätta processvärdet (PV) [43]. 17
2.11 Figuren visar hur LQR fungerar [45]. (Återgiven med tillstånd) 17
2.12 Figuren visar CAD modell av en robotgräsklippare. 19

3.1 Figuren visar hur kopplingen av CAN-adapter ser ut. 21
3.2 Figuren visar kopplingar mellan ett powerkort och en Raspberry pi. . 22
3.3 Med hjälp av denna figur kunde koordinattransformationen beräknas vilket visas i Ekvation 3.2. 24
3.4 Figuren visar kalibrering av GPS. Genom detta steg kan vinkeln beräknas mellan odometri och GPS. I nästkommande verkliga försök kan nu alla GPS-punkter roteras in i samma koordinatsystem som odometrin vilket möjliggör användning av Kalmanfiltret. 26
3.5 Figurerna visar två olika sätt att rotera GPS till odometri. Den vänstra figuren visar kalibrering genom att köra några meter och sedan beräkna vinkeln mellan GPS och odometri. Den högra figuren visar detta genom att först gå 60 meter med gräsklipparen och därav beräkna vinkeln. 26

3.6 Figuren visar första testet av Kalmanfiltret där Kalmanfiltret är applicerat efter testet var klart. Under testet användes endast odometridata. 28
3.7 Figuren visar en referensstation [49]. (Återgiven med tillstånd) 29
3.8 Figuren visar ett flödesschema över hur programmeringskoden styr gräsklipparen. Hela koden finns i Bilaga A.1. 31
3.9 Figuren visar hur målvinkeln beräknas för cirkelkörning. 32
3.10 Figuren visar första testen av LQR. 34
3.11 Figuren visar en modell för styrning av klipparen. 35
3.12 Figuren visar den lokala hemsidan där användaren enkelt kan ändra linjernas längd och antalet kamningslinjer. 36
3.13 Figuren visar målning av nedre kortlinje samt runt hörnflagga. 37
3.14 Figuren visar simulerad rutt för hela planen. I simuleringen används en P-regulator för att köra raka linjer och PID-regulator för att köra i cirklar. 38
3.15 Figuren visar gräsklipparen utan skal med färgsprutarkonstruktionen. 39
3.16 Figuren visar en servomotor monterad på hållaren för färgsprutan som vid rotation drar in avtryckaren för färgsprutan. 40
3.17 Figuren visar röret som ser till att färgsprutan målar rätt bredd på linjerna. 40
3.18 Figuren visar hur simuleringen ser ut när gräsklipparen har gått första iterationen över planen och sen tillbaka till starten. 41
3.19 Figurerna visar simulering med inzoomning så att en femtedel av planen syns. 42
3.20 Figuren visar när hela planen är kammad. Där de olika nyanserna innebär olika färdriktningar över planen. 42
3.21 Figuren visar gräsklipparen utan skal och första CAD av kam. 43
3.22 Figuren visar gräsklipparen utan skal och kamkonstruktion. 43

4.1 Figuren visar slutpositionen efter 40 meters körning med Kalmanfiltret utefter mittlilnjen. 45
4.2 Figuren visar GPS, odometri och Kalmanfiltrets väg vid körningen i Figur 4.1. 45
4.3 Figuren visar slutpositionen efter 40 meters körning med endast odometri utefter mittlilnjen. 46
4.4 Figurerna visar hur gräsklipparen av modell 550 åker efter den har fått en målposition. 47
4.5 Figurerna visar PID-regulatorn i två olika mönster med brus. Det slutgiltiga felet är felet från målet som gräsklipparen slutar på. 48
4.6 Figurerna visar hur LQR behandlar två olika mönster med brus. 48
4.7 Figuren visar när gräsklipparen försöker måla längs mittlinjen. 50
4.8 Figuren visar resultat av kamning i straffområdet. 51

Tabeller

4.1 Tabellen visar kostnaderna i projektet. 52
4.2 Tabellen visar de mål som angavs i början av projektet. 52

1
Inledning

Fotboll är en av världens största sporter med över 250 miljoner utövare i 200 länder enligt internationella fotbollsförbundet FIFA [1]. Utöver detta följs fotbollen av miljarder av människor från sidan. Fotboll är Sveriges populäraste sport som dagligen utövas av närmare 600 000 barn, ungdomar och vuxna människor i landet [2].

Många fotbollsplaner på lägre nivåer har oftast dåligt skött gräs som till exempel att gräset kan vara mycket högt och torrt. Idealt ska höjden på gräset ligga mellan 3-5 centimeter för bäst kvalité och det mest optimala är att endast klippa en tredjedel av gräset vilket betyder att gräset måste klippas två till fyra gånger i veckan [3]. Det krävs mycket arbete för en vaktmästare att hålla den kvalitet på gräsmattan som krävs för fotboll. År 2013 fanns det ungefär 3950 stycken naturgräsplaner i Sverige. Det var då 218 av 290 kommuner som hade investerat i konstgräs och siffrorna har bara ökat sen dess för antalet konstgräsplaner och därmed minskar planer med naturgräs [4]. I och med denna ökning av antalet konstgräsplaner kommer tillslut miljön att påverkas då mikroplaster kommer spridas ut i miljön [5]. Det har även visat sig att skaderisken är större för fotboll på konstgräs jämfört med naturgräs [6]. Det skulle därmed vara gynnsamt för både miljön och för fotbollsspelare att kunna spela på naturgräs.

I och med den teknologiska utvecklingen har en stor del av dagens arbeten automatiseras med hjälp av robotar. Inom 20 år är det beräknat att omkring 50% av dagens jobb i Sverige kommer att kunna ersättas av digital teknik [7]. För att spara pengar och tid på planskötsel skulle en autonom robotgräsklippare som sköter klippning och målning av linjer kunna vara en sådan automatisering. Robotgräsklippare har blivit allt vanligare de senaste åren, nu äger ungefär en femtedel av alla villaägare en robotgräsklippare [8]. Tekniken som de allra flesta robotgräsklippare idag använder sig av är att det grävs ner en begränsningkabel runt gräsmattan och roboten går mer eller mindre slumpmässigt inom det området. Det finns även varianter med GPS-navigering, den har historiskt inte varit mer exakt än att den kan se på vilken del av gräsmattan roboten har klippt under lång tid [9]. Detta gör att roboten åker och klipper gräs som redan är klippt vilket resulterar i ett ineffektivt arbete. Om även robotgräsklipparen ska tillämpas för tillverkandet av mönstret på planen kan detta inte ske slumpmässigt. Ska gräsklipparen även anpassa för målning av linjer måste robotgräsklipparen kunna veta var den befinner sig.

För detta syftet bör en lokal-GPS användas. Det finns exempel på där denna lösningen implementerats. Bland annat ett tidigare kandidatarbete [10], TurfTank [11]

som målar linjer på en fotbollsplan och även Husqvarna EPOS (Exakt Positioning Operating System) [12] där det används för att undvika begränsningskablar och lyckas utveckla algoritmer för att klippa gräsmattan enligt kundens önskemål.

Svenska fotbollsförbundet har med hjälp av Husqvarna gått ihop och gjort en besparingsanalys där de jämför vanliga maskiner för gräsklippning och Husqvarnas automowers [13]. Enligt analysen kan en förening med en fotbollsplan spara mellan 50-100 tusen kronor efter åtta år med en robot från Husqvarna med EPOS tekniken. Användingen av autonoma robotgräsklippare gör också att utsläppen minskar med 98% och det motsvarar hela 254 planterade träd. Företaget Turftank har skapat en robot för linjemålning av fotbollsplaner. De har gjort egna beräkningar och det tar för en person med deras robot ungefär 30 minuter att måla alla linjer på en fotbollsplan medan det tar 40 minuter för tre personer att måla alla linjer manuellt och nästan 10 liters färg besparas med turftank [14].

1.1 Syfte

Syftet med projektet är att vidareutveckla en befintlig autonom gräsklippare för att göra den mer användbar på fotbollsplaner. Målet är att gräsklipparen ska kunna klippa och kamma gräset i en förutbestämd rutt, samt att kunna måla linjer på fotbollsplanen.

1.2 Problem- och uppgiftbeskrivning

För att uppfylla syftet med projektet och att få gräsklippare att utföra önskade uppgifter kan projektet brytas ner i olika delar. I projektet finns det två gräsklippare att tillgå, en Husqvarna Automower 450x och en Husqvarna Automower 550 EPOS. Till en början behövs det tillgång till gräsklipparnas inre styre och sensorer som möjliggör styrning från externt håll. Därefter ska gräsklipparen kunna köras utefter bestämd rutt för att kunna kamma, klippa gräs och måla linjer. För att genomföra detta behöver robotens GPS-punkter bestämmas och optimeras för att kunna köra till rätt position och till reglering för att gräsklipparen ska kunna köra rakt. Utöver detta behöver praktiska lösningar för målning av linjer och kamning av gräs framtas. En uppgift som sköts på en fotbollsplan som dock inte kommer behandlas i detta projekt är vattning av gräs, då det inte anses kunna lösas på ett mobilt sätt. Nedan bryts uppgifterna ner mer i detalj.

Styra gräsklipparna

För att styra gräsklipparna behöver deras inre dator kontrolleras. Funktioner som hjulhastighet, dess GPS-punkt och sensorer behövs för att kunna utföra önskade funktioner. När det inre systemet kan kontrolleras behövs sen ett API som hanterar all data som med hjälp av funktioner ger robotgräsklipparna instruktioner.

Köra efter bestämd rutt

För att få robotarna att köra efter önskad rutt behöver robotarna köra utefter en ruttplanering. Denna ruttplanering ska kunna bestämmas av användaren beroende på hur kamningsmönstrer ska se ut, vart den ska klippa samt vilka linjer den ska måla.

Navigera i sin omgivning

För att gräsklipparna ska kunna köra efter önskad rutt behöver de kunna navigera i sin omgivning. Den GPS-punkt som hämtas från gräsklipparnas inre system är av varierande precision och behöver därmed om möjligt förbättras, eller modifieras med hjälp av filter. GPS-punkten används därutöver för reglering för att se till att gräsklipparna kör med låg felmarginal utefter önskad rutt.

Kamning och målning

Kamning av gräsmattor görs genom att dra partier av gräs åt olika håll, detta gör att ljuset reflekteras olika och skapar ett mönster för beskådaren. Någon konstruktion för kamning behövs därmed framtas och tillämpas på gräsklipparna.
Målning av linjer ska kunna göras automatiskt och en lösning för målning samt tillhörande konstruktion behövs därmed framtas. Därtill ska det bara målas linjer på önskat ställe, detta gör att gräsklipparen måste kunna börja och avsluta målning vid rätt position automatiskt med hjälp av dess position och annan mekanik.

1.3 Mål och delmål

Målet med projektet är att kunna utöka funktionalitet på en gräsklippare. Detta genom ta fram en konceptuell lösning som även kan måla linjer på en fotbollsplan och kamma gräset i mönster på ett automatiskt sätt. För att uppfylla målet delas uppgiften ner i en mängd delmål. Delmålen presenteras nedan och det första stycket med delmål behandlar mer basala mål för att gräsklipparen ska kunna utföra funktioner. Det andra stycket behandlar de delmål som får gräsklipparen att kunna utföra mer avancerade funktioner som uppfyller slutmålet med projektet.

Basala delmål:

- Gräsklipparen ska kunna fjärrstyras med hjälp av en dator.
- Gräsklipparen ska kunna köra till en given punkt med en felmarginal på 10 % i förhållande till längd på körsträcka.
- Gräsklipparen ska kunna kamma i raka mönster som är synliga för åskådaren.
- Gräsklipparen ska ha en funktion som målar linjer automatiskt.

Avancerade delmål:

- Gräsklipparen ska kunna navigera utefter geometrin på en hel fotbollsplan.
- Gräsklipparen ska kunna måla samtliga linjer på en fotbollsplan med rätt linjebredd och med en tolerans på 20 centimeter.
- Gräsklipparen ska kunna kamma mönster över hela fotbollsplanen utefter angivelse av användaren.

1.4 Avgränsningar och begränsningar

Målet med projektet är att tar fram ett koncept som kan sköta flera av de uppgifter som en planskötare gör idag. Med detta finns den en mängd avgränsningar och begränsningar som är värda att benämna. De två mest relevanta begränsningar med projektet är tid och budget. Tiden för projektet är fyra månader vilket leder till att vissa delar måste prioriteras och det med största sannolikhet kommer finnas möjliga vidarearbeten med projektet.

Budgeten för projektet är 5000 SEK. Budgeten kommer räcka till att köpa in framför allt de elektroniska komponenter som behövs för att styra och samla in relevant data. Pengar för till exempel en lösning för målning av linjer är därmed begränsad, vilket leder till att resultatet för exempelvis kvalitet på linjer kan bli lägre än önskat. Därmed kan lösningen för linjemålning och andra praktiska saker vara av mer demonstrativ natur än faktisk optimal lösning.

Förutom begränsningar sett till budget och tid finns det en mängd begränsningar med dem gräsklippare som finns att tillgå. Gräsklipparen av modell Husqvarna Automower 450x som finns att tillgå i CASE labbet arbetar utefter gamla mjukvaror som inte är kompatibla med många nya programvaror. Det finns även komponenter i gräsklipparna som inte är utbytbara, exempelvis batteri och servomotorer, vilket leder till att körtid och körhastighet är saker som är begränsade.

Övriga avgränsningar vad gäller projektet listas nedan:

- Inga moduler för hinderdetektering kommer framtas utan planen förutsätts vara hinderfri vid körning.
- Planen antas vara i körbart skick och inga störningar som lera eller hål kommer tas hänsyn till.
- Konceptet konstrueras inte för att klara av väderförhållande i form av kraftig vind, regn eller snö, detta i och med att en fotbollsplan sköts vid lugnare väderlag.
- Klippningen i sig kommer inte att testas då det redan är konstaterat att den kan klippa. bredd på klippdiametern kommer dock tas i hänsyn vid ruttplanering samt konstruktion av kam.
- En hel fotbollplan kommer inte kunna målas i ett stycke då det dels inte finns yta för att måla en hel fotbollsplan, och dels begränsad plats för färg för målning. Linjer kommer istället att testas att målas styckvis.
- Trots att bevattning av planen utgör är en del av planskötsel, kommer den inte att hanteras av robotgräsklipparna.

1.5 Samhälleliga- och etiska aspekter

I och med att typen av gräsklippare som ska tas fram har till uppgift att även måla och kamma fotbollsplaner kommer detta kunna bidra till att göra planskötseln

mer lättskötta. Det leder till att fler gräsplaner kan komma att utnyttjas istället för konstgräsplaner. År 2019 uppskattades det att det genomsnittliga mikroplastförluster från en elvamannaplan med konstgräs är cirka 534 kilogram per år [5]. I dagsläget i Sverige har detta inte visat några större miljö effekter, men i framtiden om mer konstgräsplaner anläggs kommer koncentrationen av mikrofibrer att bli påtaglig av miljön. Om det då istället skulle bli lättare att sköta gräsplaner skulle detta kunna ha en positiv inverkan på miljön då konstgräsplaner inte behövts i lika stor utsträckning som idag. Denna robotgräsklippares påverkan på miljön kommer även vara såpass liten att den är nästintill försumbar.

Ur ett ekonomiskt perspektiv skulle det kunna bli mer gynnsamt att införskaffa sig en robot för klubbar och kommuner som sköter fotbollsplaner autonomt. En konsekvens av detta skulle kunna vara att människor förlorar sina jobb inom branschen. Dock i och med att samhälle ständigt utvecklas kommer nya arbeten att tillkomma som gör att arbetslösheten nödvändigtviss inte behöver öka när vi går åt ett mer automatiserat samhälle.

Den autonoma gräsklipparen kommer innehålla någon form av GPS för att sköta lokaliseringen. Hur information från GPS:en sparas skulle kunna inkräkta på en köparens integritet då det skulle kunna användas för att få information om köparen. Det ska inte vara möjligt att fördelaktigt använda denna information för någons vinning. Ansvaret för att detta inte sker ligger på produktutvecklaren, men även distributören av produkten. En risk som skulle kunna inträffa är att det uppstår något typ av fel hos gräsklipparen under användning. Eftersom gräsklipparen inte ska behöva ha bevakning på sig medan den arbetar skulle det till exempel kunna leda till att den börjar köra utanför det tänka området. Den skulle då kunna komma att bli en fara för allmänheten. Det skulle därför fördelaktigt kunna framställas en lösning för att undvika detta problem som exempelvis notiser i en mobil applikation eller liknande.

Det finns en risk att djur och människor kan komma till skada om de kommer i kontakt med gräsklipparen. Idag är det främst igelkottar som kommer till skada av robotgräsklippare, men det finns även enstaka fall där även hundar och katter kommit till skada [15]. Att användning inte genomförs under natten skulle kunna vara en lösning till att minska risken för att igelkottar inte kommer till skada då det är då som dem är mest aktiva.

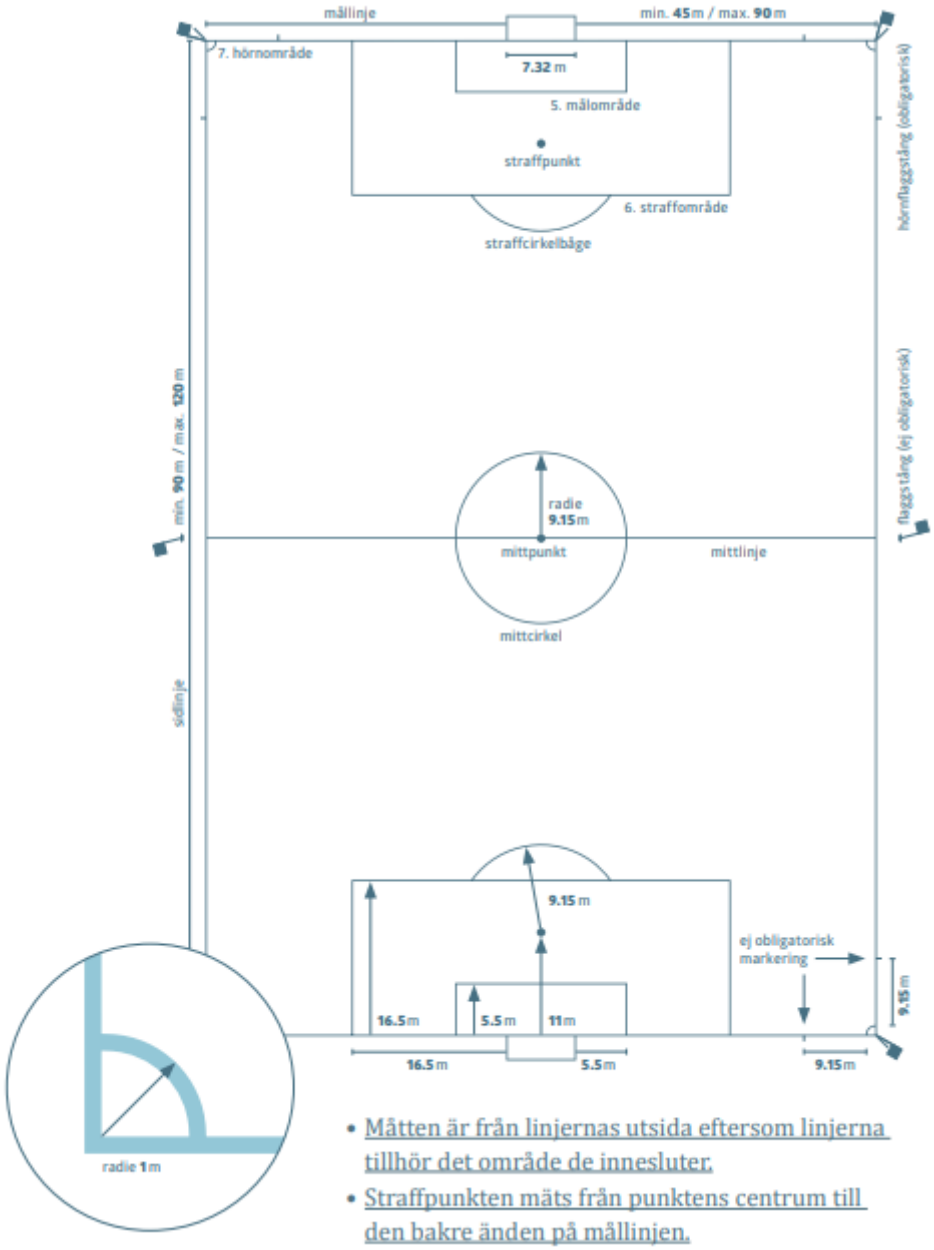
Hermansson och Hanssons trepartsmodell analyserar de etiska aspekterna och inte bara de kvantitativa. Modellen omfattar tre parter; beslutstagare, förmånstagare och risktagare. Sju centrala frågor tillkommer som avgör om beslutet är etiskt korrekt efter att risker och förmåner vägts mot varandra [16]. I situationen med en automatiserad gräsklippning, kamning och linjemålning innebär det mindre arbete för planskötaren, eventuellt arbetslöshet. Planskötaren blir då risktagare. Då planägaren är förmånstagare, beslutstagare men inte risktagare innebär det en etiskt kompromissbar situation.

2
Teori

För att en robotgräsklippare ska kunna utföra de uppgifter som finns rörande fotbollsplaner krävs modifieringar. Gräsklipparen måste kunna följa en rutt med exakt precision. När väl detta är uppnåt behöver robotgräsklipparen även kunna genomföra klippning av gräset, kamning för mönster och färgläggning av linjer. Detta avsnitt kommer att behandla de teorier som ligger till grund för detta projektet för att robotgräsklipparen ska kunna genomföra dessa uppgifter.

2.1 Fotbollsplan

Dimensionerna på fotbollsplaner i Sverige regleras av Svenska fotbollsförbundet och ska vara inom vissa intervall. Dimensionerna för fotbollsplanen ska vara inom vissa intervall; Långsidor ska vara mellan 90 och 120 meter, och kortsidan ska vara mellan 45 och 90 meter [17]. Övriga dimensioner går att läsa av i Figur 2.1. För en fotbollsplan som antas vara 105 meter lång och 64 meter bred blir den totala linjelängden 740,47 meter. Gräset är rekommenderat att ha en höjd på ungefär 30 millimeter vilket innebär att klippning sker veckovis.



Figur 2.1: Figuren visar svenska fotbollsförbundets regler för dimensioner för en elvamannaplan [17].

2.2 Husqvarna Automower

Husqvarna Automower är en serie med självkörande robotgräsklippare som är tillverkade av företaget Husqvarna Group [18]. Deras gräsklippare är utrustade med sensorer för att undvika hinder och kör autonomt runt gräsmattan och klipper gräs med hjälp av ett roterande blad på undersidan. Det finns olika versioner av robotgräsklippare med olika räckvidd, laddningstid och ljudnivå med mera. Hos Husqvarna sker navigeringen av robotgräsklipparna olika beroende på modellen. Det finns

till exempel gräsklippare som använder sig av GPS-stödd navigering och andra som använder sig av EPOS-teknologi (Exact Positioning Operating System) för mer exakt navigering.

2.2.1 Husqvarna Automower 450x

Husqvarna Automower 450x är en självkörande gräsklippare som är gjord för klippning av stora gräsmattor med en mer komplex terräng [19]. För att avgränsa ett område för klippning anläggs en begränsningskabel runt det specifika området. För klippning på ytor där det förekommer hinder och objekt har gräsklipparen utrustats med inbyggda ultraljudsensorer. Robotgräsklipparen har GPS-stödd navigering och detta ger möjlighet för gräsklipparen att köra genom passager och klippa på mer komplexa ytor. Robotgräsklipparen har en vikt på 13,9 kilogram och dess dimensioner är 72 x 56 x 31 centimeter (längd x bredd x höjd). En fulladdad gräsklipparen har en beräknad klipptid på 270 minuter och laddningstid för detta är 60 minuter. Gräsklipparen är även utrustad för att kunna klippa gräs i olika klipphöjder. Den minimala klipphöjden på gräset är 20 millimeter och den maximala klipphöjden är 60 millimeter och med detta har den en klippbredd på 24 centimeter. Nedan i figur 2.2 visas Husqvarna Automower 450x.

Figur 2.2: Figuren visar Husqvarna Automower 450x [19].

Husqvarna Automower 450x använder sig av första versionen av Husqvarna Research Platform (HRP) som benämns HRP1 som innehåller mjukvara och modeller som möjliggör styrning av gräsklipparen [20]. HRP är skapat av Husqvarna för att möjligöra vidare forskningsprojekt av robotgräsklipparna för allmänheten. HRP har därmed framställts på ett användarvänligt sätt som möjliggör enkel åtkomst till data från sensorer och styrning av motorer [21]. Denna mjukvara finns att tillgå via

ett Github-repo från Husqvarna Research och det är detta som då kallar på ROS noder från robotgräsklipparen.

2.2.2 Husqvarna Automower 550 EPOS

Husqvarna Automower 550 EPOS är en robotgräsklippare som är gjord för professionell användning [22]. Gräsklipparen använder sig av EPOS-vägledning som gör att en precision på 2-3 centimeter kan uppnås för vart den befinner sig och den kan även räkna ut bästa rutten till och från laddningsstationen. Gräsklipparen är inte i behov av en nedgrävd begränsningskabel som markerar området utan istället kan den använda sig av virtuella gränser som den förhåller sig till. Gräsklipparen har en vikt på 13,8 kilogram och dess dimensioner är 72 x 56 x 32 centimeter (längd x bredd x höjd). Klipptiden för denna modell är 210 minuter och laddningstiden för ett fullt batteri är 60 minuter. Klipphöjden på gräset kan justeras mellan 20 millimeter upp till 60 millimeter med en Klippbredd på 24 centimeter. I Figur 2.3 visas Husqvarna Automower 550 EPOS under en klippning på en fotbollsplan.

Figur 2.3: Figuren visar Husqvarna Automower 550 EPOS [22].

Likt Husqvarnas modell 450x använder sig modellen 550 EPOS av Husqvarna Reaschers Platform (HRP). Denna innehåller mjukvaran för tillgång till sensorer och styrning av motorerna. Skillnaden är dock att modellen 550 EPOS använder en nyare version av HRP (HRP2) som är tillämpad för ROS 2 men versionen finns inte tillgänglig för allmänheten än utan Husqvarna bidrog med plattformen för projektet. Detta då robotgräsklipparen är en nyare modell och har därmed andra system, men även att ROS 2 använder sig av andra ROS noder.

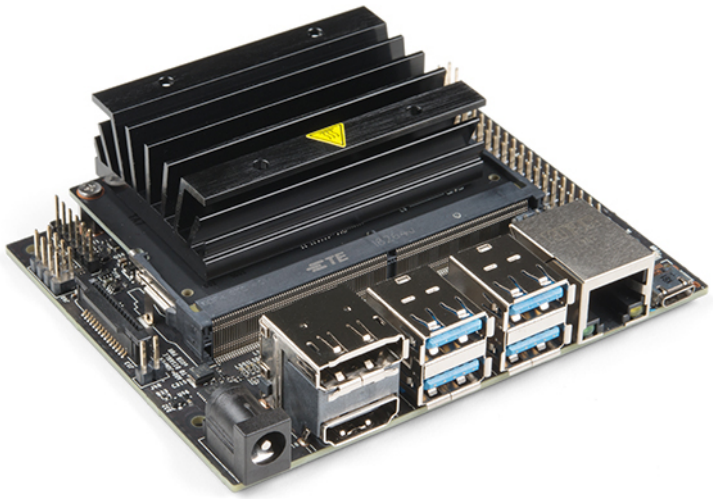
2.3 RTK teknik

Real time kinematic (RTK) är en teknik som används för att få bättre positionsmätning [23]. Positionen kan i verklig tid ge precision inom några få centimeter [24]. RTK använder sig av ett antal närliggande satelliter som skickar ut GNSS (Global

Navigation Satellite System) signaler mot en basstation eller referensstation med känd punkt och även till en rörlig GPS-mottagare. Den rörliga GPS-mottagaren kan beräkna sin exakta position genom att jämföra avståndet till referensstationens position och sin egen position som kommer från GPS-signalerna från GNSS systemet.

2.4 Jetson Nano

Jetson Nano är en styrdator som är speciellt utformad för att användas inom områdena AI och robotteknik [25]. Med Jetson Nano kan utvecklare och ingenjörer snabbt och enkelt utveckla och skapa prototyper av AI-applikationer, inklusive bildigenkänning, objektidentifiering, segmentering och talbehandling. En av fördelarna med Jetson Nano är dess kompakta formfaktor som gör den idealisk för att integreras i mindre robotar. Jetson Nanon har stöd för realtidsbearbetning av data, vilket ger stora möjligheter vid arbete av navigering. Nedan i Figur 2.4 visas en Jetson Nano.



Figur 2.4: Figuren visar en Jetson Nano [26].

2.5 Raspberry pi

Till en början skapades de första raspberry pi modellerna för undervisning av datavetenskap i skolor [27]. Dess popularitet blev större än förväntat och ledde till en pådriven utveckling. Bland de senaste modellerna finns Raspberry pi 4 som är en

liten och kraftfull enkortdator [28]. Den används idag till att bland annat driva surfdatorer, mediaspelare och servrar. Till skillnad från traditionella datorer har Raspberry pi ingen mekanisk hårddisk eller SSD-disk för lagring av operativsystem, program och filer. I stället använder den ett Micro-SD-kort. I Figur 2.5 visas en bild på en enkortdator av modell Raspberry pi 4.

Figur 2.5: Figuren visar en Raspberry pi 4 [29].

2.6 ROS

Robotoperativsystemet (ROS) är en uppsättning mjukvarubibliotek och verktyg som används för att bygga robotsystem och applikationer [30]. ROS håller på att bli standarden inom robotikprogrammering [31]. Det finns flera versioner av ROS varav två stycken är ROS 1 och ROS 2. ROS 1 Melodic använder Ubuntu 18.04 och det nyare ROS 2 Galactic använder Ubuntu 20.04. Ubuntu är ett användarvänligt Linux-baserat operativsystem och används exempelvis för att driva webbservrar, programmera eller molntjänster med mera.

ROS är uppbyggd på distribuerade processer, även kallade noder, vilket innebär att flera körbara filer kan köras samtidigt. Det är önskvärt då robotar ofta har olika processer som ska köras individuellt eller med koppling till varandra. Dessa noder har olika topics, som styr olika funktioner, till dess kan det läsas data ifrån eller skrivas data till [32].

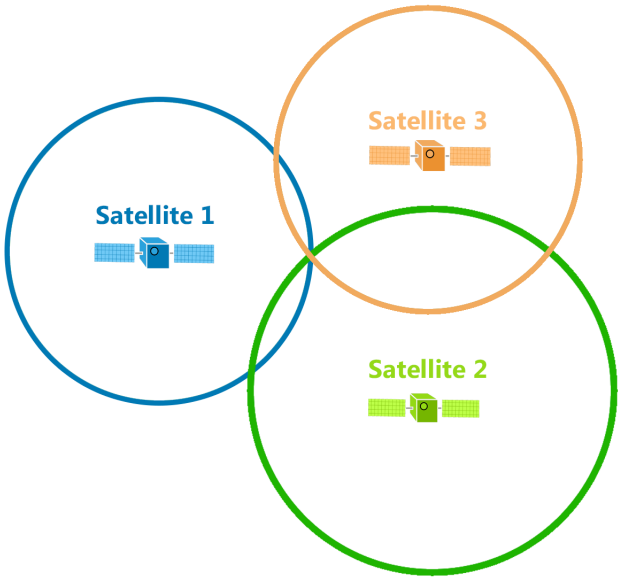
2.7 SSH

Secure Shell (SSH) är ett internetprotokoll som används för att upprätta en säker anslutning mellan datorer över internet eller i lokala nätverk oberoende av vilken nätverkstjänst som används. Eftersom SSH ger möjlighet att upprätta en fjärranslutning till en mikrodator används SSH ofta för att fjärrstyra obemannade farkoster [33]. SSH kan utföra kommandon och program som redan finns i farkosten. SSH läm-

par sig väl att använda i Linux operativsystem vilket innebär att det ofta fungerar bra för robotapplikationer som använder ROS som är byggt på Linux. SSH kan användas till både Jetson Nano och Raspberry pi 4.

2.8 Trilateration

GPS-mottagare använder sig av en mätningsmetod som kallas trilateration. Till skillnad från triangulering som mäter vinklar använder sig trilateration av avstånd [34]. En GPS-punkt på exempelvis jordens yta lokaliseras med hjälp av trilateration mellan en mängd satelliter. Satelliterna skickar ut en signal i alla riktningar som träffar GPS-mottagaren och en distans uppmätts. Distansen från varje enskild satellit gör dock inte GPS-punkten känd i med att den kan finnas längs hela randen av signalen. Används flera satelliter kan dock positionen fastställas genom att hitta brytpunkter av de olika satellitsignalers ränder. Figur 2.6 visar att i ett tvådimensionellt fall räcker det med tre satelliter för att få en unik brytpunkt där GPS-mottagaren befinner sig.



Figur 2.6: Figuren visar signal från satellit 1, 2 och 3 [35].

I verkligheten skickar satelliterna ut signaler i en sfär snarare än en cirkel. Dessutom finns det en felmarginal i avståndsberäkningen från satelliterna vilket gör att brytpunkten är mer svårdefinerad. Då får uppskattningar göras för att bestämma den ungefärliga GPS-punkten.

2.9 Kalmanfilter

Ett Kalmanfilter kan användas i många olika projekt där informationen om tillståndet är osäkert i ett dynamiskt system. Med hjälp av filtret kan en välgrundad approximation göras om vad systemet kommer att göra härnäst [36]. Kalmanfilter fungerar väl för system som ständigt förändras.

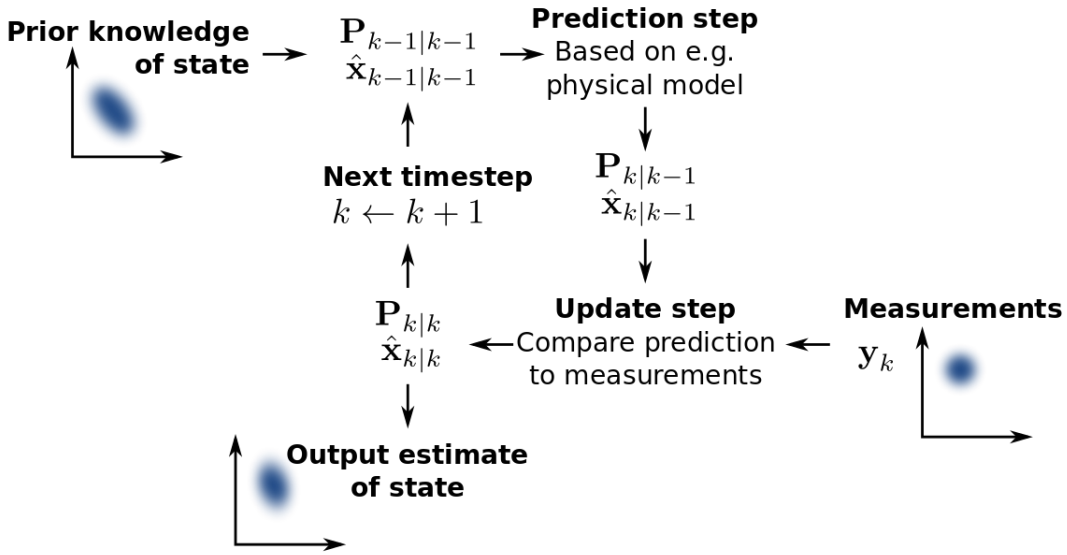
Första steget i Kalmanfilter algoritmen som visas i Figur 2.7 är förutspåsteget, som vidare i texten benämns ”prediction-steget”, där nästa steg förutspås med hjälp av föregående tillstånd och med ytterligare insignaler som påverkar tillståndet [37].

Ekvationen 2.1 beskriver hur prediction-steget fungerar där x är tillståndet som fås fram med hjälp av en enhetsmatrisen A och hur tillståndet såg ut vid föregående tidssteg. Till det föregående tillståndet adderas sedan hur insignalerna, som betecknas u, påverkar de olika variablerna i tillståndet x som betecknas med matrisen B. I detta steget räknas även matrisen P ut som visas i Ekvation 2.2. P är kovariansmatrisen för prediction-tillståndet och matrisen A är samma matris som innan som multipliceras med föregående kovariansmatris P och sedan adderas matrisen Q_{kalman} som är det uppskattade felet som kommer från in signalerna.

$$x_t = A \cdot x_{t-1} + B \cdot u_{t-1} \tag{2.1}$$
$$P_t = A \cdot P_{t-1} \cdot A^T + Q_{kalman} \tag{2.2}$$

Andra steget i ett Kalmanfilter är en korrektion av prediction-steget där filtret med hjälp av mätdata korrigerar hur rätt prediction-steget var [37]. I det här steget används olika typer av sensorer som avgör hur mycket fel föregående steg blev. I Ekvation 2.3 räknas först felet, y_t, ut mellan de uppmätta värdena, z_t, och de förutspådda prediction-värdena, x_t, som även multipliceras med en omvandlingsmatris H för att passa mätvärdena. Det läggs även på brus bestående av en vektor $\overrightarrow{w}$. Matrisen S som fås fram ur Ekvation 2.4 är den kvarvarande kovariansen från mätningen som bygger på enhetsmatrisen H, prediction-stegets kovariansmatris P och R_{Kalman} som är matrisen för kovariansen av bruset i mätningarna. Med hjälp av matriserna S, H och P beräknas sedan K som är Kalmanvärdet. Kalmanvärdet beskriver hur mycket värdena i prediction-steget ska korrigeras från resultatet av matningarna. Ett exempel är att om K är nära noll är mätningarna för dåliga och kommer därför inte påverka slutdatan, om K dock är nära värdet ett betyder det att mätningarna är bättre än prediction-värdena. Kalmanvärdet används i Ekvation 2.6 där de gamla värdena uppdateras till de mer korrekta värdena med hjälp av felet, y, och de tidigare prediction-värdena x. Till slut uppdateras även variansmatrisen för prediction-värdena med hjälp av Kalmanvärdet K, mätningsmatrisen H och föregående variansmatris för predict P. I Figur 2.7 illustreras samtliga steg i ett Kalmanfilter.

$$y_t = z_t - H_t \cdot x_t - w_t \tag{2.3}$$
$$S_t = H_t \cdot P_t \cdot H_t^T + R_{t,Kalman} \tag{2.4}$$
$$K_t = P_t \cdot H_t^T \cdot S_t^{-1} \tag{2.5}$$
$$x_t = x_t + K_t \cdot y_t \tag{2.6}$$
$$P_t = (I - K_t \cdot H_t)P_t \tag{2.7}$$

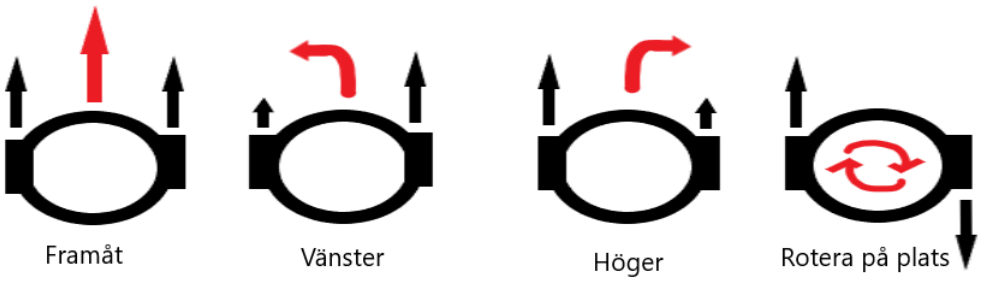


Figur 2.7: Figuren visar hur uppskattning av positionen i Kalmanfilter beräknas [38].

2.10 Differentiellstyrning

Gräsklipparna i detta projektet använder sig av en dynamik som kallas för differentiellstyrning [39]. Det innebär två hjul som är monterade på samma axel, men att hjulen styrs individuellt. Differentialstyrning är bland de vanligaste framdrivningssätten då det är okomplicerat att konstruera och styra roboten [40].

Då robotens hjul styrs oberoende kan det vara svårt att upprätta en rak körning. Då krävs reglering med hjälp av återkoppling som anpassar motorernas varvtal genom att variera spänning över motorerna. I Figur 2.8 visar bilden olika riktningar som en robot åker beroende på hjulens hastighet och riktning av rotation. Om höger hjul går snabbare än vänster svänger roboten vänster och vice versa. Om hjulen roterar olika riktningar i samma hastighet kommer roboten rotera kring mittpunkten av hjulen.



Figur 2.8: Figuren visar en differentiellstyrning av en robot som kör olika riktningar beroende på hjulens rotation [41].

2.11 Regulatorer

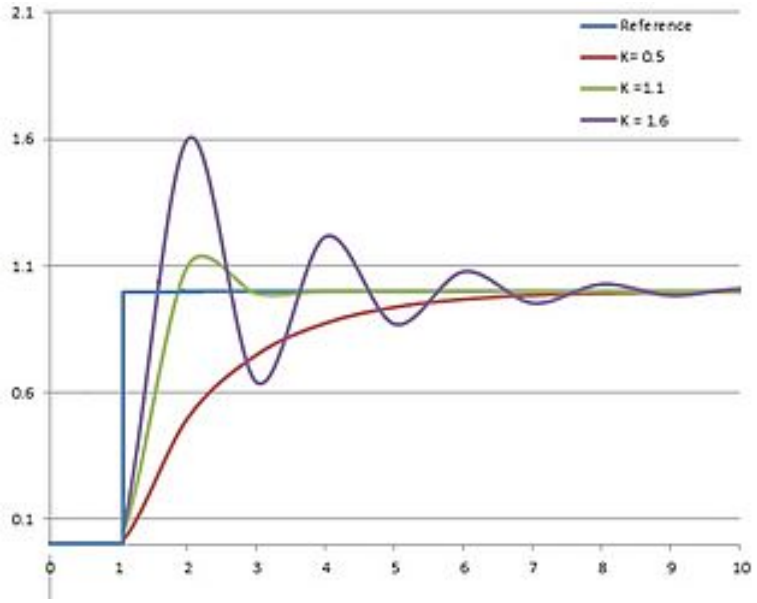
En regulator är en komponent eller ett system som används för att stabilisera och kontrollera en process eller ett system. En regulator tar en insignal exempelvis hastighet och använder den för att generera en utsignal. Utsignalen är avsedd att styra processen eller systemet för att uppnå önskat resultat. Återkoppling är en viktig del av regleringssystemet eftersom det möjliggör kontroll över processen eller systemet. Återkoppling innebär att utsignalvärdet skickas tillbaka till regulatorn för att justera insignalen i realtid för att hålla processen eller systemet stabilt och på rätt spår.

2.11.1 PID-regulator

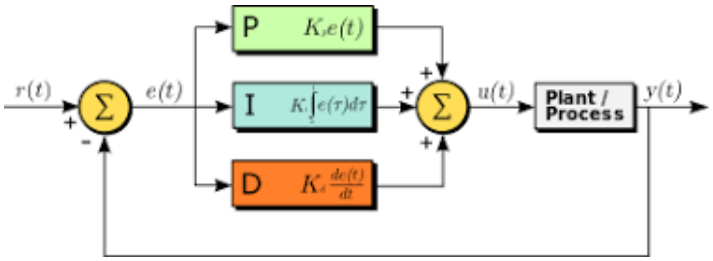
PID-regulator är en generisk benämning på en typ av regulatorer där en linjär kombination av proportionell, integrerande och deriverande verkan av ett reglerfel används för beräkning av en styrsignal [42]. För att implementera en PID-regulator krävs parametrar för proportionell, integrerande och deriverande verkan. Formeln för den ideala PID-regulatorn är:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\mathrm{d}\tau + K_d \frac{\mathrm{d}e(t)}{\mathrm{dt}} + u_0, \tag{2.8}$$

där u(t) är regulatorns utsignal och $e(t) = r(t) - y(t)$ är skillnaden mellan referensvärdet r(t) och mätvärdet y(t), vilket är reglerfelet. Regulatorns justerbara parametrar är, förutom styrsignalens basvärde u_0 (ofta är $u_0 = 0$), proportionella förstärkningen K_p, den integrerande förstärkningen K_i och den deriverande förstärkningen K_d [42]. I Figur 2.9 visas hur referensvärdet följs beroende på olika parametrar. I Figur 2.10 visas ett blockschema över en PID-regulator i en återkopplingsslinga.



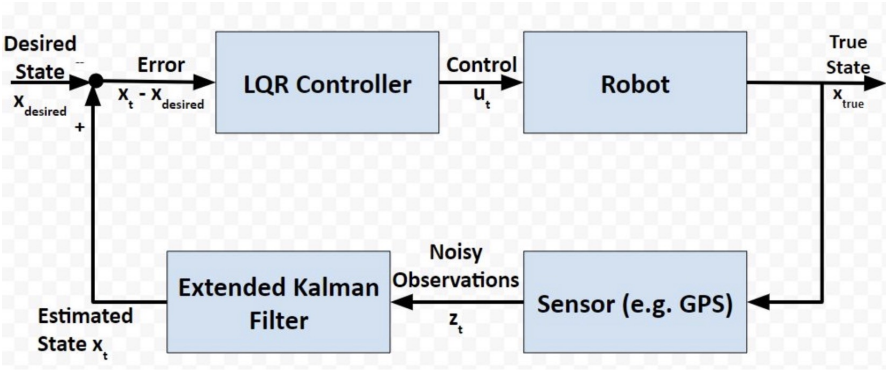
Figur 2.9: Figuren visar en graf över hur referensvärdet följs beroende på parametrarna K_p, K_i och K_d [43].



Figur 2.10: Figuren visar ett blockschema över en PID-regulator i en återkopplingsslinga. r(t) är det önskade processvärdet eller börvärdet (SP), och y(t) är det uppmätta processvärdet (PV) [43].

2.11.2 LQR

Linear Quadratic Regulator (LQR) är en regulator som löser ett optimeringsproblem för att beräkna vilka utsignaler som ska användas [44]. LQR använder tekniken ”optimal feedback control” som är en algoritm som skapar utsignaler som minskar felet mellan verkliga tillstånd och önskade tillstånd. Det objekt som använder regulatorn behöver sensorer som skickar tillbaka mätningar till systemet. Regulatorn jämför sedan informationen från sensorerna och önskade värden för att sedan skicka de optimala utsignalerna för att minska felet. I Figur 2.11 nedan visas hur LQR fungerar.



Figur 2.11: Figuren visar hur LQR fungerar [45]. (Återgiven med tillstånd)

Ekvation 2.9 visar ett optimeringsproblem som ska lösas där minimum av J ska tas fram. J är en kostnadsfunktion som bygger på tillståndet x och rörelse insignaler u [46]. Q och R är kostnadsmatriser för ett tillstånd x respektive insignaler u [44]. Olika Q och R sätts för att straffa olika värden på antingen x eller u. Ett exempel på det är att om Q innehåller stora siffror i diagonalen som är större än R matrisen straffas fel i tillståndet mycket hårt och därmed är det viktigare att tillståndet är sant än att insignalerna är låga. Ekvationen 2.10 beskriver hur nästa tillstånd ser ut där x är tillståndet som fås fram med hjälp av en enhetsmatrisen A och hur tillståndet såg ut vid föregående tidssteg. Till det föregående tillståndet adderas sedan hur insignalerna, u_t, påverkar variablerna i tillståndet x som betecknas med matrisen B. I LQR-regulatorn räknas felet på tillståndet, x_{fel}, ut vilket gör det

möjligt att få felet att närma sig noll.

$$J = \int_0^\infty x_t^T Q\, x_t + u_t^T R\, u_t\, dt \tag{2.9}$$

$$x_t = A\, x_{t-1} + B\, u_{t-1} \tag{2.10}$$

$$x_{fel} = x_t - x_{önskad} \tag{2.11}$$

$$A^T S + SA - SBR^{-1}B^T S + Q = 0 \tag{2.12}$$

$$K = R^{-1}B^T S \tag{2.13}$$

$$u_t = -Kx_{fel} \tag{2.14}$$

För att lösa optimeringsproblemet måste S lösas ur Ekvation 2.12 som kallas Algebraic Ricati Equation [46]. Med hjälp av S kan sedan optimala responsvärdet K som är ett viktigt för att få optimala insignaler till regulatorn när K multipliceras med felet i staten x_{fel} i Ekvation 2.14.

2.12 Pulsbreddsmodulering (PWM)

Pulsbreddsmodulering är ett sätt att kontrollera analoga enheter med digitala signaler. Digitala signaler är signaler som antingen kan representeras av noll eller ett, medan analoga signaler kan ta andra värden. Pulsbreddmodulering fungerar genom att den snabbt växlar spänning mellan två värden. Den mottagande enheten är av inducerande karaktär och hinner inte växla mellan dess maximum och minimum som den digitala enheten, utan antar ett värde mellan dem. De tre viktigaste parametrarna för PWM som påverkar utfallet är cykeltid, tillslagstid och frekvens [47]. Cykeltiden utgörs av hur lång tid det tar för en PWM-signal att fullborda en cykel, alltså hur lång tid det tar för en cykel att gå från på till av och till på igen. Tillslagstiden representerar i procentandel hur länge en signal är på över en period. Som exempel, om en digital signal har tio volt som maximum och tillslagstiden är 50 % kommer utfallet vara en medelspännings på fem volt. Tillslagstiden beräknas enligt:

$$T = \frac{T_{on}}{Cykeltid} * 100, \tag{2.15}$$

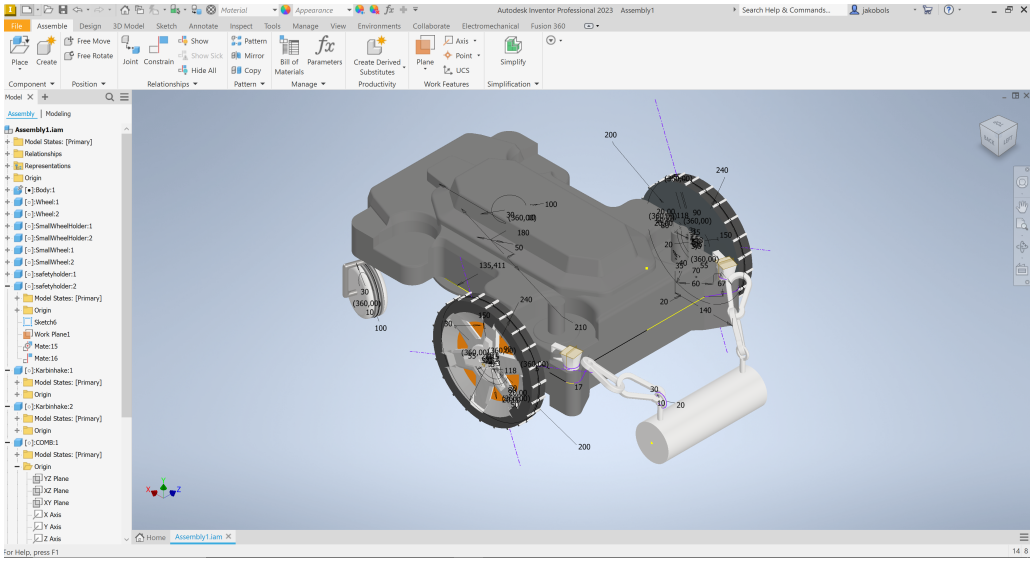
där T_{on} = Tiden som signalen är på.
Medelspänningen kan då beräknas enligt:

$$V_{avg} = \frac{D}{100} * V_{max}. \tag{2.16}$$

Frekvensen är kopplad till cykeltiden och är hur många gånger cykeln upprepas under en sekund. Frekvensen varierar beroende på användningsområdet. Ett viktigt användningsområde för PWM är att kontrollera hastigheten för likströmsmotorer samt servomotorer.

2.13 Computer Aided Design (CAD)

CAD (Computer-Aided Design) är en teknik som används inom många olika branscher såsom arkitektur, maskinteknik, elektroteknik med mera. Tekniken möjliggör tillverkningen av digitala representationer i två samt tre dimensioner. Program såsom CATIA, Inventor och Solidworks är alla väletablerade CAD-program som används inom industrin. CAD har revolutionerat designbranchen genom att öka precision, hastighet och effektivitet. I Figuren 2.12 visas hur CAD kan användas för att skapa 3D-modell.



Figur 2.12: Figuren visar CAD modell av en robotgräsklippare.

3
Metod

Arbetet fokuserade på att lösa uppgifterna som behöver skötas på fotbollsplaner vars underlag är av naturligt gräs. Detta inkluderar klippning, måla linjer, klippa effektivt och kamma gräset till mönster. För att kunna lösa dessa uppgifter krävdes en mängd åtgärder. Projektets början utgick från att kunna kommunicera med gräsklipparna. Vid projektets start fanns en Husqvarna Automower 450x att tillgå, men efter ungefär halva projektets gång donerades en Automower 550 EPOS från Husqvarna och metoden att få igång dessa skiljdes åt i några led. Där det skiljer sig beskrivs arbetsmetodiken för dessa separat.

Vidare byggdes en styrningsenhet baserad på programmeringskod för att kunna utföra önskade uppgifter såsom ruttplanering och upphämtning av gräsklipparens GPS-punkt. GPS-punkten användes för att kunna få gräsklipparen att åka utefter önskat mönster, men i med att alla GPS-signaler har en felmarginal av varierande storlek behövdes något slags filter som hanterar detta. Dessutom var reglering tvunget att tillämpas för att kunna få gräsklipparen att åka utefter önskat mönster.

Utöver de teoretiska delarna av projektet behövdes vissa konstruktioner framtas. Till en början behövdes en hållare för färgsprutan och något som ser till att den kan spruta automatiskt och som reglerar bredden på linjerna. Dessutom tillverkades det en kam och hållare för denna. I detta kapitel beskrivs det i närmre detalj hur arbetet gått till för de olika delarna och vilka avvägningar som gjorts för att få ut ett bra slutresultat.

3.1 Styrdator

För att kunna kommunicera och styra robotgräsklipparna krävdes en styrdator som innehöll den nödvändiga informationen. Det två olika gräsklipparna var kompatibla med olika ROS-system och gjorde därmed att olika styrdatorer valdes att tillämpas.

3.1.1 Husqvarna Automower 450x

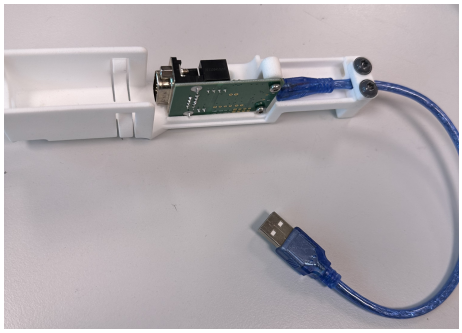
När valet av dator till gräsklipparen skulle göras fanns det redan en Raspberry pi 4 inkopplad i Husqvarnas Automower 450x vilket gjorde att denna valdes till en början. Efter genomförande av tester med installationer, läst litteratur och fört dialog med personer som var insatta i gräsklipparen blev slutsatsen att Raspberry pi 4 inte kunde användas. Den var inte kompatibel med det operativsystemet som behövdes

användas. Alternativet var då att använda en Raspberry pi 3 som var kompatibel med dessa. Problemet med denna var dock att den inte var tillräckligt snabb eller kraftfull för att utföra körning och navigering av gräsklipparen. Valet föll därför istället på ett Jetson Nano 2 GB Developer Kit. Att valet föll på en Jetson Nano var att den var mycket kraftfull och speciellt användningsbar för arbete med robotik och autonoma fordon. Året 2016 markerade lanseringen av Husqvarna Automower 450x och i och med detta används inte ROS 2 utan istället ROS Melodic. För att kunna använda ROS Melodic på robotgräsklipparen var det därför tvunget att installera Ubuntu 18.04 som operativsystem. Nyare versioner av Ubuntu kunde inte användas då ROS Melodic inte var kompatibelt för dessa. Operativsystemet installerades på ett SD-kort 32 Gb som sedan sattes in i Jetson Nanon. Efter detta installerades Ros Melodic på Ubuntu 18.04 och även ett HRP(Husqvarna Research platform). HRP installerades för att kunna använda alla sensorer och data som finns att tillgå i gräsklipparen. HRP innehåller all programmeringskod som Husqvarna använder till deras gräsklippare. Detta installerades via ett GitHub-repo från Husqvarna Research. Genom att köra filerna i HRP kunde därmed gräsklipparen att styras.

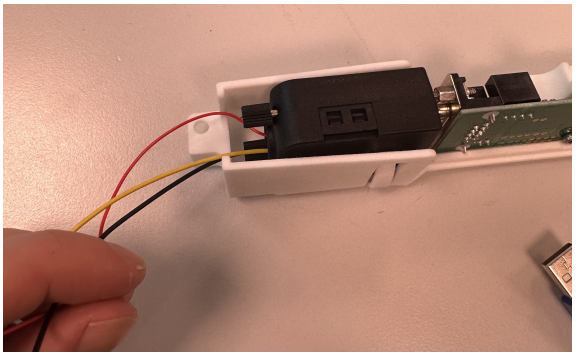
För att kunna på ett smidigt sätt kunna arbeta med Jetson Nanon installerades SSH. Detta gav då möjligheten att ansluta till gräsklipparen via fjärrstyrning och då styra Automower 450x trådlöst via en nätverksuppkoppling. För att på ett smidigt sätt hantera kodningen valdes SSH att kopplas samman med Jetson Nano via Visual Studio Code.

3.1.2 Husqvarna Automower 550 EPOS

Gräsklipparen av modell 550 EPOS som donerats av Husqvarna är en gammal testgräsklippare vilket betydde att ett arbete behövde utföras innan den var i skick för att kunna styras med Raspberry pi 4 som styrdator. Arbetet som behövdes göras innan det gick att få kontakt med roboten via en Raspberry pi visas i Figur 3.1(a) och 3.1(b). Figurerna visar hur en CAN-adapter kopplas in för att kunna kopplas vidare till en Raspberry pi och in till Husqvarnas powerkort som visas i Figur 3.2(a).



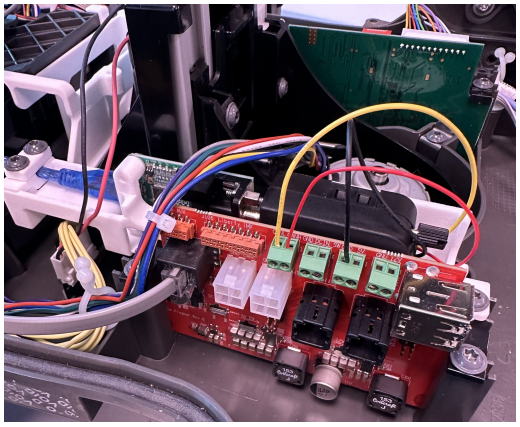
((a))



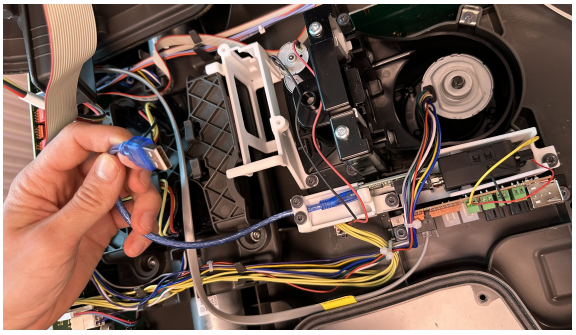
((b))

Figur 3.1: Figuren visar hur kopplingen av CAN-adapter ser ut.

Husqvarnas powerkort som visas i Figur 3.2(a) används för att dela upp energin från batterierna till Raspberry pi:n och informationen som ska in i gräsklipparen från Raspberry pi:n. En CAN sladd används för att överföra energin och informationen till powerkortet från gräsklipparen. Från powerkortet kopplas dem röda och svarta sladdarna för att överföra informationen till CAN-adapter. Figur 3.2(b) visar översiktlig koppling utan Raspberry pi:n.



((a)) Figuren visar Husqvarnas powerkort.



((b)) Figuren visar översiktligt hur kopplingen ser ut utan Raspbery pi:n inkopplad. Raspberry pi:n ska sitta i den vita 3D-printade hållaren.

Figur 3.2: Figuren visar kopplingar mellan ett powerkort och en Raspberry pi.

När Raspberry pi:n var inkopplad kunde installationen påbörjas. Dokumentation om installering erhölls från Husqvarna. Dokumentationen innehöll instruktioner som följdes. På Raspberry pi:n skulle ROS 2 Galactic installeras och för att den skulle fungera krävdes Ubuntu 20.04 som operativsystem. Därefter kopplades VSC samman till Raspberry pi:n via SSH. Nästa steg var att ladda över Husqvarnas HRP2 till Raspberry pi:n. Dessa filer dekomprimerades och sedan överfördes till Raspberry pi:n via SSH. Dock uppstod ett problem då filerna till en början dekomprimerades i Windows istället för i Ubuntu, vilket resulterade i förlust av viktig information från filerna och därmed funktionsfel. Efter ett möte med Husqvarna kunde problemet identifieras och åtgärdas. Efter korrekt installation av Husqvarna HRP2 kunde filerna nu köras på gräsklipparen och användas för att styra den.

3.2 Beräkning av position

För att förbättra precisionen av robotgräsklipparen 450x för att få den att kunna följa en önskad rutt behövdes dess position bestämmas noggrant. Detta gjordes främst för 450x eftersom den fungerade bättre. Detta skulle även kunna appliceras på 550 EPOS med små förändringar. Detta för att utifrån den kunna följa de rutter som krävs för målning av linjer och kamning. För att kunna beräkna den befintliga positionen av gräsklipparen användes inbyggda sensorer tillsammans med GPS:en.

Den väsentliga datan som fås från robotgräsklipparen är odometri, som ger x- och y-position, samt GPS-positionen. Odometri utgår från en startpunkt och uppskattar positionen utifrån hur däcken roterar och tar inte i beaktning hurvida gräsklipparen faktiskt flyttar sig den sträckan i verkligheten. X- och y-positioner från odometrin var det första som nyttjades och testades för uppskatta en precision hos hjulen. När robotgräsklipparen endast körde utefter odometrin blev felet litet på kort avstånd men mycket stort på långa avstånd. Precisionen på x- och y-positionerna gav på en 2 meters sträcka ett fel på ungefär 5 centimeter. Däremot divergerar detta felet kraftigt över tid vilket gjorde att på 60 meter var felet 14 meter. Genom att då använda odometri tillsammans med GPS erhölls en bättre precision.

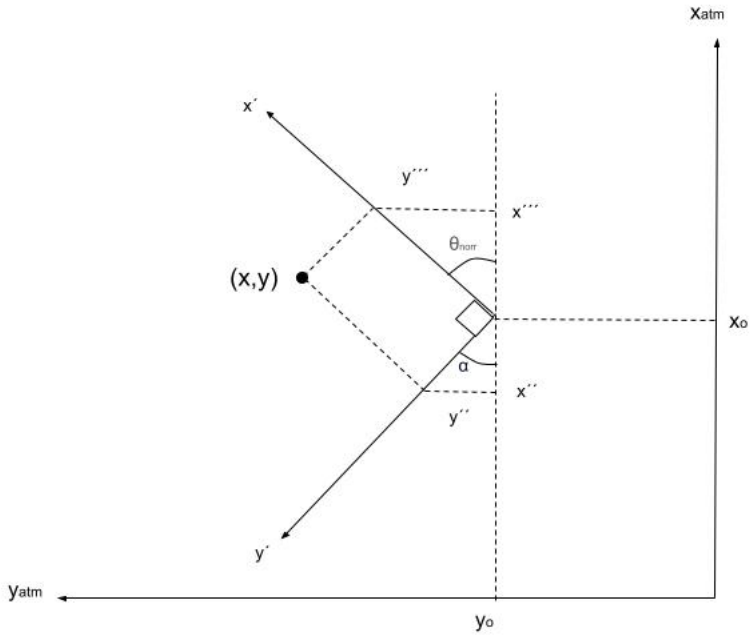
3.2.1 Koordinatsystem

För att beskriva en målpunkt behövdes ett enkelt koordinatsystem. Detta gjordes genom att x-axeln var framåt på gräsklipparen och y-axeln var till vänster, räknat från gräsklipparens ursprungsposition. Positionen (5,10) är därför fem meter fram och tio meter åt vänster. För att gräsklipparen skulle använda detta koordinatsystem gjordes detta koordinatsystem om till gräsklipparens egna koordinatsystem. Origo placerades automatiskt där gräsklipparen startade upp första gången och ändrades därför inte mellan tester. När gräsklipparen sattes igång några meter ifrån gräsklipparens första position skildes därav dessa koordinatsystem. Därför gjordes koordinattransformationen nedan för att beskriva målkoordinaten i gräsklipparens koordinatsystem:

$$\begin{bmatrix} x_{\text{goal\_automower}} \\ y_{\text{goal\_automower}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_{\text{init}}) & -\sin(\theta_{\text{init}}) & x_{\text{start\_automower}} \\ \sin(\theta_{\text{init}}) & \cos(\theta_{\text{init}}) & y_{\text{start\_automower}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{goal\_prim}} \\ y_{\text{goal\_prim}} \\ 1 \end{bmatrix}, \quad (3.1)$$

där θ_{init} är ursprungsvinkeln från gräsklipparen, $x_{\text{goal\_automower}}$ samt $x_{\text{start\_automower}}$ är koordinater enligt gräsklipparen själv och x_{goal_prim} är x-målet i det enkla koordinatsystemet där origo är där gräsklipparen startar. Vinkeln från robotgräsklipparen erhölls i Kvaternioner. Kvaternioner är ett matematiskt verktyg som används ofta inom 3D-rotationer och orientering. Från gräsklipparen erhölls komponenter, *z_dir* och *w_dir*, givna i kvaternioner. Sedan användes funktionen *tf.transformations.euler_from_quaternion()* vilken är en funktion som omvandlar kvaternioner till Euler-vinklar, vilket ger en mer intuitiv förståelse av objektets orientering. Sedan extraherades yaw-vinkeln vilket är roteringen runt Z-axeln. För att kunna kombinera odometri med GPS behöver dem beskriva positionen i samma koordinatsystem. Från 450x erhölls latitud och longitud från gräsklipparen. Detta gjordes om till ett kartesiskt koordinatsystem genom ”Easting and northing” koordinatsystemet [48]. Det ger ett koordinatsystem där y-axeln beskriver norr och x-axeln beskriver öst. Däremot gjordes detta om till att x-axeln istället beskrev norr och y-axeln beskrev väst.
Vinkeln mot norr som erhölls från IMU var opålitlig. Vid fallet att den hade varit tillförlitlig hade istället en invers transformation till Ekvation 3.1 utförts. Därefter skulle gräsklipparens koordinatsystem kunna transformeras till ”Easting and

northing”-koordinatsystemet. Detta gjordes genom att rita upp koordinatsystemen vilket visas i Figur 3.3 och transformationen beräknas genom transformationen i Ekvation 3.2.



Figur 3.3: Med hjälp av denna figur kunde koordinattransformationen beräknas vilket visas i Ekvation 3.2.

$$\begin{bmatrix} x_{\text{utm}} \\ y_{\text{utm}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_{\text{north}}) & -\sin(\theta_{\text{north}}) & x_{\text{start}} \\ \sin(\theta_{\text{north}}) & \cos(\theta_{\text{north}}) & y_{\text{start}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{automower}} \\ y_{\text{automower}} \\ 1 \end{bmatrix} \tag{3.2}$$

Utm beskriver koordinaterna i ”Easting and northing” koordinatsystemet, θ_{north} är vinkeln mot norr i matematiskt positiv rikting och x_{start} och y_{start} är den första GPS-punkten i ”Easting and northing” koordinatsystemet. Eftersom θ_{north} inte erhölls gjordes istället endast en translation av koordinaterna i Automower-koordinatsystemet till ”Easting and northing”-koordinatsystemet vilket är ekvivalent med att sätta θ_{north} till noll. För att beräkna vinkeln mellan odometrin och GPS i Figur 3.4 åkte gräsklipparen långsamt framåt med hjälp av odometrin och samlade in data för tio GPS-punkter.

Därefter anpassades en linje efter dessa punkter på formen $y = kx + m$:

$$k = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}, \tag{3.3}$$

$$m = \frac{\sum_{i=1}^{n} y_i - k\sum_{i=1}^{n} x_i}{n}, \tag{3.4}$$

där n är antalet punkter. För att beräkna vinkeln mellan linjerna för GPS och odometri användes Ekvation 3.5.

$$\theta = \arctan\left(\frac{k_2 - k_1}{1 + k_1 \cdot k_2}\right) \tag{3.5}$$

Ytterliggare återgärder togs även för att försäkra om att detta var rätt vinkel, eftersom fyra vinklar kan erhållas mellan två linjer. Detta beskrivs i Python-koden, fil coord_sys_trans.py, se Bilaga A.4.

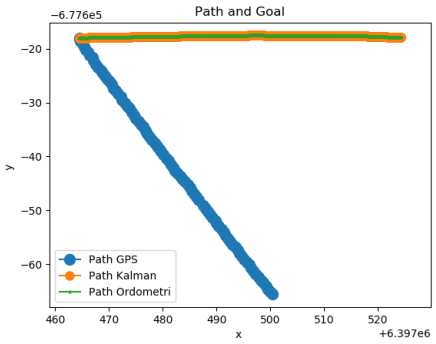
Genom att jämföra denna linjes vinkel med odometrins linje kan GPS-datan roteras till odometrin. Detta gjordes genom att först förskjuta linjen till origo, sedan rotera den och sedan translatera tillbaka den till dess ursprungsposition.

$$x_{\text{translated}} = x - x_0 \tag{3.6}$$
$$y_{\text{translated}} = y - y_0 \tag{3.7}$$

$$\begin{bmatrix} x_{\text{rotated}} \\ y_{\text{rotated}} \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x_{\text{translated}} \\ y_{\text{translated}} \end{bmatrix} \tag{3.8}$$

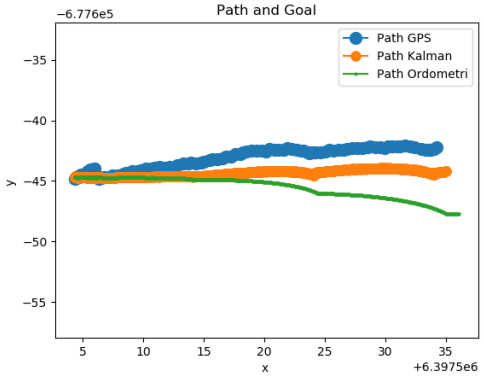
$$x_{\text{new}} = x_{\text{rotated}} + x_0 \tag{3.9}$$
$$y_{\text{new}} = y_{\text{rotated}} + y_0 \tag{3.10}$$

Resultatet från att köra lite fram och beräkna vinkeln genom det visas i Figur 3.5(a). Däremot gav detta klar avvikelse från det önskvärda resultatet och därför kalibrerades gräsklipparen genom att bära den 60 meter rakt fram. Genom att beräkna vinkeln för denna linje enligt GPS jämfört med odometrin och upprepa kalibreringen tre gånger, kunde ett medelvärde erhållas och vinkeln mellan odometrin och GPS beräknas vilket visas i Figur 3.4

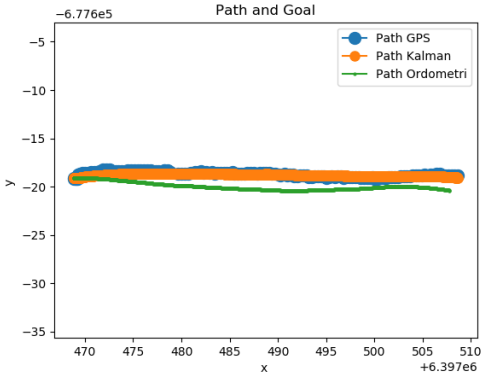


Figur 3.4: Figuren visar kalibrering av GPS. Genom detta steg kan vinkeln beräknas mellan odometri och GPS. I nästkommande verkliga försök kan nu alla GPS-punkter roteras in i samma koordinatsystem som odometrin vilket möjliggör användning av Kalmanfiltret.

Detta gav en bättre vinkeln än tidigare vilket visas i Figur 3.5(b).



((a)) Figuren visar en 30 meters körning genom att köra långsamt i tio sekunder och sedan räta upp GPS till odometri. Detta är det ena sättet att kalibrera klipparen så att GPS och odometri befinner sig i samma koordinatsystem. I verkligheten roterades GPS för lite och därför gick gräsklipparen snett i detta test.



((b)) Figuren visar en 50 meters körning genom att kalibrera genom att först bära gräsklipparen 60 meter flera gångar och därav räkna ut vinkeln mellan GPS och odometri.

Figur 3.5: Figurerna visar två olika sätt att rotera GPS till odometri. Den vänstra figuren visar kalibrering genom att köra några meter och sedan beräkna vinkeln mellan GPS och odometri. Den högra figuren visar detta genom att först gå 60 meter med gräsklipparen och därav beräkna vinkeln.

Däremot varierade den fortfarande lite mellan olika gånger vilket gjorde att mätningarna ibland fick varierande resultat. Kalibreringen behövdes göras om då vinkeln

varierade över tid. Därav behövde kalibrering ske inom några timmar innan en körning. Detta hade kunnat undvikas med en basstation. Genom att rotera GPS till odometri erhölls positionsdatan i samma koordinatsystem och möjliggjorde därför för ett Kalmanfilter.

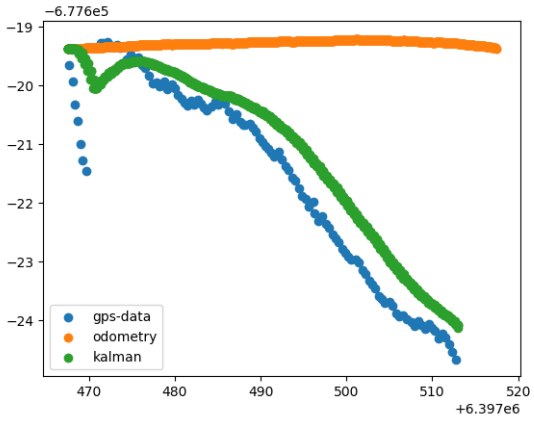
3.2.2 Kalmanfilter

För att få gräsklipparen att värdera till vilken utsträckning den ska lita på GPS-signalerna användes ett Kalmanfilter. Genom användning av detta filter möjliggjordes valet mellan två olika x- och y-positioner baserat på deras respektive trovärdighet, vilket medgav användning av båda positionerna för att identifiera och lokalisera ett medelläge. Detta bidrog till att förbättra noggrannheten och tillförlitligheten hos positioneringsalgoritmer. Från GPS-datan tillkom en kovariansmatris som är ett mått på GPS-datans säkerhet i en given punkt. Med användning av kovariansmatrisen kunde Kalmanfiltret nu utnyttja informationen för att bestämma en viktning av mätdata vid varje mätögonblick. Tillståndsmatrisen x som används i Kalmanfiltret valdes till att innehålla variablerna x, y och θ för att roboten är rör sig i ett plan med två dimensioner. Oftast brukar hastighet och vinkelhastighet användas för att beräkna nästa position, men då robotten ger x- och y-positioner tio gånger per sekund användes istället skillnaderna på värden av x och y för att beräkna nästa position enligt Ekvation 3.11 som kan jämföras med Ekvation 2.1. Skillnaderna i värdena på x och y räknades ut genom att jämföra värdena innan på x och y och dem nya värdena från odometrin. Ekvationen 3.12 beräknar skillnaden i vinkeln $\Delta\theta$. I Ekvation 2.2 i matrisen P började det med att testa några värden i diagonalen till exempel värdet 0,1 och matrisen Q valdes till några rimliga värden för felet på odometridatan. Data för odometrin och GPS-data samlades in från gräsklipparen av modell 450x. Från gräsklipparen användes GPS-kovariansens x- och y-värden som multiplicerades med 0,001. Vinkelns kovarians multiplicerades med 0,002 och kovariansen för vinkel och positionerna lades in i matris R för att kunna jämföra hur mycket Kalmanfiltret ska lita på odometrin respektive GPS-datan.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{3.11}$$

$$\Delta\theta = \arctan(\frac{\Delta y}{\Delta x}) \tag{3.12}$$

Figuren 3.6 visar första testet av den plottade datan från Kalmanfilteret när den första insamlade datan från odometrin och GPS-datan används. Kalmanfiltret är i detta fallet konstruerat för att följa GPS datan mer än odometrin. Kalmanfiltret är applicerat efter testet var klart. Under testet användes enbart odometri.



Figur 3.6: Figuren visar första testet av Kalmanfiltret där Kalmanfiltret är applicerat efter testet var klart. Under testet användes endast odometridata.

3.2.3 Husqvarna Automower 550 EPOS

När kommunikation till gräsklipparen etablerats som beskrivs i avsnitt 3.1.2 testkördes python filen, medföljande med HRP2, hqv_keyboard_remote_drive. Detta gjorde att tangenterna på datorn som var kopplad till gräsklipparen kunde styra gräsklipparen framåt, bakåt och rotera den. Vidare observerades huruvida det gick att få hastigheten, vinkeln och GPS-positionen från gräsklipparen. Nästa steg i processen för gräsklipparen var att få kontakt med en referensstation som även den skänktes av Husqvarna som visas i Figur 3.7. Referensstationen behövde placeras på ett skaft utomhus för att få bra signal till GPS-sateliter. När referensstationen hade fått kontakt med satelliterna kunde den kopplas till gräsklipparen via appen som gräsklipparen är kopplad till. Därefter testades att få RTK-postionen som då är relativt referenstationen och positionen gavs med avståndet från basstationen i norr och öster. Detta printades ut samt mätosäkerheterna för dessa positioner. Nästa steg var att kunna köra gräsklipparen via en mainfil som tog in en position. Denna fil skulle då kunna kommunicera med robotgräsklipparen för att få gräsklipparen att köra till den önskade positionen på egen hand.

Figur 3.7: Figuren visar en referensstation [49]. (Återgiven med tillstånd)

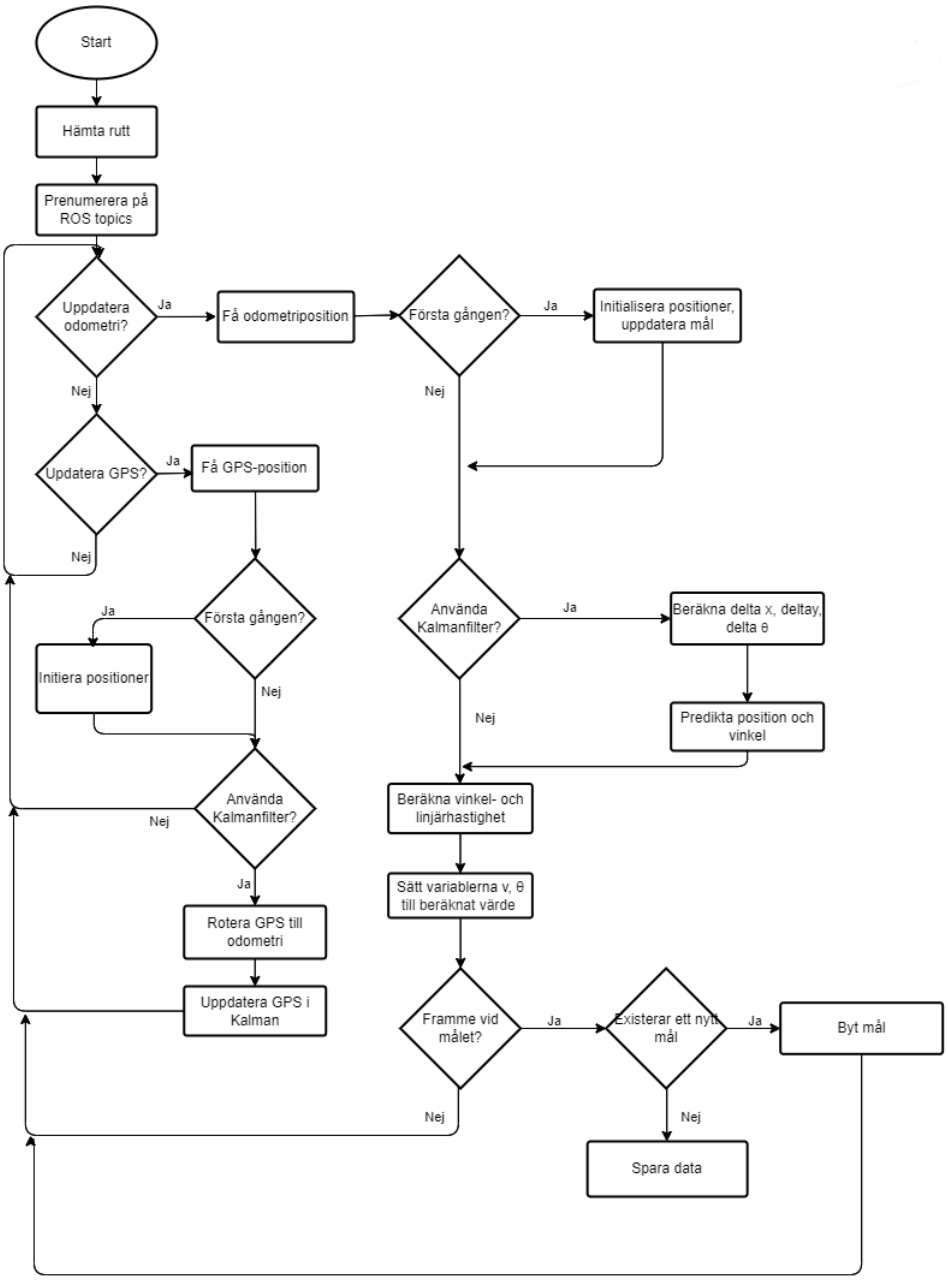
Eftersom arbetet med 450x hade kommit längre hade det skapats en main-fil med uppgift att samla in datan som önskades från sensorer. Detta för att sedan använda det för att beräkna hastighet och vinkelhastighet för att sedan kommunicera detta till gräsklipparen. Det som behövde göras för modell 550 EPOS var att översätta denna för att fungera på ROS2 samt att anpassa beräkningarna till den datan som mottogs.

För att beräkna positionen på 550 EPOS används den positionsdata som fås från basstationen samt från den inbyggda IMU som finns i gräsklipparen. Datan från basstationen är gräsklipparens relativa position från basstationen given i öst och norr. Från IMU fås vinkeln. Ett problem med datan från basstationen var att koordinatsystemet inte stämde överens med det som förväntades, det vill säga att värdet på norr respektive öst inte var värdet på avståndet från basstationen i dessa riktningar. Fallet istället var att koordinatsystemet var vridet 90°samt spegelvänt. Detta innebar att en omvandling av koordninatsystemet behövde implementeras. Det upptäcktes även att vinkeln som IMU gav inte stämde överens med vinklarna i koordinatsystemet och krävde omvandling. När detta upptäcktes var tiden som var avsatt för praktiskt arbete i princip slut. Detta innebar att det gjordes några försök att lösa det men utan resultat. Orsaker till dessa problem utreds mer i diskussionen, Avsnitt 5.2.

3.3 Programmeringskod

För att möjliggöra att gräsklipparen skulle köra behövdes mycket programmeringskod skrivas. Detta gjordes i Python 2.7 för 450x och Python 3.10 för 550 EPOS.

Detta eftersom 450x endast stödde en äldre version av ROS vilket endast var kompatibelt med en äldre Python version. Genom att prenumerera på ROS-noder kallas funktionerna på automatiskt vilket var tio gånger per sekund för odometrin och 1 gång per sekund för GPS. Hela koden finns bifogat i Bilaga A.1. För att göra koden lättläst presenteras koden som ett flödesschema i Figur 3.8.



Figur 3.8: Figuren visar ett flödesschema över hur programmeringskoden styr gräsklipparen. Hela koden finns i Bilaga A.1.

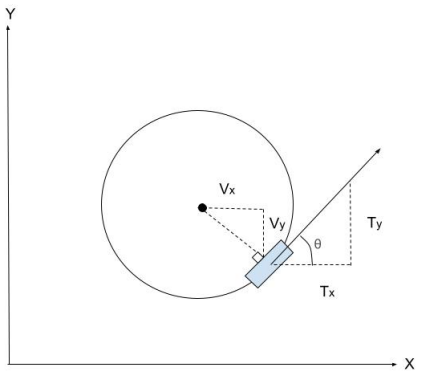
3.4 Reglering och styrning

Reglering tillämpas för att följa de vägar önskades. Två olika regulatorer utformades för att testa vilken regulator som var bäst för detta syfte.

3.4.1 PID

Till en början designades en PID regulator som förklaras mer i avsnitt 2.11.1 för att kunna följa en cirkel. Den designades genom att ställa roboten i en vinkel som var

ortogonal mot en linje från mittpunkten på cirkeln till cirkelbågen. Denna vinkeln beräknas i Ekvation 3.20 med beteckningar som visas i Figur 3.9.



Figur 3.9: Figuren visar hur målvinkeln beräknas för cirkelkörning.

$$V_x = x - x_{mid} \tag{3.13}$$
$$V_y = y - y_{mid} \tag{3.14}$$
$$|\hat{V}| = \sqrt{V_x^2 + V_y^2} \tag{3.15}$$
$$\hat{V}_x = \frac{V_x}{|\hat{V}|} \tag{3.16}$$
$$\hat{V}_y = \frac{V_y}{|\hat{V}|} \tag{3.17}$$
$$T_x = -\hat{V}_y \tag{3.18}$$
$$T_y = \hat{V}_x \tag{3.19}$$
$$\theta_{\text{mål}} = \text{Normalize}(\text{dir} + \arctan(T_y, T_x)) \tag{3.20}$$

Där Normalize funktionen åstakommer $\theta_{\text{mål}} < |\pi|$. Dir är vilket håll klipparen är riktad och är 0 för matematisk positiv rikting och π för matematisk negativ riktning. Beteckningarna är enhetliga med de i Python-koden. En sammanfattning för PID-regulatorn för cirkelkörning visas i Ekvation 3.25.

$$r_{\text{current}} = \sqrt{(x - x_{\text{mid}})^2 + (y - y_{\text{mid}})^2} \tag{3.21}$$

$$e_r = r - r_{\text{current}} \tag{3.22}$$

$$\frac{de_r}{dt} = \frac{e_r - e_{r,\text{prev}}}{dt} \tag{3.23}$$

$$e_\theta = \theta_{\text{current}} - \theta_{\text{goal}} \tag{3.24}$$

$$v_{\text{ang}} = \begin{cases} K_p \cdot e_r \cdot dt + K_i \cdot \sum e_r + K_d \cdot \frac{de_r}{dt} & \text{Om } |e_\theta| < 7° \\ K_p \cdot e_\theta \cdot dt & \text{Annars} \end{cases} \tag{3.25}$$

Det sista steget förklaras i de fallen att gräsklipparen har en stor differens relativt den verkliga vinkeln. Då kommer gräsklipparen att vinkla upp sig innan den forsätter sin körning. Värdena på K_p, K_i och K_d optimerades i en optimeringsfunktion vilket förbättrade cirkelkörning. Den minimerade en kost-funktion vilket bestod av tre komponenter, vilka multiplicerades ihop och alla viktade med exponenter beroende på hur viktig det ansågs vara. Den mest viktiga var summan av det kvadratiska felet. Den näst viktiga var antalet gånger gräsklipparen översteg 7°och därför behövde stanna. Den tredje mest viktiga var tiden till målet. Optimeringen gjordes med particle swarm optimization. Denna använde 30 ”partiklar” och flera hundra iterationer för att hitta de bästa värdena på K_p, K_i och K_d, alltså de som gav lägst kostfuntionsvärde. Optimeringen skedde då brus lades till i både hastigheterna och positionen i varje steg. När det sedan testades på robotgräsklipparen uppstod det fel i koden vilket gjorde att det inte fungerade robust vilket ej löstes på grund av tidsbrist. I praktiken användes endast en P-regulator som först riktade in sig mot målet och sedan körde dit och cirkelkörning hanns endast med i simulering.

3.4.2 LQR

För att jämföra PID-regulatorn med en annan regulator framtogs även en LQR-regulator som beskrivs i Avsnitt 2.11.2. För att den med hjälp av en kostnadsfunktion få ut en signal som försöker minska felet till noll utan att använda för mycket kraft från motorerna då den rättar till felet. En tillståndsmodell gjordes för olika typer av körningar. Tillståndsmodellen för cirkelkörning och körning rakt fram längs en linje såg likadana ut. Tillståndsmodellen visas i Ekvation 3.27. Modellen ser ut enligt ekvationen för att nästa tillstånd påverkas av föregående position och hur tillstånden förändras med hastigheten v och ω. För rotation runt sin axel togs en ny tillståndsmodell fram som gör att roboten bara snurrar på stället som visas i Ekvation 3.28. Den påverkas bara av föregående tillstånd och vinkelhasigheten ω. A och B matris kan jämföras mellan Ekvation 3.26 och Ekvation 3.27 respektive 3.28.

$$x_t = A \cdot x_{t-1} + B \cdot u_{t-1} \tag{3.26}$$

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta_{t-1}) \cdot dt & 0 \\ \sin(\theta_{t-1}) \cdot dt & 0 \\ 0 & dt \end{bmatrix} \begin{bmatrix} v_{t-1} \\ \omega_{t-1} \end{bmatrix} \tag{3.27}$$

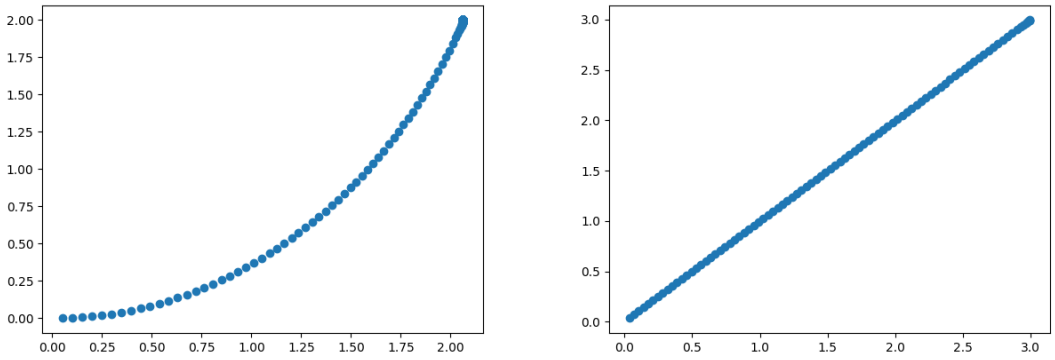
$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix} \begin{bmatrix} v_{t-1} \\ \omega_{t-1} \end{bmatrix} \tag{3.28}$$

I Ekvationen 2.9 används Q- och R matriserna nedan för att sätta kostnader på hur mycket roboten straffas för att vara fel eller hur mycket kraft den använder. Ekvation 3.29 är för tillståndsmodellen som kör rakt fram eller i cirklar och Ekvation 3.30 används för tillståndsmodellen med rotation på samma plats. Matriserna för Q och R varierar beroende på vilken tillståndsmodell som används och värdena i matriserna testades noggrant för att få bäst resultat då en angiven refernspunkt med värden på x_t, y_t och θ_t skulle maximeras. Vinkeln som skulle uppnås vid cirkelkörning räknades ut på samma sätt som i Ekvation 3.20 men utan funktionen Normalize.

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.8 \end{bmatrix}, R = \begin{bmatrix} 0.4 & 0 \\ 0 & 0.35 \end{bmatrix} \tag{3.29}$$

$$Q = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, R = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{3.30}$$

Figurna 3.10 visar första testerna då LQR i 3.10(a) tar sig till en punkt med en annan vinkel än den utgående och 3.10(b) visar när LQR tar sig till en punkt längre fram i samma vinkel som utgångsvinkeln.



(a) LQR till en punkt med annan vinkel än utgångsvinkeln.

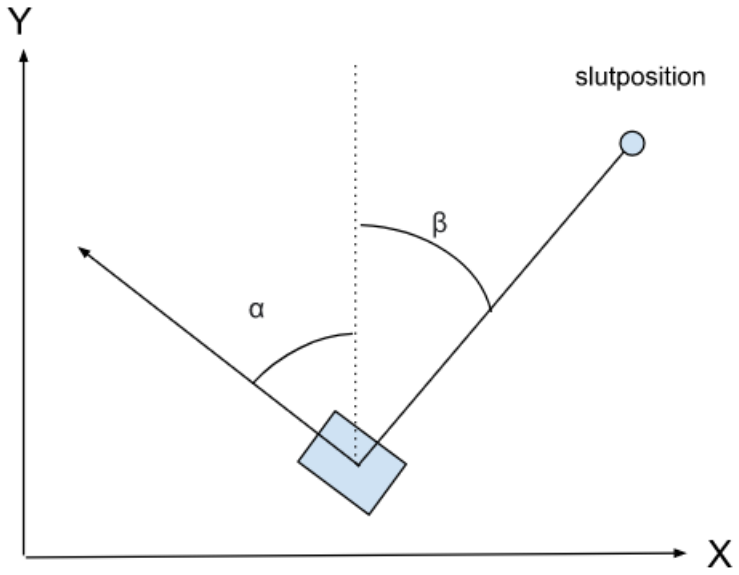
(b) LQR till en punkt med samma vinkel som utgångsvinkeln.

Figur 3.10: Figuren visar första testen av LQR.

3.4.3 Styrning

Den data som gräsklipparen använder sig av för att kunna åka från en position till en annan är avståndet från nuvarande position till önskad position, samt vinkeln som skiljer dem åt. Styrningen av gräsklipparen sker genom att reglera vinkelhastigheten

för det bägge hjulen, vid samma vinkelhastighet rör den sig rakt fram och vid olika hastigheter i en cirkel eller runt sin axel. I Figur 3.11 syns en modell av roboten där vinkeln α visar körriktningens vinkel i förhållande till Y-axeln samt vinkeln β som visar vinkeln till slutpositionen. Gräsklipparen räknar ut skillnaden till önskad position och med hjälp av en P-regulator roterar runt sin axel till den kör mot rätt riktning. P-regulatorn består av en kostant som ser till att rotationshastigheten är hög när felet är stort och saktar ner när den börjar närma sig sitt målvärde.
När gräsklipparen positionerat sig åt rätt håll, inom 7°börjar den köra mot slutpositionen. Distansen mellan slutpositionen och gräsklipparens nuvarande position definieras som linjära felet, även här används en P-regulator som signalerar om hög hastighet när felet är stort och och lägre när den är nära slutpositionen. En maxhastighet bestäms dock för att uppnå en kontrollerad körning. Koden som styr beräkningen av hjulhastigheten finnes i Bilaga A.2



Figur 3.11: Figuren visar en modell för styrning av klipparen.

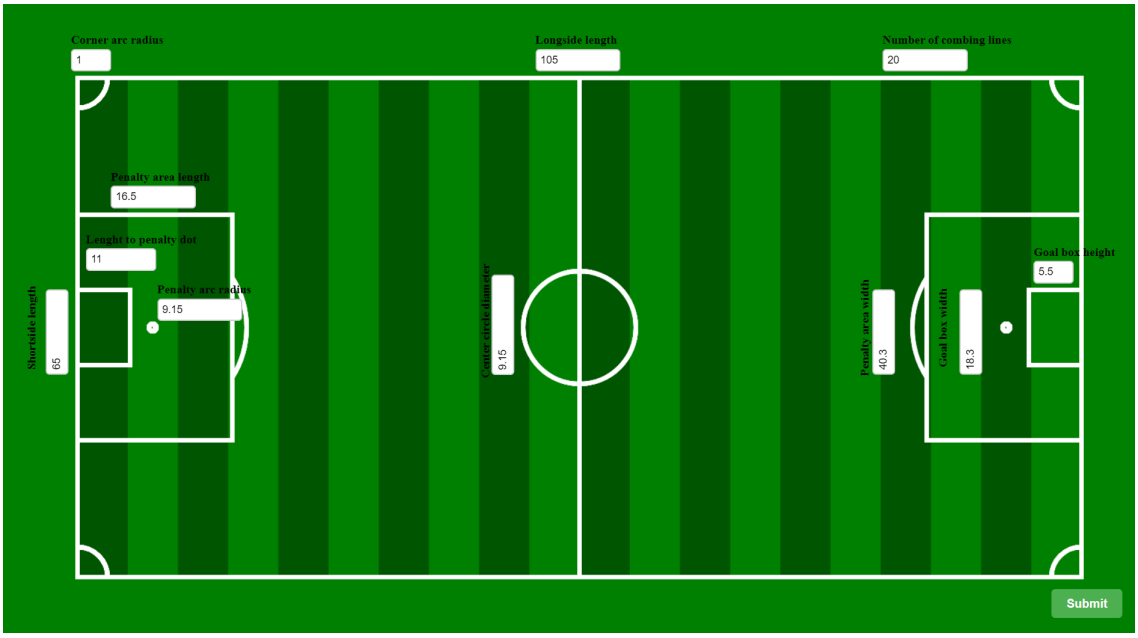
3.5 Måla linjer

För att måla linjer krävdes en rutt för gräsklipparen att följa som optimerar körvägen och ger sammanhängande linjer. För att följa denna rutt behöver den använda sig av sin position som är beskrivet i tidigare sektion. När gräsklipparen är på rätt ställe ska den börja måla linjer. Denna sektion behandlar hur dessa delar arbetats med.

3.5.1 Längder av linjer

För att göra produkten användarvänlig, byggdes en lokal hemsida där användaren enkelt kan skriva in vilka linjer den vill måla och hur långa dessa linjer ska vara. Användaren anger även in hur många kamningslinjer som gräsklipparen ska kamma i vilket ändras dynamiskt. Detta gör att användaren kan se precis hur planen kommer

se ut innan den ska målas. Detta är nödvändigt eftersom alla fotbollsplaner är olika stora och möjliggör även att användaren kan välja att måla exempelvis en rugbyplan istället, vilket gör produkten betydligt mer användbar. Detta eftersom en kommun kan använda den till många olika idrottsanläggningar. Att den även kan styras utan stor teknisk kompetens gör även att produkten skulle vara mer attraktiv. När användaren sedan trycker på ”submit”, kommer informationen automatiskt sparas och linjerna kommer att målas utefter användarens instruktioner. Instruktionerna för att använda verktyget beskrivs i filen ”README.md vilken finns i Bilaga A.7.



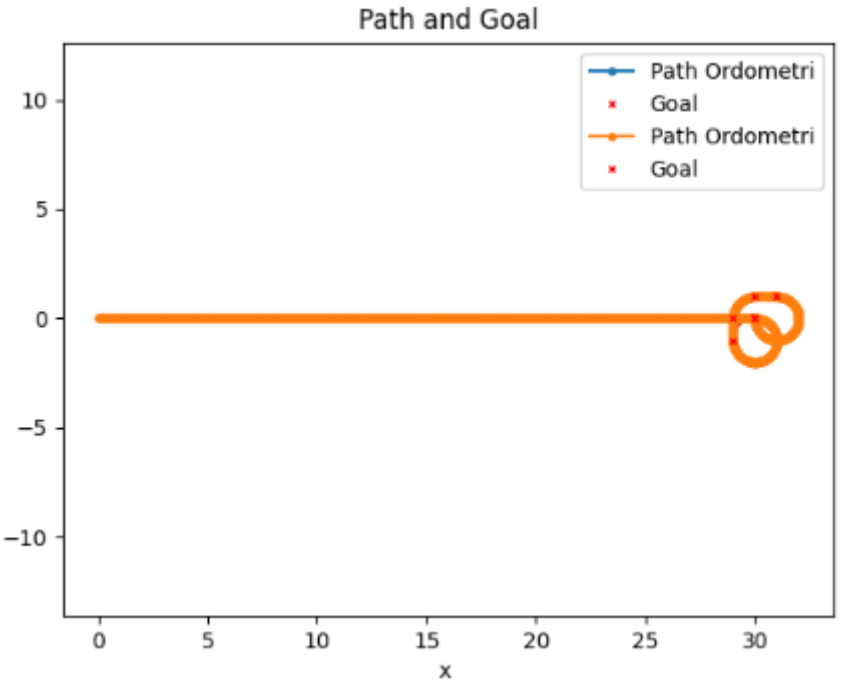
Figur 3.12: Figuren visar den lokala hemsidan där användaren enkelt kan ändra linjernas längd och antalet kamningslinjer.

3.5.2 Ruttplanering

För att kunna måla samtliga linjer på fotbollsplanen behövdes en algoritm för ruttplanering skapas. Ruttplaneringen innehåller all information för dimensioner på fotbollsplanen men är också optimerad för att åka kortas möjliga transportsträckor mellan linjerna. Fotbollsplanen delades upp i en mängd olika partier, till exempel, sidlinje, mittcirkel och straffpunkt. Algoritmen bestämmer sen i vilken ordning i förhållande till varandra som linjerna ska målas. Användaren kan bestämma hurvida hela fotbollsplanen ska målas eller om bara ett visst område ska målas, exempelvis straffområdet på grund av algoritmens egna koordinatsystem. Algoritmen är utformad på sättet att den alltid målar färdigt en linje innan den påbörjar nästa, exempelvis kör den inte halva sidolinjen för att sen börja måla mittlinjen för att sen fortsätta måla sidolinjen. Detta för att undvika fula kanter eller diskoninuiteter i linjerna då det alltid finns en liten felmarginal.

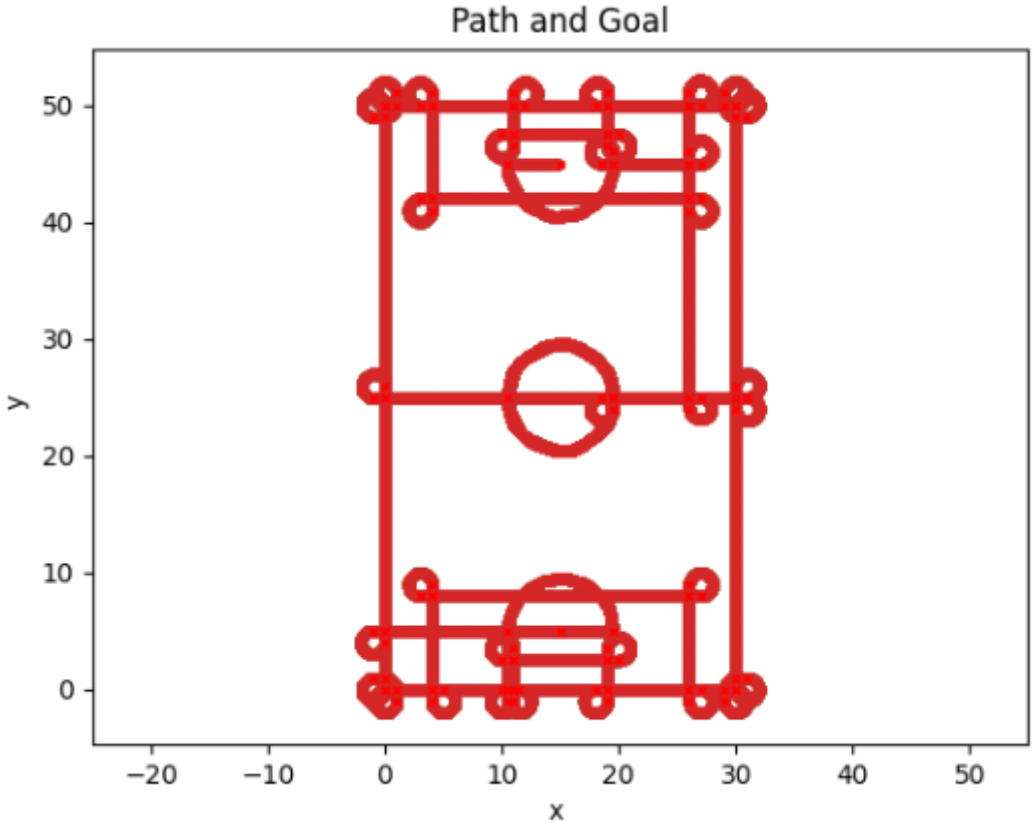
Det upptäcktes att gräsklipparen, vid stillastående position, roterar runt sin axel får sina framhjul snedställda. Detta gör att när gräsklipparen ska röra sig framåt

igen precis vid startmomentet kör snett. För att kontrollera detta sågs det till att robotgräsklipparen kör i 270°cirkel för att sedan vara rättvänd för att måla nästa linje. I Figur 3.13 syns det hur algoritmen kör när den kört ena kortsidan, sedan målar linjen vid hörnflaggan för att sedan vara positionerad för att börja måla långsidan.



Figur 3.13: Figuren visar målning av nedre kortlinje samt runt hörnflagga.

För att kunna måla linjerna behövdes det, utöver rutter för att gräsklipparen ska vara riktad åt rätt håll, ytterligare transportsträckor för att köra mellan linjerna. En transportsträcka är en rutt där linjer inte målas och ska därmed vara kortast möjliga för att kunna uppnå snabbast möjliga rutt och minimera körtiden. Figur 3.14 visar en simulering för hur gräsklipparen kört och målat hela fotbollsplanen. I simuleringen har gräsklipparen först målat ytterlinjerna av fotbollsplanen utan behov av några transportsträcka, för att sedan måla nedre straffområdet, nedre målområdet, mittlinje, mittcirkel och till sista över straff- och målområde. Det går att se i figuren konturerna av en fotbollsplan med tillhörande linjer, samt extra transportsträckor och svängar av antingen 270°eller 180°. Den fullständiga ruttplaneringen finns i Bilaga A.5.

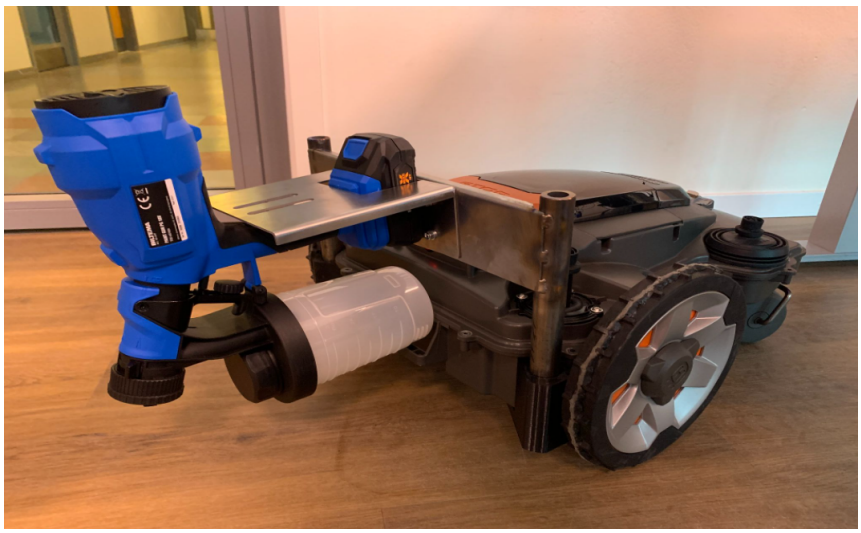


Figur 3.14: Figuren visar simulerad rutt för hela planen. I simuleringen används en P-regulator för att köra raka linjer och PID-regulator för att köra i cirklar.

3.5.3 Färgspruta

För att kunna måla linjerna valdes det att köpa en batteridriven färgspruta från Biltema [50]. Den batteridrivna färgsprutan är mobil och är i storlek rimlig i förhållande till gräsklipparen. Kostnaden är också relativt låg jämfört med liknande lösningar på marknaden. Färgsprutan med batteri och maximal mängd färg vägde ungefär 4,5 kilogram vilket medförde ett hållfasthetkrav på hållaren. För att kunna fästa färgsprutan på gräsklipparen behövdes det en hållare för färgsprutan. Valet av material för hållaren föll därmed på en två millimeter tjock plåt av rostfritt stål. Konstruktionen är byggd av en mittendel, två sidodelar och två stänger. Mittendel är utformad för att färgsprutan med batteri ska passa in och med avlånga hål framtill för att kunna spänna fast sprutan med spännband. Bottensidan av mittendelen är bockad för att kunna sättas ihop med sidodelarna. Sidodelarna är utformade med två avlånga hål för att kunna fästas i mittendelen, samt för att kunna justera bredden på hela annordningen. Sidodelarna är i sin tur svetsade i var sin stång. Sedan tidigare fanns två 3D-printade hållare med 27 millimeter i diameter hål inuti fästa på baksidan på gräsklipparen, inuti dessa kunde stängerna föras in. Hela konstruktionen kan alltså fästas på gräsklipparen utan skruv eller dylikt för fasthållning, vilket leder till smidig montering. Ett problem som uppstod var att gräsklipparen blev baktung och tippade bakåt vid framåtacceleration, detta kompenserades för

genom att lägga in vikter framtill i gräsklipparen. Hela annordningen tillsammans med färgspruta och gräsklippare illustreras i Figur 3.15.

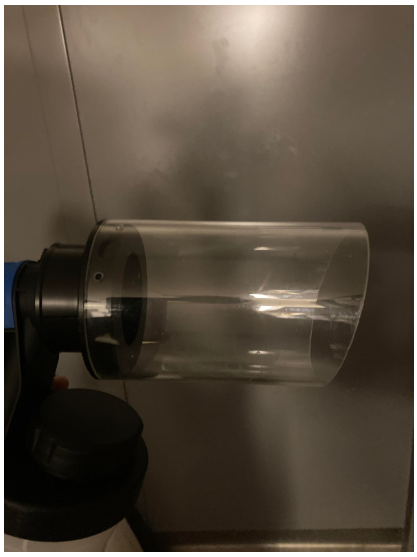


Figur 3.15: Figuren visar gräsklipparen utan skal med färgsprutarkonstruktionen.

Färgsprutan sprutar färg genom att trycka in en avtryckare. För att kunna måla linjer automatiskt behövdes därmed ett sätt för att avlösaren att gå av vid de ställen som linjer ska målas, samtidigt som gräsklipparen rör sig framåt. Detta löstes genom att montera en servomotor på hållaren. På rotorbladet av servomotorn fästes ett snöre som drogs runt avtryckaren och fästes igen undertill på hållaren. Servomtorn är kopplad till mikrokontrollern som med hjälp av algoritmen för ruttplanering meddelar gräsklipparen på vilka positioner den ska måla. Pulsbreddsmodulering ser till att roterar till önskad vinkel. Färg sprutas därmed vid rotation av servomotorn som drar åt snöret och avlöser avtryckaren och slutar måla när servon roterar tillbaka till ursprungspositionen. I Figur 3.16 illustreras servomotorn som fästs på hållaren som drar in avtryckaren på färgsprutan.

Figur 3.16: Figuren visar en servomotor monterad på hållaren för färgsprutan som vid rotation drar in avtryckaren för färgsprutan.

Linjerna på en fotbollsplan har ett krav på att vara 11 centimeter breda och för att få färgsprutan att måla korrekt bredd behövdes därmed en konstruktion även för detta, därtill är det oönskat att färg sprids till bakom den punkt som linjen är tänkt att börja. Figur 3.17 visar ett plaströr med innerdiameter 11 centimeter som fästes i munstycket på färgsprutan. Röret gör att färg enbart målas precis undertill färgsprutan och sprids varken bakom, framför eller brevid linjen.



Figur 3.17: Figuren visar röret som ser till att färgsprutan målar rätt bredd på linjerna.

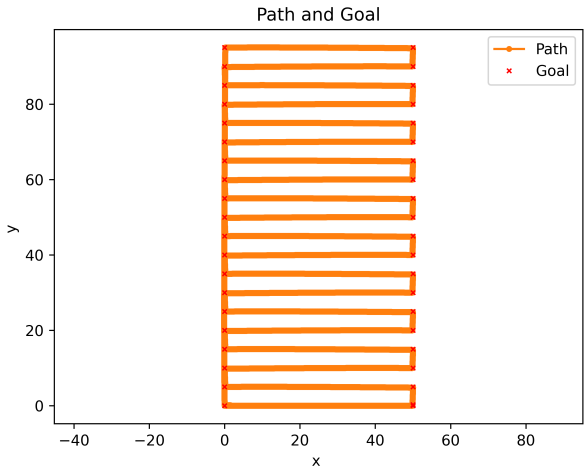
3.6 Kamning

För att kamma planen krävs en rutt som är optimerad för att minska körtiden samt kammar gräset. För att följa denna rutt används gräsklipparens position samt att den ges en målposition som den ska ta sig till. Algoritmerna i detta avsnitt kan även användas för att klippa gräset, men då gräsklipparen inte hade knivar har vi fokuserat på kamningen.

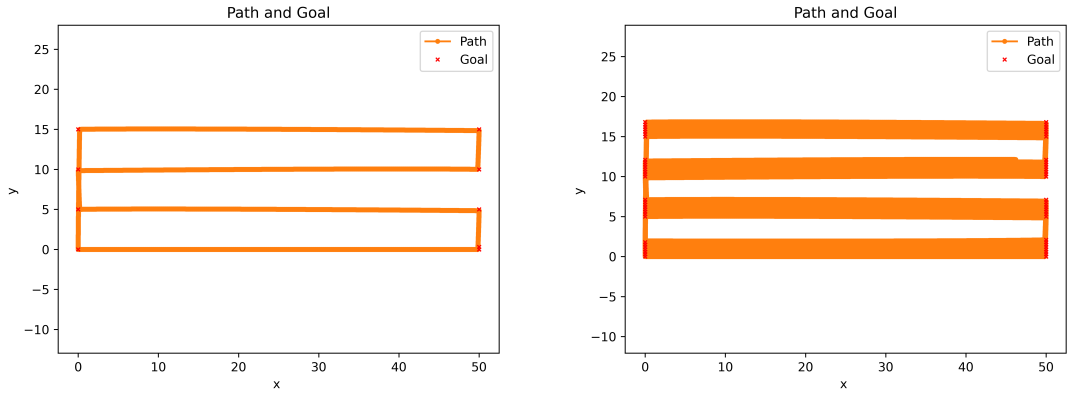
3.6.1 Ruttplanering

Algoritmen som gör detta är skriven i Python och tar in informationen som fås i användargränssnittet beskrivet i Avsnitt 3.5.1 om planens längd och bredd samt antal ränder på planen. Utifrån dessa värden beräknas bredden och längden på varje rand samt antal resor som krävs för att hela randen ska bli kammad. Därefter ger algoritmen gräsklipparen dess målposition som antingen innebär att den ska köra till starten på en rand eller köra från starten till målet på randen. Dessa målvärden kommer från den beräknade bredden och längden på randen. Gräsklipparen fortsätter tills den har gått en gång på varje rand och åker sen tillbaka till starten men har då en offset på kammens bredd gentemot den tidigare startposition och sen upprepar. Algoritmen är uppbyggd på två stycken for-loopar där den första hanterar att åka på varje rand och den andra hanterar antalet resor som krävs totalt för att hela planen ska bli kammad samt att se till att det blir en offset från den tidigare resan, den fullständiga ruttplaneringen finns i Bilaga A.6.

För att testa algoritmen innan den används i praktiken gjordes simuleringar. Figur 3.18 visar när den första resan över planen är gjord. Figur 3.19 visar hur det ser ut efter ungefär halva tiden, här illustreras det att varje rand fylls på med tiden. Figur 3.20 visar när hela planen kammats. I simuleringen vissa rutten för 20 stycken ränder och då planens längd är 100 meter och bredd 50 meter.



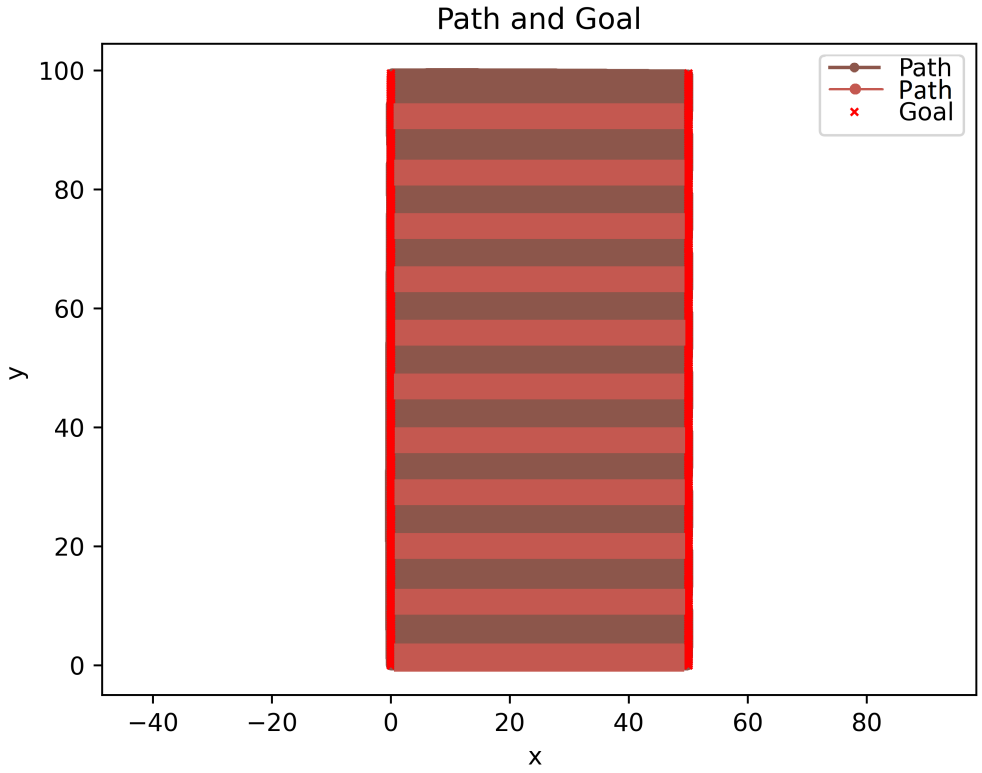
Figur 3.18: Figuren visar hur simuleringen ser ut när gräsklipparen har gått första iterationen över planen och sen tillbaka till starten.



((a)) Figuren visar simulerad rutt för första iterationen över planen.

((b)) Figuren visar samma simulering efter att ungefär hälften av iterationerna över planen är gjorda.

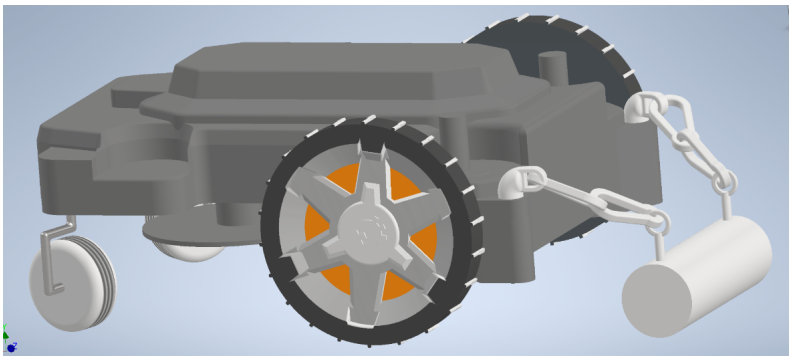
Figur 3.19: Figurerna visar simulering med inzoomning så att en femtedel av planen syns.



Figur 3.20: Figuren visar när hela planen är kammad. Där de olika nyanserna innebär olika färdriktningar över planen.

3.6.2 Kam

För att kamma planen används en kam som fästs bakom gräsklipparen. Kamningen sker samtidigt som gräset klipps. Kammen behöver vara av viss tyngd för att kunna dra gräset i åkriktningen men samtidigt inte påverka gräsklipparens prestanda i för hög utsträckning. CAD användes för att visualisera olika versioner. Detta sparade tid och resurser genom att eliminera behovet av att bygga och testa lika många prototyper. Lösningen som illustreras i Figur 3.21 visar en CAD-modell med en cylinder med åtta centimeter diameter och en längd på 24 centimeter, samma bredd som klippdiametern. Cylindern är kopplad till ögleskruvar och karbinhakar som fästs i två 3D-printade cylindrar som sätts i samma fastmonterade hållare som hållaren för färgsprutan. Figur 3.22 visar den färdiga konstruktionen av kammen som består av ett tennisrör med åtta centimeter diameter och en längd på 28 centimeter. Bredden på kammen blev därmed något bredare än klippdiameten men kan användas för illustrera hur kamning sker. Volymen på cylindern blev 0,0014 m^3 och den fylls med grus med en densitet på 1500 kg/m^3 vilket ger en totalt vikt på 2,1 kg.



Figur 3.21: Figuren visar gräsklipparen utan skal och första CAD av kam.

Figur 3.22: Figuren visar gräsklipparen utan skal och kamkonstruktion.

4
Resultat

Resultatdelen består av praktiska och simulerade resultat. I simuleringen kör modellen utefter samma parametrar som gräsklipparen gör i verkligheten men utan GPS. Därmed ger de simulerade testerna en bra bild för hur den tillämpade regleringen av gräsklipparna fungerar tillsammans med ideala GPS-signaler.

När tester gjordes på de riktiga gräsklipparna kunde det konstateras att GPS-signalerna för gräsklipparen av modell 450x var för opålitliga för att kunna styra den med tillräckligt god precision för att måla linjer på en fotbollsplan, vilket diskuterades redan i planeringsrapporten. För gräsklipparen av modell 550 EPOS lyckades inte GPS-punkten från koordinatsystemt som basstationen angav tolkas på ett sätt som gjorde att den kunde styras utefter angivelse av koordinater. Problemet med cirkelkörning löstes ej i praktiken eftersom fokus låg på att få klipparen att köra rakt. Detta eftersom klipparen behöver kunna köra rakt för att kunna köra i mönster. Därav fokuserade de praktiska testerna på att köra gräsklipparen i en rak linje med en slutposition så nära den önskade som möjligt. Tester för hur målning av linjer och kamning gick till presenteras också i denna resultatdel.

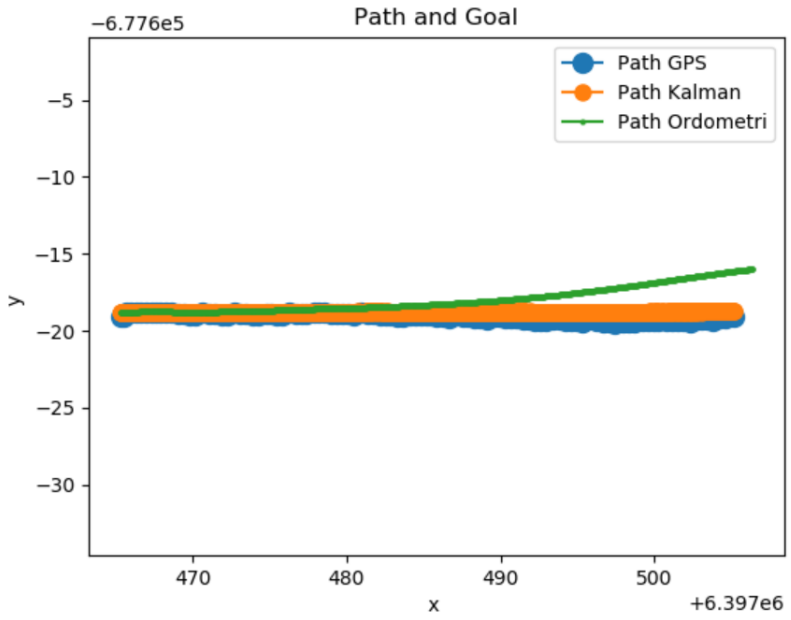
4.1 Navigering

I detta avsnitt presenteras resultaten av navigeringen för respektive gräsklippare samt resultaten göt kamning och linjemålning.

4.1.1 Navigering 450x

Tester för navigeringen av 450x gjordes genom att köra gräsklipparen mellan två punkter längs mittlinjen. För att kunna applicera Kalmanfiltret undersöktes vilken riktning som GPS:en tolkar att den åker i förhållande till odometrin. Därför lyftes klipparen längs mittlinjen i 60 meter flera gånger för att tolka en vinkelskillnad mellan GPS-signalerna och odometrin för att sedan flytta de uppmätta punkterna för att de ska hamna i samma riktning som odometrin. När detta var gjort testades gräsklipparen att köra 40 meter till en slutpunkt rakt fram flera gånger med och utan Kalmanfilter aktiverat. Figur 4.1 visar hur gräsklipparen med hjälp av Kalmanfilter tolkar hur den kör vid det bästa testet. Enligt GPS:en åker roboten rakt fram medan odometrin svänger åt vänster. I praktiken ledde detta till en felmarginal på mellan 0,1 och 0,5 meter, beroende på vilket test, efter en 40 meters körning. Figur 4.2 visar en bild med avstånd från linjen när gräsklipparen körde med en felmarginal på

max 0,3 meter från linjen och stannar där bilden visar som är ungefär tio centimeter från önskad position. Hela detta test visas även i videon som spelades in som kan visas genom att använda länken: https://youtu.be/LBpYDC6b2uc [51]. Resultaten som är beskrivna ovan erhölls när Kalmanfiltret använde sig av positionerna x, y och vinkeln från odometrin samt GPS-punkter. Kalmanfiltret uppskattade därför positionen och vinkeln vilket sedan reglerades rätt med hjälp av en P-regulator.



Figur 4.1: Figuren visar slutpositionen efter 40 meters körning med Kalmanfiltret utefter mittlilnjen.

Figur 4.2: Figuren visar GPS, odometri och Kalmanfiltrets väg vid körningen i Figur 4.1.

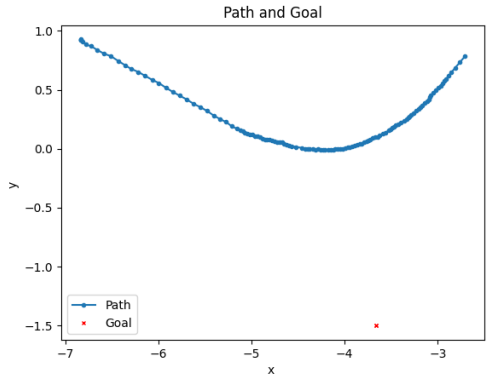
När gräsklipparen testades att köra utan Kalmanfiltret aktiverat och bara körde på odometri blev felmarginalen mellan 10 - 15 meter vilket visas i Figur 4.3.

Figur 4.3: Figuren visar slutpositionen efter 40 meters körning med endast odometri utefter mittlilnjen.

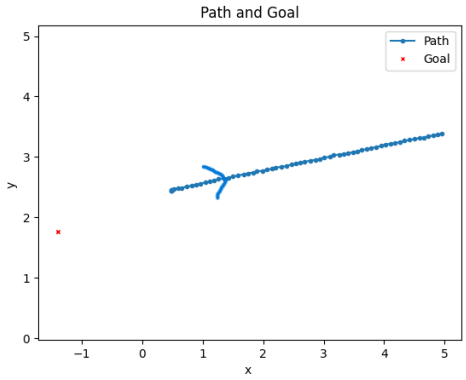
4.1.2 Navigering 550 EPOS

För gräsklipparen av modell 550 EPOS gjordes inga slutliga tester för att presentera resultatet. Under processen för att få klipparen att åka utefter angivelse kunde inte klipparen åka åt det håll som önskades. Figur 4.4 visar hur gräsklipparen åker jämfört med vilket mål den har.

I Figur 4.4(a) visar resultat när gräsklipparen ges en position och att den ska köra dit. Resultaten i denna Figur är ifrån innan problemet med koordinatsystem som beskrivs mer i Avsnitt 3.2.3 upptäcktes. Det som kan observeras är att när gräsklipparen ska reglera vinkeln för att hålla kursen viker den av att andra hållet istället. I Figur 4.4(b) åker den åt fel håll. Orsaken till detta är inte klart och tas upp mer i diskussionen, Avsnitt 5.2.



((a)) Figuren visar hur gräsklipparen av modell 550 reglerar för att hålla kursen mot målet men istället viker den av åt fel håll.



((b)) Figuren visar hur gräsklipparen av modell 550 åker åt fel håll.

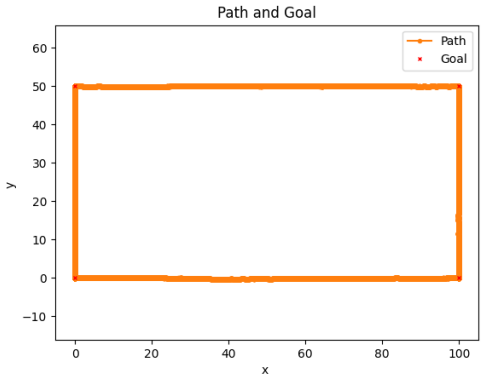
Figur 4.4: Figurerna visar hur gräsklipparen av modell 550 åker efter den har fått en målposition.

4.2 Simulering

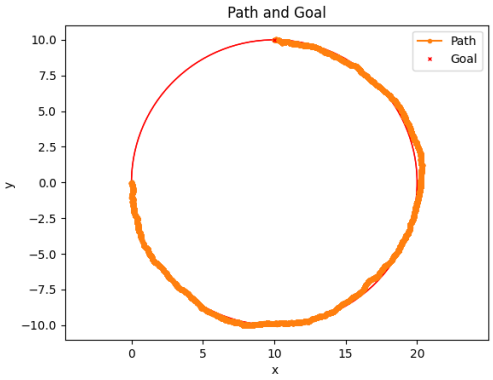
För att jämföra PID- och LQR-regulatorn gjordes liknande tester för de båda. Testerna bestod av körning av en tre fjärdedelars cirkel med en radie på tio meter och en rektangel på 100 x 50 meter för att illustrera mittcirkeln på en fotbollsplan respektive sidlinjerna på en fotbollsplan. Brus lades till för att simulera mer verklighetstrogen situation. Detta medförde att felet på positionen i varje steg kunde vara tre millimeter och felet på hastigheten kunde variera med 0,003 meter per sekund och vinkelhastigheten 0,003 radianer per sekund. Detta uppdaterades tio gånger per sekund vilket resulterar i verklighetstrogna brus. För att jämföra LQR med PID användes det slutgiltiga felet vilket är sträckan från målpunkten till dit gräsklipparen skulle kommit.

4.2.1 PID

Vid simulation av en rektangel med dimensioner 50x100 meter erhölls Figur 4.5(a) vilket gav det slutgiltiga felet 0,15 millimeter. Vid simulation av en 270°cirkel med tio meters radie erhölls Figur 4.5(b). Det slutliga felet var 3,9 millimeter.



((a)) Figuren visar då PID-regulatorn kör i en rektangel med dimensioner 50x100 meter. Det slutgiltiga felet var 0,15 millimeter.

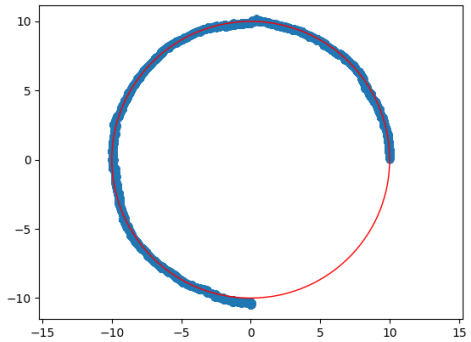


((b)) Figuren visar då PID-regulatorn kör i en 270°cirkel med tio meters radie. Det slutgiltiga felet var 3,9 millimeter.

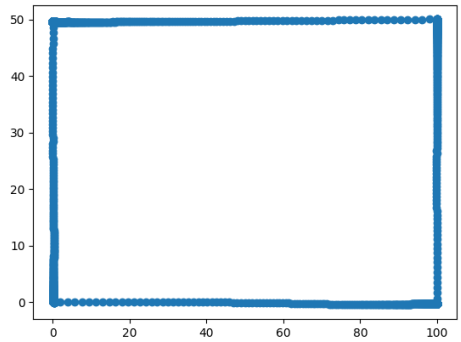
Figur 4.5: Figurerna visar PID-regulatorn i två olika mönster med brus. Det slutgiltiga felet är felet från målet som gräsklipparen slutar på.

4.2.2 LQR

Vid test av LQR modellen av cirkelkörning delades cirkeln upp i 100 punkter och 75 av dem punkterna kördes i moturs riktning. Resultatet visas i 4.6(a) och cirkeln följs någorlunda bra men totala felet i sista punkten är ungefär 0,3 meter men tittas det noggrannare i figuren blir felet större då roboten pekar i riktning med vinklarna π, $\frac{3}{4}\pi$, 2π (punkterna: 0,10; -10,0; 0,-10). Testet av rektangeln som visas i Figur 4.6(b) gjordes med endast fyra punkter i hörnen då uppgiften blev att köra till nästa punkt och sedan rotera på plats i riktning till nästa punkt. Vinkelfelet blev noll och slutfelet ungefär 0,2 meter men ibland driftade roboten iväg lite längs sträckorna pågrund av störningarna som lades till.



((a)) Figuren visar LQR av en vid cirkelkörning med 75 av 100 körda punkter.



((b)) Figuren visar LQR vid körning av rektangel med fyra punkter.

Figur 4.6: Figurerna visar hur LQR behandlar två olika mönster med brus.

4.3 Målning av linjer

Vid tester för målning av linjer monterades färgsprutan med hjälp av hållaren på gräsklipparen av modell 450. Det uppstod problem med att synkronisera servomotorn med mikrokontrollen, Jetson Nano 2GB Developer Kit, där den inte detekterade det kort som hanterade pulsbreddsmodulering och servomotorn kunde därmed inte appliceras. Istället fick målning ske genom att manuellt trycka in avtryckaren med hjälp av tejp vid körning. Konstruktionen för hållaren av färgsprutan med vätska påfylld gör gräsklipparen baktung vilket kompenserades för med vikter framtill.

Målningen testades genom att fylla vätskehållaren med vatten och röd karamellfärg. Gräsklipparen skulle måla längs mittlinjen på en fotbollsplanen rakt fram till en punkt medan avtryckaren för färgsprutan hålls in med tejp. Då volymen för vätskehållaren var begränsas till 0,9 liter och det inte går att reglera mängden färg som sprutas var målartiden begränsad till ungefär 30 sekunder. Resultatet blev att färgen målades i rätt bredd i förhållande till fotbollslinjerna. Då navigeringen inte tillät gräsklipparen köra i andra mönster än raka linjer testades inga alternativa körriktningar. Vilket innebar att ett test av målningsalgoritmen var inte möjlig i verkligheten för andra mönster än raka linjer. Däremot fungerade målningsalgoritmen bra i simulering vilket visas i Figur 3.14. Med mer tid skulle målningsalgoritmen troligtvis även fungera i praktiken också. Figur 4.7 visar när gräsklipparen kör längs linjen och målar samtidigt.

För gräsklipparen av modell 550 EPOS kunde inte linjemålning testas då navigeringen inte fungerade. Dock testades avtryckaren med hjälp av servomotorn på denna gräsklippare. Den visade sig fungera. Servomotorn som är kopplad till Raspberry pi drar åt färgsprutan och aktiverar målning utefter kommandon av användaren.

Figur 4.7: Figuren visar när gräsklipparen försöker måla längs mittlinjen.

4.4 Kamning av gräs

Eftersom navigeringen i gräsklipparen inte möjliggjorde körning utefter kamningsalgoritmen, testades kamningen genom manuell styrning av gräsklipparen. Kamningskonstruktionen fästes baktill och gräsklipparen kördes sedan upp och ner i en bredd på ungefär två meter vardera genom att styra den med piltangenterna. I Figur 4.8 går det se att mittenpartiet av gräset är ljusare än gräset brevid. Där det är ljusare har gräsklipparen rört sig uppåt och där det är mörkare rörde sig gräsklipparen nedåt relativ bilden. Mönster syns alltså i gräset när gräsklipparen kört med kamning. Mönstrena förstärks i fallet att gräset är ännu tätare och åskådas från längre och högre håll.

Figur 4.8: Figuren visar resultat av kamning i straffområdet.

4.5 Kostnader

I Tabell 4.1 visas en sammanställning av alla inköpta delar. Här ingår inte robotgräsklipparna då de båda fanns att tillgå och blev därmed ingen kostnad för projektet.

Produkt	Antal	Kostnad (per st)
Jetson Nano 2GB Developer Kit	1 st	750 kr
SD-kort	1 st	230 kr
Färgspruta PG 18V med batteri	1 st	867 kr
Usb-can	1 st	402 kr
Wifi dongles	2 st	300 kr
Cooling RPI	1 st	200 kr
Nano pwn	1 st	140 kr
IMU1.0	1 st	250 kr
SparkFun 9DoF IMU Breakout - ICM-20948 (Qwiic)	1 st	222 kr
Qwiic cable	1 st	16,9 kr
Ögleskruv	2 st	35 kr
Karbinhake	2 st	65 kr
Total kostnad		3873,9 kr

Tabell 4.1: Tabellen visar kostnaderna i projektet.

4.6 Måluppfyllnad

I Tabell 4.2 listas de mål som angavs i början av projektet och i vilken grad de uppfyllts.

Mål/Modell	450x	550 EPOS	Simulering
Fjärrstyrning med dator	Ja	Ja	Ja
10% felmarginal i förhållande till rutt	Ja	Nej	Ja
Navigera med max 20 centimeters fel	Ibland	Nej	Ja
Kamma synliga mönster	Ja	Ja	-
Styra färgspruta automatiskt	Nej	Ja	Ja
Navigera på en fotbollsplan (precision)	Ja (0,1-0,5 m)	Nej	Ja
Köra i cirklar	Nej	Nej	Ja

Tabell 4.2: Tabellen visar de mål som angavs i början av projektet.

5
Diskussion

Genom att analysera resultatet av projektet kan vissa slutsatser kring projektets utfall göras med kopplingar till de ursprungliga problem- och uppgiftbeskrivningarna samt målen. Det kan konstateras att arbetet inte uppnådde alla delmål som sattes upp från början. Ambitionerna för projektet var höga och projektet hade möjligen behövts smalnats av något i ett tidigare skede. Den främsta anledningen till att projektet inte nådde hela vägen som önskat var dock att det underskattades hur lång tid det skulle ta att börja kontrollera gräsklipparna. Förhoppningsvis kan detta projekt ligga som grund för framtida projekt att i ett tidigare stadie kunna styra sina gräsklippare och därför komma längre i sina projekt. Nedan listas och diskuteras de ursprungliga problemen och hur resultaten motsvarar dem.

5.1 Styra gräsklipparna

Målet om styrning av och kommunikation med gräsklipparna av både modell 450x och 550 EPOS kan bedömas vara delvis uppfyllt. Styrning berodde till en början av att kommunikationen mellan användare och gräsklipparen upprättas. Detta har skett genom en mängd mjukvaror och till sist genom program i Python. I Python har samtliga algoritmer för styrning av gräsklippare, simulering av körning, input av gräsklipparens sensorer etc. skapats och samlats i en fil i Visual Studio Code. Genom att ta in input av gräsklipparens sensorer om hjulhastighet och vinkel kan gräsklipparna instureras att exempelvis köra en meter framåt. Gräsklipparen av modell 450x gick också köra i olika riktningar med hjälp av enbart odometrin medan 550 EPOS inte svängde åt de håll som önskades.

5.2 Navigera i sin omgivning

Navigeringen för gräsklipparen av modell 450x uppnådde det basala delmålet att köra till en punkt med en felmarginal på 10 % i förhållande till längd på körsträcka. Detta eftersom gräsklipparen körde ungefär 10-50 centimeter fel på 40 meter var felet ungefär 0,5 % i förhållande till längd på körsträcka. Den uppnådde även ibland det avancerade delmålet att köra med en tolerans på 20 centimeter. Däremot, eftersom detta inte uppnåddes varje gång anses detta delmål inte som uppnått. De andra delmålen redovisas i Tabell 4.2.

Koordinaterna från GPS:en förändrades över tid. Detta berodde troligen på att atmosfären förändras till följd av tid på dagen och vädret och en basstation reducerar

detta fel. Detta gör att kalibreringen behövde utföras inom någon timme innan testkörningen. Eftersom kalibreringens noggrannhet skiftade mellan olika gånger skiftade även noggrannheten navigeringen mellan olika dagar och kalibreringar. Den inbyggda GPS:en har en stor felmarginal på ungefär tio meter. Det är därför mycket ambitiösa mål att kunna måla en fotbollsplan med denna GPS. Trots att det troligen fanns saker som kunnat förbättra positionen något var GPS:ens stora felmarginal den huvudsakliga anledningen till att gräsklipparen inte kunde måla linjerna på en fotbollsplan med tillräcklig noggrannhet. Detta var även känt ifrån början eftersom GPS:en är gammal och inte är utformad för mycket noggranna mätningar. Genom att applicera ett Kalmanfilter och en regulator kunde GPS:ens stora felmarginalen minskas markant. Utan Kalmanfilter hade gräsklipparen ett fel på ungefär tio meter vid körning på 40 meter och med Kalman en felmarginal på oftast under 0,5 meter för samma distans vilket är en betydande förbättring.

Genom att använda en IMU hade noggrannheten på navigeringen kanske kunnat förbättras samt undvika tidskrävande kalibrering. Detta eftersom med hjälp av IMU skulle vinkeln mellan odometrin och GPS kunna beräknas utan kalibrering. Detta hade kunnat underlätta projektet eftersom kalibreringssteget hade undvikits. Däremot hade det kunnat bli sämre positionering eftersom IMU använder magnetfältet för att hitta vinkeln mellan GPS och odometri. Däremot tar detta ej hänsyn till att GPS:en kan vara förskjuten på grund av atmosfären. För att testa ifall detta hade givit ett bra resultat köptes en IMU och denna testades att använda i gräsklipparen. Däremot blev utsignalen från IMU:n dock inte konsistent med riktningen mot norr samt att utsignalen var mycket brusig. Möjligen störddes IMU:n förmåga att uppfatta väderstrecken av något magnetiskt i gräsklipparens motorer. IMU skulle hjälpa projektet med att få ut bättre mätbar vinkel vilken skulle kunna användas i Kalmanfiltret och därmed hade roboten kunnat åka rakare. Däremot hade detta inte hjälpt roboten att komma närmare den slutliga målpunkten.

Simuleringarna tar inte hänsyn till GPS-signalen utan antar att odometrin är exakt vilket gör att resultatet blir bättre. I simuleringarna används PID-regulatorn. Simuleringarna använder sig av samma indata från gräsklipparnas inre styre, såsom vinkelhastighet och vinkel, som den gör i praktiken vilket innebär att simuleringarna i hög utsträckning efterliknar körning i verkligheten med optimal positionering, dock med brus.

Gräsklipparen av modell 550 EPOS innehar ett bättre GPS-system med 2-3 centimeters felmarginal. Ett problem som dock inte löstes med Husqvarna 550 EPOS var att gräsklipparens odometri inte lyckades erhållas. Förhoppningen var att RTK från basstationen skulle vara tillräcklig. Det fungerar att ge kommandon till gräsklipparen och få gräsklipparen att åka. Informationen som skickas tillbaka om gräsklipparens position är dock felaktig då basstationens koordinatsystem är roterat 90° och spegelvänt. Avsnitt 4.1.2 och Figur 4.4(a) visar konsekvenserna av detta. Detta rör sig troligen om en bugg i basstationen. En översättning av koordinatsystemt implementerades. Därefter kvarstod problem vilket kan ses i Avsnitt 4.1.2 och Figuren 4.4(a). Även om försök till att lösa dessa problem gjordes fanns inte tid till det, problemen

upptäcktes när det inte var lång tid kvar av projektet. Orsakerna till felen är inte helt klara men hypotesen är att det beror på något eller flera av följande:

- Fel i den implementerade översättningen av koordinatsystem.
- Felaktigt beräknad vinkel för att ta sig till målpunkt.
- Vinkeln från IMU överensstämmer inte med den vinkeln som tar gräsklipparen till rätt position i koordinatsystemet.
- Bugg i koden, troligtvis beroende på fel när koden skulle anpassas till 550:n från 450:n.

I och med att koordinatsystemet och vinkeln inte stämmer för modell 550 EPOS kan inte gräsklipparen röra sig som planerat samt att Kalmanfilter, reglering eller andra algoritmer inte kan tillämpas vilket gör gräsklipparen, för ändamålet, obrukbar.

5.3 Reglering

När PID-regulatorn och LQR jämförs är det PID-regulatorn som slutar med minst fel vid både cirkelkörning och rektangelkörning. Det kan bero på att den redigerar felet bättre för den har en integrerande faktor som hela tiden forsätter minska felet. LQR:en kan även vara lite fel konstruerad då den är konstruerad till att ta sig till en punkt och inte utgår från en linje som den ska minska både avståndet och vinkeln till. Det kan rättas till genom att att beräkna närmaste avståndet till en punkt på linjen som ska följas och sedan observera en punkt längre fram på linjen som vinkeln ska stämma överens med. Brusfelet ökade felet mycket för LQR-kontrollen. I Figur 4.6(b) visas när felet är så stort att det inte gick att åtgärda. Det hade gått att lösa om det linjen delas upp i fler punkter än bara fyra slutpunkter.

5.4 Kamning och målning

Resultatet av kamning och målning var i hög grad beroende av resultatet av navigeringen. I och med att gräsklipparen inte kunde köras enligt önskad rutt kunde det inte testas att måla hela planen, därmed uppfylldes inte det målet. Simuleringarna visar att gräsklipparna kunde köra utefter hela fotbollsplanen men i praktiken begränsades den av precitionsfel. Servon som kontrollerar färgsprutan skulle testas med gräsklipparen av modell 550 EPOS i och med att synkroniseringen med pulsbreddsmodulering fungerade med den Raspberry pi som satt i gräsklipparen. Eftersom testerna inte gjorde med 550 som planerat kunde det inte testas huruvida gräsklipparen kunde börja och sluta måla vid rätt positioner. Testning av kammen genomfördes, istället för att köra kamningsalgoritmen, genom att fjärrstyra gräsklipparen på ett mindre område. Eftersom att mönster kunde urskiljas i gräset uppnåddes delvis delmålet för kamning men inte fullt ut eftersom gräsklipparen inte kunde köra utefter kamningsalgoritmen autonomt och skapa mönster.

6

Slutsats

Projektets syfte var att vidareutveckla en befintlig autonom gräsklippare för att göra den mer användbar på fotbollsplaner. Mer specifikt att gräsklipparen ska kunna klippa och kamma gräset i en förutbestämd rutt, samt att kunna måla linjer på fotbollsplanen. Detta syfte lyckades inte att uppfyllas fullt ut vilket visas i resultaten. Robotgräsklipparen av modell 450x kunde köras efter en rak linje, men med en varierande precision på mellan 0,1-0,5 meter. Denna precisionen är inte tillräcklig för att utföra syftet med att måla och kamma gräset. Resultatet blev begränsat till raka linjer då projektet var mycket tidskrävande och detta ansågs kritiskt för att få det andra att fungera. Eftersom IMU:n inte fungerade med tillräckligt hög precision användes GPS:en för att finna riktningen som gräsklipparen rörde sig med enligt odometrin jämfört med GPS:en. Precisionen på modell 450x var inte tillräcklig för att kunna utföra syftet. Detta var huvudanledningen till att gräsklipparen inte kunde åka rakare och till att klipparen inte kunde användas för målning och kamning av linjer. Dessutom när gräsklipparen står stilla och roterar kan den inte avgöra sin riktning vilket skulle lösas genom att klipparen aldrig stod stilla och roterade. Robotgräsklipparen av modell 550 EPOS gav inga resultat då den inte lyckades följa en rutt då dess positionen den fick från basstationen var felaktig. I och med detta kunde ingen körning efter rutt följas.

Målningskonstruktionen fungerade och linjer lyckades göras med rätt bredd på elva centimeter. Även kamningen fungerade och gav ett tillfredsställande resultat för ändamålet. Eftersom att gräsklipparen inte kan följa en rutt med önskad precision kan inte gräsklipparen utföra dessa uppgifter själv och ge ett tillräckligt bra resultat för att kunna spela fotboll på.

De slutsatser som går att dra från resultaten är att precisionen på körningen gick att förbättra med Kalmanfilter istället från att använda sig av endast odometri. Precisionen är dock inte tillräcklig för att uppfylla ändamålet. Om modell 550 EPOS hade kommit att fungera som modell 450x hade resultatet kunna bli avsevärt mycket bättre då RTK kan ge mycket bättre precision. Konstruktionen av kamningen och målningen fungerar och skulle kunna användas om gräsklipparen lyckas följa en rutt med god precision. Projektets storlek kan varit för stor då inte alla delmål uppfylldes. Att fokusera på två robotgräsklippare som båda krävde mycket tid borde nog istället lagts på en gräsklipparen.

För framtida arbeten skulle fokuset kunna ligga på att bara använda sig av modell 550 EPOS då dess precision är betydligt bättre och därmed få ett bättre resultat.

Den har mycket potential och kan troligen med lite tid få att fungera mycket bra och uppfylla målen. Ett stort fokus bör läggas på att få ut den riktning som gräsklipparen har och testa koordinatsystemen.

Litteraturförteckning

[1] wikipedia.org, ”Association football”. [Internet]. Tillgänglig: https://en.wikipedia.org/wiki/Association_football. (Hämtad 2023-01-29)

[2] umeaidrottsgala.se, ”Sveriges 5 populäraste sporter”. [Internet]. Tillgänglig: https://umeaidrottsgala.se/sveriges-5-popularaste-sporter/. (Hämtad 2023-01-30)

[3] linkoping.se, ”skötselplan fotbollsplaner”. [Internet]. Tillgänglig: https://www.linkoping.se/globalassets/uppleva-och-gora/arenor-och-idrottsanlaggnignar/forfragningsunderlag-fotbollsplaner-och-friidrottsarenaisoval/skotselplan-fotbollsplaner.pdf. (Hämtad 2023-01-29)

[4] fogis.se, ”Konstgräsplanerna växer i antal”. [Internet]. Tillgänglig: https://fogis.se/anlaggningarenor/arkiv/startsida/2013/11/konstgrasplanerna-vaxer-i-antal/. (Hämtad 2023-01-29)

[5] ivl.se, ”Mikroplast från konstgräs – frågor och svar”. [Internet]. Tillgänglig: https://www.ivl.se/press/analys-och-kommentarer/analys-och-kommentarer/2019-06-17-mikroplast-fran-konstgras---fragor-och-svar.html. (Hämtad 2023-01-29)

[6] idrottsforskning.se, ”Konstgräs = skador?”. [Internet]. Tillgänglig: https://www.idrottsforskning.se/wp-content/uploads/2014/04/Kort-idrottsforskning-nr3-2013.pdf

[7] svt.se, ”Robotar kan ta över hälften av jobben inom 20 år”. [Internet]. Tillgänglig: https://www.svt.se/nyheter/vetenskap/robotar-kan-ta-over-halften-av-jobben-inom-20-ar. (Hämtad 2023-01-29)

[8] husqvarna.se, ”Var femte villaägare klipper gräset automatiskt”. [Internet]. Tillgänglig: https://www.husqvarna.com/se/utforska-och-upptack/nyheter-och-media/svenskarna-och-deras-robotgrasklippare. (Hämtad 2023-01-31)

[9] robotnyheter.se, ”3 nya robotgräsklippare från Husqvarna”. [Internet]. Tillgänglig: http://robotnyheter.se/2013/02/06/3-nya-robotgrasklippare-fran-husqvarna/. (Hämtad 2023-01-31)

[10] docplayer.se, ”Robotgräsklippare med ruttplanering ”. [Internet]. Tillgänglig: https://docplayer.se/55869094-Robotgrasklippare-med-ruttplanering.html. (Hämtad 2023-01-29)

[11] turftank.com, ”Turf tank”. [Internet]. Tillgänglig: https://turftank.com/se/vart-foretag/. (Hämtad 2023-01-31)

[12] husqvarna.com, ”Utforska och upptäck Husqvarna EPOS”. [Internet]. Tillgänglig: https://www.husqvarna.com/se/utforska-och-upptack/husqvarna-epos/. (Hämtad 2023-01-31)

[13] svenskfotboll.se, ”Lösning -en fotbollsplan”. [Internet]. Tillgänglig: https://www.svenskfotboll.se/4967fe/globalassets/svff/dokumentdokumentblock/anlaggning/losning---1-fotbollsplan.pdf. (Hämtad 2023-01-30)

[14] turftank.com, ”idrottsklubbar och ungdomsorganisationer”. [Internet]. Tillgänglig: https://turftank.com/se/idrottsklubbar-och-ungdomsorganisationer/. (Hämtad 2023-01-31)

[15] inspekto.se, ”Så skyddar du djuren från robotgräsklipparen”. [Internet]. Tillgänglig: https://inspekto.se/sa-skyddar-du-djuren-fran-robotgrasklipparen/. (Hämtad 2023-01-29)

[16] Hermansson, H., Hansson, S. O. (2007). A Three-Party Model Tool for Ethical Risk Analysis. Risk Management, 9(3), 129–144. Tillgänglig: https://doi.org/10.1057/palgrave.rm.8250028 (hämtad 2023-04-25)

[17] svensk-fotboll.com, ”spelregler för fotboll 2019”. [Internet]. Tillgänglig: https://www.svensk-fotboll.com/_downloads/spelregler2019.pdf

[18] cision.com, ”husqvarna group”. [Internet]. Tillgänglig:https://news.cision.com/se/husqvarna-group/?q=husqvarna (Hämtad 2023-03-31)

[19] husqvarna.com, ”HUSQVARNA AUTOMOWER® 450X”. [Internet]. Tillgänglig: https://www.husqvarna.com/fi-sv/robotgrasklippare/automower-450x/. (Hämtad 2023-04-31)

[20] github.com, ”HusqvarnaResearch/hrp”. [Internet]. Tillgänglig: https://github.com/HusqvarnaResearch/hrp

[21] ros.org, ”Husqvarna Research Platform”. [Internet]. Tillgänglig: https://www.ros.org/news/2016/05/husqvarna-research-platform.html

[22] husqvarna.com, ”HUSQVARNA AUTOMOWER® 550 EPOS™”. [Internet]. Tillgänglig: https://www.husqvarna.com/se/robotgrasklippare/automower-550epos/. (Hämtad 2023-04-31)

[23] Wikipedia.org, ”Real time kinamatic”, [Internet]. tillgänglig: https://sv.wikipedia.org/wiki/Real_time_kinematic. (Hämtad 2023-05-03)

[24] pdf, ”Study the capabilities of RTK Multi GNSS under forest canopy in regions of Indonesia”, By: Andreas, Heri; Zainal Abidin, Hasanuddin; Anggreni Sarsito, Dina; Pradipta, Dhota; Wijaya, D.D.. E3S Web of Conferences , 5/21/2019, Vol. 94, pN.PAG-N.PAG, 6p. Publisher: EDP Sciences., Database: Complementary Index https://www.e3s-conferences.org/articles/e3sconf/pdf/2019/20/e3sconf_isgnss2018_01021.pdf doi:https://doi.org/10.1051/e3sconf/20199401021

[25] developer.nvidia.com, ”Getting Started with Jetson Nano 2GB Developer Kit”. [internet]. Tillgänglig: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit (hämtad 2023-04-27)

[26] commons.wikipedia.org, ”NVIDIA Jetson Nano Developer Kit”. [Internet]. Tillgänglig: https://commons.wikimedia.org/wiki/File:NVIDIA_Jetson_Nano_Developer_Kit_%2847616885631%29.jpg

[27] wikipedia.org, ”Raspberry Pi”. [Internet]. Tillgänglig:https://en.wikipedia.org/wiki/Raspberry_Pi (Hämtad 2023-03-31)

[28] kjell.com, ”raspberry pi 4”. [Internet]. Tillgänglig:https://www.kjell.com/se/kunskap/hur-funkar-det/dator/enkortsdatorer/raspberry-pi (Hämtad 2023-03-31)

[29] commons.wikipedia.org, ”Raspberry Pi 4 Model B”. [Internet]. Tillgänglig: https://commons.wikimedia.org/wiki/File:Raspberry_Pi_4_Model_B_-_Side.jpg (hämtad 2023-04-24)

[30] hackster.io, ”ROS”. [Internet]. Tillgänglig:https://www.hackster.io/shahizat/getting-started-with-ros-melodic-on-raspberry-pi-4-model-b-cbdec8. (Hämtad 2023-03-30)

[31] theconstructism.com, ”ROS”. [Internet]. Tillgänglig:https://www.theconstructsim.com/what-is-ros/. (Hämtad 2023-03-31)

[32] ros.org, ”ROS” [Internet]. Tillgänglig:http://wiki.ros.org/Topics. (Hämtad 2023-05-03)

[33] R. K. Megalingam, S. Tantravahi, H. S. Surya Kumar Tammana, N. Thokala, H. Sudarshan Rahul Puram and N. Samudrala, Robot Operating System Integrated robot control through Secure Shell(SSH),"2019 3rd International Conference on Recent Developments in Control, Automation Power Engineering (RDCAPE), Noida, India, 2019, pp. 569-573, doi: 10.1109/RDCAPE47089.2019.8979113. (Hämtad 2023-04-19)

[34] gisgeography.com, ”How GPS Receivers Work – Trilateration vs Triangulation”. [Internet]. Tillgänglig: https://gisgeography.com/trilateration-triangulation-gps/ (hämtad 2023-05-02)

[35] gisgeography.com, ”How GPS Receivers Work – Trilateration vs Triangulation”. [Internet]. Tillgänglig: https://gisgeography.com/wp-content/uploads/2016/11/Trilateration-4.png

[36] bzarg.com, ”kalman filter”. [Internet]. Tillgänglig:https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ (Hämtad 2023-03-31)

[37] automaticaddison.som, ”Extended Kalman filter (ekf) with python example”,[Internet]. Tillgänglig:https://automaticaddison.com/extended-kalman-filter-ekf-with-python-code-example/ (Hämtad 2023-04-15)

[38] commons.wikipedia.org, ”Basic concept of Kalman filtering”. [Internet]. Tillgänglig: https://commons.wikimedia.org/wiki/File:Basic_concept_of_Kalman_filtering.svg (hämtad 2023-04-24)

[39] cs.columbia.edu, ”Differential Drive Robots”. [Internet]. Tillgänglig:https://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf. (Hämtad 2023-03-28)

[40] theseus.fi, ”Differentiellstyrning”. [Internet]. Tillgänglig:https://www.theseus.fi/bitstream/handle/10024/37816/Klemets_Jonatan_Siffren_Sebastian.pdf?sequence=1. (Hämtad 2023-03-28)

[41] medium.com, ”Differentiellstyrning bild”. [Internet]. Tillgänglig:https://medium.com/manual-robotics/drives-76c2b2dac97c. (Hämtad 2023-03-28)

[42] http://users.abo.fi/, ”PID-regulator”. [Internet]. Tillgänglig:http://users.abo.fi/khaggblo/RT/RTk7.pdf. (Hämtad 2023-03-28)

[43] en.wikipdia.org, ”PID-regulator wikipedia”. [Internet]. Tillgänglig:https://en.wikipedia.org/wiki/PID_controller. (Hämtad 2023-03-28)
[44] automaticaddison.com, ”Linear Quadradic Ragulator (LQR) With Python Code Example”,[Internet]. Tillgänglig: https://automaticaddison.com/linear-quadratic-regulator-lqr-with-python-code-example/ (Hämtad 2023-04-18)
[45] automaticaddison.som, ”Combine The Extended Kalman Filter With LQR”,[Internet]. Tillgänglig: https://automaticaddison.com/combine-the-extended-kalman-filter-with-lqr/ (Hämtad 2023-05-11)
[46] Youtube.com, Christopher Lum, minut 45, ”Introduction to Linear Quadratic Regulator (LQR) Control”, [Internet], Tillgänglig: https://www.youtube.com/watch?v=wEevt2a4SKI (Hämtad 2023-04-18)
[47] circuitbread.com, ”pulsbreddmodulering”. [Internet]. Tillgänglig: https://www.circuitbread.com/ee-faq/what-is-a-pwm-signal (Hämtad 2023-04-18)
[48] gis.com, “Easting and Northing”. [Internet]. Available: http://wiki.gis.com/wiki/index.php/Easting_and_northing
[49] Husqvarna.com, ”Husqvarna EPOS referenstation”. [Internet]. Tillgänglig: https://www.husqvarna.com/se/installation-av-robotgrasklippare/epos-referensstation/ (Hämtad 2023-05-15)
[50] biltema.se, ”Färgspruta PG 18V”, [Internet] Tillgänglig: https://www.biltema.se/verktyg/multix/multix-18v/fargspruta-pg-18v-2000046804?gclid=Cj0KCQjww4-hBhCtARIsAC9gR3aoHVbXqgooGVz_urgNUuMeq_Vzq4Mk8HpafNL8-nqZWeQCHM77hWMaAsWPEALw_wcB (Hämtad 2023-03-29)
[51] Youtube.com, Gustav Malmström, ”Final test of automower with kalmanfilter 2x”. [Internet]. Tillgänglig: https://www.youtube.com/watch?v=LBpYDC6b2uc&ab_channel=GustavMalmstr%C3%B6m. (Hämtad 2023-05-15)

A

Programmeringskod

A.1 main.py

Det är denna filen som styr hela programmet.

```
#!/usr/bin/env python
import sys

sys.path.append("/home/kandidatarbete/450/src/calculations")
import rospy
import tf.transformations
import signal
import json
from geometry_msgs.msg import Twist, PoseStamped
from sensor_msgs.msg import NavSatFix
# from gps_common.msg import NavSatFix
import numpy as np
from calc_velocities import CalcVelocities
from paint import get_paint_order
from plot_data2 import plot_data
from change_goal import change_goal
from coord_sys_trans import *
from imu import *
from kalman import EKF2D
import datetime
class Drive_to:
    def __init__(self, reset_angle=True):
        self.update_freq = 10.0  # Hz but doesn't work? stuck at 10
        ↪  Hz
        self.store_data = True
        self.data = {
            "x": [],
            "y": [],
            "x_ordometry": [],
            "y_ordometry": [],
            "x_goal": [],
            "y_goal": [],
            "x_mid": [],
```

```
        "y_mid": [],
        "radius": [],
        "x_gps": [],
        "y_gps": [],
        "lat": [],
        "lon": [],
        "lat_start": [],
        "lon_start": [],
        "angle_north": [],
        "angle_north_init": [],
        "covariance": [],
        "angle_ordometry": [],
        "angle": [],
        "GPS_angle": [],
    }
    self.x = None
    self.y = None
    self.x_start = None
    self.y_start = None
    self.init_angle = None
    self.radius = None
    self.drive_in_circle = False
    self.reset_angle = reset_angle
    self.twist = Twist()
    self.lat_start = None
    self.lon_start = None
    self.covariance = None
    self.gps_covariance_factor = 0.01
    self.gps_angle_factor = 3

    self.calc_velocities = None
    self.reached_goal = False
    rospy.init_node("move_forward")
    self.pub = rospy.Publisher("/cmd_vel", Twist, queue_size=10)
    sub = rospy.Subscriber("/pose", PoseStamped,
    ↪  self.pose_callback)
    self.gps_sub = rospy.Subscriber("/GPSfix", NavSatFix,
    ↪  self.gps_callback)
    self.rate = rospy.Rate(self.update_freq)
    self.order = get_paint_order()
    self.angle_north = 0#np.pi
    self.phi = 0
    self.min_data_points = 3
    self.phi_always =
    ↪  -0.9313#-0.897701#-0.942#-0.96#-0.9783404808#-1
    ↪  #-0.9224036087
```

```
def gps_callback(self, fix):
    if self.lat_start is None and self.lon_start is None:
        self.lat_start = fix.latitude
        self.lon_start = fix.longitude
        if self.lat_start <= 5:
            self.lat_start = 57.68727
            self.lon_start = 11.97958
        self.data["lat_start"].append(self.lat_start)
        self.data["lon_start"].append(self.lon_start)
        self.data["angle_north_init"].append(0)
        self.data["lat"].append(self.lat_start)
        self.data["lon"].append(self.lon_start)
    lat = fix.latitude
    lon = fix.longitude
    x_gps, y_gps = convert_lat_lon_to_utm(lat, lon)
    # y_gps = (y_gps - self.data["y_gps"][0])*(-1) +
    ↪  self.data["y_gps"][0] #TODO test this
    if len(self.data["x_gps"]) == self.min_data_points:
        k1,m1 = best_fit_line(self.data["x_gps"],
        ↪  self.data["y_gps"])
        k2, m2 = best_fit_line(self.data["x"], self.data["y"])
        self.angle = angle_between_lines(k1, m1, k2, m2)
        self.angle_correct =
        ↪  angle_between_points(self.data["x_gps"][0],
        ↪  self.data["y_gps"][0], self.data["x_gps"][-1],
        ↪  self.data["y_gps"][-1])
        self.phi = closest_angle(self.angle, self.angle_correct)
        self.phi = self.angle_correct
        self.phi = self.phi_always
        self.data["k1"] = k1
        self.data["k2"] = k2
        self.data["m1"] = m1
        self.data["m2"] = m2
        initial_state = np.array([self.x, self.y, 0])
        initial_input = np.array([0.4, 0])
        initial_covariance = np.eye(3) * 0.1
        process_noise = np.eye(3) * 0.001
        self.ekf = EKF2D(initial_state, initial_input,
        ↪  initial_covariance, process_noise)
    elif len(self.data["x_gps"]) < self.min_data_points:
        self.calc_velocities.max_vel_lin = 0.1
    else:
        self.calc_velocities.max_vel_lin = 0.4

    gps_covariance = fix.position_covariance[0]
```

```
            self.data["covariance"].append(gps_covariance)
            gps_angle = None
            if self.phi != 0:
                # self.phi = 0
                x_gps, y_gps = rotate_point(x_gps, y_gps, self.x_start,
                ↪  self.y_start, -self.phi)
                # y_gps = (y_gps - self.data["y_gps"][0])*(-1) +
                ↪  self.data["y_gps"][0]
                if len(self.data["x_gps"]) > self.min_data_points:
                    gps_angle = np.arctan2(y_gps -
                    ↪  self.data["y_gps"][-1], x_gps -
                    ↪  self.data["x_gps"][-1])
                else:
                    gps_angle = 0
                gps_covariance = int(gps_covariance) *
                ↪  self.gps_covariance_factor
                gps_angle_covariance = gps_covariance *
                ↪  self.gps_angle_factor
                self.ekf.update_gps(x_gps, y_gps,gps_angle,
                ↪  gps_covariance, gps_angle_covariance)
            self.data["GPS_angle"] = gps_angle
            self.data["x_gps"].append(x_gps)
            self.data["y_gps"].append(y_gps)
            self.data["lat"].append(lat)
            self.data["lon"].append(lon)
            self.data["angle_north"].append(self.angle_north)
            self.calc_velocities.log_message()
           print("x_gps", x_gps)
            print("y_gps", y_gps)

    def drive(self):
        signal.signal(
            signal.SIGINT, self.ctrlc_shutdown
        )  # shuts down when ctrl+c is pressed
        while not rospy.is_shutdown() and not self.reached_goal:
            self.pub.publish(self.twist)
            self.rate.sleep()
        self.stop()

    def stop(self):
        self.twist.linear.x = 0.0
        self.twist.angular.z = 0.0
        self.pub.publish(self.twist)
        timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
        timestamp = int(timestamp.replace("-", ""))
        #make filename variable with python 2.7
```

```
        filename = "../data/{}.json".format(timestamp) #check if
        ↪  this works
        print("phi", self.phi)
        # filename = f"../data/{timestamp}.json"
        if self.store_data:
            with open(filename, "w") as json_file:
                json.dump(self.data, json_file)
            plot_data(GPS = True, filename = filename)
        rospy.loginfo(
            "Automower has moved to position x=%s, y=%s",
            round(self.x, 2),
            round(self.y, 2),
        )

    def pose_callback(self, pose):
        if len(self.data["lat"]) > 0:
            z_dir = pose.pose.orientation.z
            w_dir = pose.pose.orientation.w
            current_ang =
            ↪  tf.transformations.euler_from_quaternion([0, 0,
            ↪  z_dir, w_dir])[2]
            if (
                self.reset_angle
                and self.init_angle is None
                and self.x_start is None
                and self.y_start is None
            ):
                self.init_angle = current_ang
                self.x_start_automower= pose.pose.position.x
                self.y_start_automower = pose.pose.position.y
                print("init lat" , self.data["lat"][0], "init lon",
                ↪  self.data["lon"][0])
                self.x_start, self.y_start =
                ↪  convert_lat_lon_to_utm(self.data["lat"][0],
                ↪  self.data["lon"][0])
                self.calc_velocities =
                ↪  CalcVelocities(self.update_freq)
                change_goal(self)  # sets initial goal/

            x_automower = pose.pose.position.x
            y_automower = pose.pose.position.y

            self.x, self.y = convert_automower_to_utm(self,
            ↪  x_automower, y_automower)
            self.data["x_ordometry"].append(self.x)
            self.data["y_ordometry"].append(self.y)
```

```
self.data["angle_ordometry"].append(current_ang)
if self.phi != 0:
    # delta_x = self.x - self.ekf.get_state()[0]
    # delta_y = self.y - self.ekf.get_state()[1]
    delta_x = self.data["x_ordometry"][-1] -
    ↪  self.data["x_ordometry"][-2]
    delta_y = self.data["y_ordometry"][-1] -
    ↪  self.data["y_ordometry"][-2]
    # delta_ang = self.data["angle"][-1] -
    ↪  self.data["angle"][-2]
    # delta_ang = np.arctan2(delta_y, delta_x)
    delta_ang = self.data["angle_ordometry"][-1] -
    ↪  self.data["angle_ordometry"][-2]
    # measurement_angle = np.arctan2(delta_y, delta_x) -
    ↪  self.ekf.get_state()[2]
    dt = 1 / self.update_freq
    self.ekf.predict(delta_x, delta_y,delta_ang, dt)
    self.x,self.y, current_ang = self.ekf.get_state()
    print(current_ang)
    #normailize angle
    # while current_ang2 > np.pi:
    #     current_ang2 -= 2 * np.pi
    # while current_ang2 < -np.pi:
    #     current_ang2 += 2 * np.pi
    # self.data["angle"].append(current_ang2)
self.data["x"].append(self.x)
self.data["y"].append(self.y)
lin_vel, ang_vel =
↪  self.calc_velocities.calc_vel(current_ang, self.x,
↪  self.y)
# if lin_vel != 0:
#     current_ang = np.arctan2(self.data["y"][-1] -
↪  self.data["y"][-2], self.data["x"][-1] -
↪  self.data["x"][-2])
#     lin_vel, ang_vel =
↪  self.calc_velocities.calc_vel(current_ang, self.x,
↪  self.y)
self.twist.linear.x = lin_vel
self.twist.angular.z = ang_vel
try:
    print("angle kalman", current_ang, "ordometry_angle",
    ↪  self.data["angle_ordometry"][-1], "gps_angle",
    ↪  self.data["GPS_angle"][-1])
except:
    pass
# if close to goal, cahnge goal
```

```
            if lin_vel == 0.0 and ang_vel == 0.0:
                change_goal(self)

    def ctrlc_shutdown(self, sig, frame):
        self.stop()
        rospy.signal_shutdown("User interrupted by ctrl+c")


if __name__ == "__main__":
    drive_to = Drive_to()
    drive_to.drive()
```

A.2 calc_velocities

Det är denna filen som hjälper main.py att beräkna vilka hastigheter klipparen ska ha för att komma fram till målet.

```
import numpy as np
# from lqr import LQR

class CalcVelocities:
    #def __init__(self, Kp_circle=-206.5510399,
    ↪  Ki_circle=-18.7532926, Kd_circle=-14.2643917,
    ↪  Kp90_circle=-45.9726824):
    def __init__(self, Kp_circle=-218.26760943389627, Ki_circle=
    ↪  -12.686635740990944, Kd_circle=-19.457658986144885,
    ↪  Kp90_circle=-42.58772926624732):
        self.tol_lin = 0.002# tolerance in meter
        self.tol_ang = 7. * np.pi / 180
        self.min_tol_ang = 0.3 * np.pi / 180.  # to avoid
        ↪  calculations error
        self.max_vel_lin = 0.2
        self.max_vel_ang = 0.8
        self.Kp_l = 0.9
        self.Kp_a = 0.7
        self.Kp90_circle = Kp90_circle
        self.Kp_circle = Kp_circle
        self.Ki_circle = Ki_circle
        self.Kd_circle = Kd_circle
        self.error_radius = 0
        self.error_radius_sum = 0
        self.error_radius_prev = 0
        self.x_goal = None  # global coordinates
        self.y_goal = None
        update_freq = 10.
        self.dt = 1.0 / update_freq
        self.current_ang = None
```

```
        self.error_lin = None
        self.vel_lin = None
        self.x = None
        self.y = None
        self.paint_circle = False
        self.x_mid = None
        self.y_mid = None
        self.radius = None
        self.square_error_radius = 0
        self.times_above_tol_ang = 0
        self.has_moved = False
        self.dir = 1
        self.abs_error_radius = 0
        # self.lqr=LQR([self.x, self.y, self.current_ang], [0.0])
        ↪  #fixa sen
    def set_circle_params(self, radius, x_mid, y_mid, dir):
        self.radius = radius
        self.x_mid = x_mid
        self.y_mid = y_mid
        if dir == "negative":
            self.dir = np.pi
            self.Kp_circle = np.abs(self.Kp_circle) *-1
            self.Ki_circle = np.abs(self.Ki_circle)*-1
            self.Kd_circle = np.abs(self.Kd_circle)*-1
            self.Kp90_circle = np.abs(self.Kp90_circle)*-1
        else:
            self.dir = 0
            self.Kp_circle = np.abs(self.Kp_circle)
            self.Ki_circle = np.abs(self.Ki_circle)
            self.Kd_circle = np.abs(self.Kd_circle)
            self.Kp90_circle = -np.abs(self.Kp90_circle)
        self.paint_circle = True
        self.max_vel_lin = 0.2
        if self.radius > 5.: #by test, kp circle is too high for big
        ↪  circles
            self.Kp_circle = self.Kp_circle/2
        self.tol_ang = 4. * np.pi/180

    def not_in_circle(self):
        self.radius = None
        self.x_mid = None
        self.y_mid = None
        self.paint_circle = False
        self.max_vel_lin = 0.2
        self.tol_ang = 7. * np.pi/180
```

```
def calc_radius_velocities(self):
    current_radius = np.sqrt(
        (self.x - self.x_mid) ** 2 + (self.y - self.y_mid) ** 2
    )
    self.error_radius = self.radius - current_radius
    self.error_radius_sum += self.error_radius * self.dt
    self.square_error_radius += self.error_radius**2
    self.abs_error_radius = np.abs(self.error_radius)
    self.error_radius_deriv = (
        self.error_radius - self.error_radius_prev
    ) / self.dt
    self.error_radius_prev = self.error_radius
    self.goal_ang = self._goal_angle_circle(
        self.x_mid, self.y_mid, self.x, self.y
    )
    self.error_ang = self.current_ang - self.goal_ang
    self.error_ang = self._normalize_angle(self.error_ang)

    self.vel_lin = self.max_vel_lin
    if np.abs(self.error_ang) < self.tol_ang:
        self.vel_ang = -(
            self.Kp_circle * self.error_radius * self.dt
            + self.Ki_circle * self.error_radius_sum
            + self.Kd_circle * self.error_radius_deriv
        ) / self.radius
    else:
        self.vel_ang = self.Kp90_circle * self.error_ang *
        ↪  self.dt / self.radius
        self.vel_lin = 0

def calc_line_velocities(self):
    self.goal_ang = self._goal_angle_line()
    self.error_ang = self.goal_ang - self.current_ang
    self.error_ang = (self.error_ang + np.pi) % (2 * np.pi) -
    ↪  np.pi
    if np.abs(self.error_ang) < self.min_tol_ang:  # to avoid
    ↪  calculatins error
        self.error_ang = 0
    self.vel_lin = self.Kp_l * self.error_lin  # slow down when
    ↪  close to goal
    self.vel_ang = self.Kp_a * self.error_ang
    self.abs_error_radius = np.abs(self.error_lin)

def calc_vel(self, current_ang, x, y):
    self.current_ang = self._normalize_angle(current_ang)
```

```
        self.x = x
        self.y = y
        self.goal_ang = self._goal_angle_line()

        self.error_lin = np.sqrt(
            (self.x_goal - self.x) ** 2 + (self.y_goal - self.y) ** 2
        )
        if self.paint_circle:
            # self.goal_ang = self._goal_angle_line()
            # desired_state = [self.x_goal, self.y_goal,
            ↪  self.goal_ang]
            # u_star = self.lqr.lqr(desired_state, self.dt)
            # print(u_star, "u_star")
            self.calc_radius_velocities()
        else:
            self.calc_line_velocities()

        # self.vel_lin,self.vel_ang = self.lqr.lqr_position(self.x,
        ↪  self.y, self.current_ang, self.x_goal, self.y_goal,
        ↪  self.goal_ang, 0)
        self.vel_lin = np.clip(self.vel_lin, -self.max_vel_lin,
        ↪  self.max_vel_lin)
        self.vel_ang = np.clip(self.vel_ang, -self.max_vel_ang,
        ↪  self.max_vel_ang)
        if self._has_reached_goal():
            self.vel_lin = 0
            self.vel_ang = 0
        if np.abs(self.error_ang) > self.tol_ang:
            self.vel_lin = 0
            if self.has_moved:
                self.times_above_tol_ang += 1
        else:
            self.has_moved = True
        return self.vel_lin, self.vel_ang

    def get_sqaure_error_radius(self):
        return self.square_error_radius

    def _circle_vel_ang(self):
        # Calculate the theoretical angular velocity based on the
        ↪  desired linear velocity and radius
        return self.max_vel_lin / self.radius  # TODO check if this
        ↪  is correct

    def _goal_angle_line(self):
```

```
        return np.arctan2(self.y_goal - self.y, self.x_goal -
        ↪  self.x)

    def _goal_angle_circle(self, x_mid, y_mid, x, y):
        # Calculate the vector from the center of the circle to the
        ↪  robot's position
        V_x = x - x_mid
        V_y = y - y_mid
        # Normalize the vector
        V_mag = np.sqrt(V_x**2 + V_y**2)
        V_x_norm = V_x / V_mag
        V_y_norm = V_y / V_mag
        # Calculate the tangent vector
        T_x = -V_y_norm
        T_y = V_x_norm
        return self._normalize_angle(self.dir + np.arctan2(T_y,
        ↪  T_x))

    def _normalize_angle(self, angle):
        while angle > np.pi:
            angle -= 2 * np.pi
        while angle < -np.pi:
            angle += 2 * np.pi
        return angle

    def _has_reached_goal(self):
        return self.error_lin < self.tol_lin

    def log_message(self):
        print(
            "error_lin: ",
            round(self.error_lin, 2),
            "vel_lin: ",
            round(self.vel_lin, 2),
        )
    def set_goal_coords(self, x_goal, y_goal):
        self.x_goal = x_goal
        self.y_goal = y_goal
```

A.3 kalman.py

Det är denna filen som tar in data från odometri och GPS och beräknar den mest troliga positionen som klippare har.

```
import numpy as np
import matplotlib
import datetime
```

```
class EKF2D:
    def __init__(self, initial_state, initial_input,
    ↪  initial_covariance, process_noise):
        self.state = initial_state   #state tre varden: x y theta
        self.input= initial_input  # tva varden v  omega
        self.P_k = initial_covariance
        self.Q_k = process_noise    # state model noise, cov matrix
        self.measurement_noise = 0#measurement_noise
        self.theta = self.state[2]

    def get_B(self, theta, dt):
         B = np.array([[np.cos(theta)*dt, 0],
                       [np.sin(theta)*dt, 0],
                        [0, dt]])
         return B
    def predict(self, delta_x, delta_y,delta_ang, dt):

        A = np.array([[1,0,0],
                      [0,1,0],
                      [0,0,1]])
        B = self.get_B(self.theta, dt)

        u = np.array([delta_x, delta_y, delta_ang])
        self.state = np.dot(A, self.state) + u

        self.P_k = np.dot(A, np.dot(self.P_k, A.T)) + self.Q_k
    def get_state(self):
        return self.state

    def update_gps(self, gps_x, gps_y,yaw, measurement_noise,
    ↪  angle_noise):
        self.measurement_noise = np.array([
            [measurement_noise, 0, 0 ],
            [0, measurement_noise, 0 ],
            [0, 0, angle_noise]
        ])
        return self.update(gps_x, gps_y,yaw, self.measurement_noise)

    def update(self, gps_x, gps_y, yaw, position_covariance):
        self.R_k = position_covariance
        H_k = np.array([
            [1, 0, 0 ],
            [0, 1, 0 ],
            [0, 0, 1]
        ])
```

```
        y_k = np.array([gps_x, gps_y, yaw]) - np.dot(H_k, self.state)
        ↪  # Calculate the difference between the actual sensor
        ↪  measurements

        S_k = np.dot(H_k, np.dot(self.P_k, H_k.T)) + self.R_k
        ↪  # R_k=position_covariance
        K_k = np.dot(self.P_k, np.dot(H_k.T, np.linalg.inv(S_k)))

        self.state = self.state + np.dot(K_k, y_k)
        self.P_k = self.P_k - np.dot(K_k, np.dot(H_k, self.P_k))
```

A.4 coord_sys_trans.py

Det är denna filen som gör koordinattransformationerna samt roterar GPS till ordometri.

```
# from pyproj import Proj, Transformer
import numpy as np
import utm
import math
def convert_to_xy(lat, lon, lat_start, lon_start):
    x = (lat - lat_start) * 111139
    y = (lon - lon_start) * 111139
    return x, y

def change_coord_sys(
    x_goal_prim, y_goal_prim, x_start, y_start, init_angle
):  # automowers relative coordinates => global coordinates
    x_goal = (
        x_start
        + x_goal_prim * np.cos(init_angle)
        - y_goal_prim * np.sin(init_angle)
    )
    y_goal = (
        y_start
        + x_goal_prim * np.sin(init_angle)
        + y_goal_prim * np.cos(init_angle)
    )
    return x_goal, y_goal  # automowers global coordinates

def inverse_change_coord_sys(x_goal, y_goal, x_start, y_start,
↪  init_angle): #TODO test this function. Then we can use it in
↪  plot
    x_goal_prim = (
        (x_goal - x_start) * np.cos(init_angle)
        + (y_goal - y_start) * np.sin(init_angle)
    )
```

```
    y_goal_prim = (
        -(x_goal - x_start) * np.sin(init_angle)
        + (y_goal - y_start) * np.cos(init_angle)
    )
    return x_goal_prim, y_goal_prim  # automowers relative
    ↪  coordinates


def convert_automower_to_utm(self, x_automower, y_automower):
    x_utm = self.x_start +  x_automower*math.cos(self.angle_north) -
    ↪  y_automower*math.sin(self.angle_north)
    y_utm = self.y_start + y_automower*math.cos(self.angle_north) +
    ↪  x_automower*math.sin(self.angle_north)
    return x_utm, y_utm

def convert_lat_lon_to_utm(lat, lon):
    east, north, number, letter = utm.from_latlon(lat, lon)
    return north, -east
    # return east, north

def get_angle_north(x_utm1, y_utm1, x_utm2, y_utm2):
    #x is north and y is west
    angle_north = math.atan2(x_utm2 - x_utm1, y_utm2 - y_utm1)
    return angle_north

# def get_angle_gps_ordometry(x_gps1, y_gps1, x_gps2, y_gps2,
#     #x is north and y is west
#     angle_gps = math.atan2(x_gps2 - x_gps1, y_gps2 - y_gps1)
#     return angle_gp

def rotate_point(x, y, x0, y0, phi):
    x_translated = x - x0
    y_translated = y - y0

    x_rotated = x_translated * math.cos(phi) - y_translated *
    ↪  math.sin(phi)
    y_rotated = x_translated * math.sin(phi) + y_translated *
    ↪  math.cos(phi)

    x_new = x_rotated + x0
    y_new = y_rotated + y0

    return x_new, y_new
```

```
def best_fit_line(x_gps, y_gps):
    x = np.array(x_gps)
    y = np.array(y_gps)

    n = len(x)
    sum_x = np.sum(x)
    sum_y = np.sum(y)
    sum_x_squared = np.sum(x**2)
    sum_xy = np.sum(x*y)

    k = (n * sum_xy - sum_x * sum_y) / (n * sum_x_squared - sum_x**2)
    m = (sum_y - k * sum_x) / n

    return k, m

def angle_between_lines(k1, m1, k2, m2):
    numerator = k2 - k1
    denominator = 1 + k1 * k2
    angle = np.arctan2(numerator, denominator)
    return angle

def angle_between_points(xg1, yg1, xg2, yg2):
    cosphi = (xg2-xg1)/np.sqrt((xg2-xg1)**2 + (yg2-yg1)**2)
    return np.arccos(cosphi)

def closest_angle(angle, correct_ang):
    angle1 = angle
    angle2 = np.pi - angle
    angle3 = -angle
    angle4 = -np.pi - angle
    angles = [angle1, angle2, angle3, angle4]
    closest = min(angles, key=lambda x: abs(x - correct_ang))
    return closest
```

A.5 paint.py

Det är denna filen som beskriver för klipparen hur den ska åka för att måla linjerna.

```
import sys
import json
import os

sys.path.append("/home/kandidatarbete/450/src/userInput")
current_file = __file__
parent_dir = os.path.dirname(current_file)
# grand_parent_dir = os.path.dirname(parent_dir)
sibling_dir = os.path.join(parent_dir, "userInput")
```

```
child_file = os.path.join(sibling_dir, "paintDimension.json")
child_file_rel = os.path.relpath(child_file, current_file)

with open(child_file_rel) as paintDimensions:
          user_input = json.load(paintDimensions)
width = round(float(user_input["shortside"]), 4)
lenght = round(float(user_input["longside"]), 4)
penalty_area_width = round(float(user_input["penaltyAreaWidth"]), 4)
penalty_area_height = round(float(user_input["penaltyAreaHeight"]),
↪  4)
goal_box_height = round(float(user_input["goalBoxHeight"]), 4)
goal_box_width = round(float(user_input["goalBoxWidth"]), 4)
penalty_dot_length = round(float(user_input["penaltyDotLength"]), 4)
center_circle_diameter =
↪  round(float(user_input["centerCircleDiameter"]), 4)
corner_arc_radius = round(float(user_input["cornerArcRadius"]), 4)
penalty_arc_radius = round(float(user_input["penaltyArcRadius"]), 4)
num_comb_lines = round(float(user_input["numCombLines"]), 4)

#after_end = turn270flag(width, 0, 'x') + [line1] + turn270()
# This file can only be exceuted from the calculations directory or
↪  by rosrun am_driver main.py
def get_paint_order():
    with open(child_file_rel) as paintDimensions:
        user_input = json.load(paintDimensions)
    width = round(float(user_input["shortside"]), 4)
    lenght = round(float(user_input["longside"]), 4)
    penalty_area_width = round(float(user_input["penaltyAreaWidth"]),
    ↪  4)
    penalty_area_height =
    ↪  round(float(user_input["penaltyAreaHeight"]), 4)
    goal_box_height = round(float(user_input["goalBoxHeight"]), 4)
    goal_box_width = round(float(user_input["goalBoxWidth"]), 4)
    penalty_dot_length = round(float(user_input["penaltyDotLength"]),
    ↪  4)
    center_circle_diameter =
    ↪  round(float(user_input["centerCircleDiameter"]), 4)
    corner_arc_radius = round(float(user_input["cornerArcRadius"]),
    ↪  4)
    penalty_arc_radius = round(float(user_input["penaltyArcRadius"]),
    ↪  4)
    num_comb_lines = round(float(user_input["numCombLines"]), 4)

    shortside_down = {"end": (width, 0),"after_end":
    ↪  turn270flag(width,0,'x'), "type": "line"}
```

```
longside_right = {"end": (width, lenght), "after_end":
↪  turn270flag(width, lenght,'y'),  "type": "line"}
shortside_up = {"end": (0, lenght), "after_end": turn270flag(0,
↪  lenght,'-x'), "type": "line"}
longside_left = {"end": (0, 0), "after_end": turn270flag(0,
↪  0,'-y'),"type": "line"}
line_to_penalty_area_down_1 = {"end": (width / 2 -
↪  penalty_area_width / 2, 0),
                               "after_end": turn270neg(width / 2
                               ↪  - penalty_area_width / 2,
                               ↪  0,'x'),
                               "type": "line"}
penalty_area_down_1 = {
    "end": (width / 2 - penalty_area_width / 2,
    ↪  penalty_area_height),
    "after_end": turn270pos(width / 2 - penalty_area_width / 2,
    ↪  penalty_area_height,'y'),
    "type": "line"}
penalty_area_down_2 = {
    "end": (width / 2 + penalty_area_width / 2,
    ↪  penalty_area_height),
    "after_end": turn270pos(width / 2 + penalty_area_width / 2,
    ↪  penalty_area_height, 'x'),
    "type": "line",}
penalty_area_down_3 = {
    "end": (width / 2 + penalty_area_width / 2, 0),
    "after_end": turn270pos(width / 2 + penalty_area_width / 2,
    ↪  0,'-y'),
    "type": "line",
}
line_to_goal_box_down_1 = {
    "end": (width / 2 + goal_box_width / 2, 0),
    "after_end": turn270pos(width / 2 + goal_box_width / 2, 0,
    ↪  '-x'),
    "type": "line"}
goal_box_down_1 = {
    "end": (width / 2 + goal_box_width / 2, goal_box_height),
    "after_end": turn270neg(width / 2 + goal_box_width / 2,
    ↪  goal_box_height,'y'),
    "type": "line",
}
goal_box_down_2 = {
    "end": (width / 2 - goal_box_width / 2, goal_box_height),
    "after_end": turn270neg(width / 2 - goal_box_width / 2,
    ↪  goal_box_height,'-x'),
    "type": "line",
```

```
}
goal_box_down_3 = {
    "end": (width / 2 - goal_box_width / 2, 0),
    "after_end": turn270neg(width/2 - goal_box_width / 2,
    ↪  0,'-y'),
    "type": "line",
}
line_to_penalty_arc_down1 = {
    "end": (width / 2 -  penalty_arc_radius, 0),
    "after_end": turn270neg(width / 2 -  penalty_arc_radius,
    ↪  0,'x'),
    "type": "line"}
line_to_penalty_arc_down2 = {
    "end": (width / 2 -  penalty_arc_radius, penalty_dot_length),
    "type": "line"}
penalty_arc_down = {
    "end": (width / 2 +  penalty_arc_radius, penalty_dot_length),
    "center": (width / 2, penalty_dot_length),
    "radius": penalty_arc_radius,
    "type": "circle",
    "direction": "negative"}
line_to_penalty_dot_down = {"end": (width / 2,
↪  penalty_dot_length), "type": "line"}
line_after_dot = {
    "end": (0, penalty_dot_length),
    "after_end": turn270pos(0, penalty_dot_length,'-x'),
    "type": "line"}
line_to_midline = {
    "end": (0, lenght/2),
    "after_end": turn270pos(0, lenght/2,'y'),
    "type": "line"}
midline = {
    "end": (width, lenght / 2),
    "after_end": turn180(width, lenght / 2),
    "type": "line"}
line_to_mid_circle = {
    "end": (width/2 + center_circle_diameter / 2, lenght / 2),
    "after_end": turn270pos(width/2 + center_circle_diameter / 2,
    ↪  lenght / 2,'-x'),
    "type": "line" }
mid_circle1 = {
    "end": (width/2 - center_circle_diameter/2, lenght / 2),
    "center": (width/2, lenght/2),
    "radius": center_circle_diameter / 2,
    "type": "circle",
    "direction": "positive"}
```

```
mid_circle2 = {
    "end": (width/2 + center_circle_diameter/2, lenght / 2),
    "center": (width/2, lenght/2),
    "radius": center_circle_diameter / 2,
    "type": "circle",
    "direction": "positive"}
line_after_mid_circle = {
    "end": (width / 2 + penalty_area_width / 2, lenght / 2),
    "after_end": turn270neg(width / 2 + penalty_area_width / 2,
    ↪  lenght / 2,'x'),
    "type": "line"}
line_to_penalty_area_up = { "end": (width/2 +
↪  penalty_area_width/2, lenght), "type": "line" }
penalty_area_up_1 = {
    "end": (width/2 + penalty_area_width/2, lenght),
    "after_end": turn270neg(width/2 + penalty_area_width/2,
    ↪  lenght,'y'),
    "type": "line"}
line_to_goal_box_up1 = {
    "end": (width / 2 + goal_box_width / 2, lenght),
    "after_end": turn270neg(width / 2 + goal_box_width / 2,
    ↪  lenght,'-x'),
    "type": "line" }
goal_box_up_1 = {
    "end": (width / 2 + goal_box_width / 2, lenght -
    ↪  goal_box_height),
    "after_end": turn270pos(width / 2 + goal_box_width / 2,
    ↪  lenght - goal_box_height,'-y'),
    "type": "line",}
goal_box_up_2 = {
    "end": (width / 2 - goal_box_width / 2, lenght -
    ↪  goal_box_height),
    "after_end": turn270pos(width / 2 - goal_box_width / 2,
    ↪  lenght - goal_box_height,'-x'),
    "type": "line",}
goal_box_up_3 = {
    "end": (width / 2 - goal_box_width / 2, lenght),
    "after_end": turn270neg(width / 2 - goal_box_width / 2,
    ↪  lenght,'y'),
    "type": "line",}
line_to_penalty_area_up2 = {
    "end": (width / 2 - penalty_area_width / 2, lenght),
    "after_end": turn270neg(width / 2 - penalty_area_width / 2,
    ↪  lenght,'-x'),
    "type": "line" }
penalty_area_up_2 = {
```

```
        "end": (width / 2 - penalty_area_width / 2, lenght -
        ↪  penalty_area_height),
        "after_end": turn270neg(width / 2 - penalty_area_width / 2,
        ↪  lenght - penalty_area_height,'-y'),
        "type": "line"}
    penalty_area_up_3 = {
        "end": (width / 2 + penalty_area_width / 2, lenght -
        ↪  penalty_area_height),
        "after_end": turn270neg(width / 2 + penalty_area_width / 2,
        ↪  lenght - penalty_area_height,'x'),
        "type": "line"}
    line_to_penalty_arc_up1 = {
        "end": (width / 2 +  penalty_area_width / 2, lenght -
        ↪  penalty_dot_length),
        "after_end": turn270neg(width / 2 +  penalty_area_width/2,
        ↪  lenght - penalty_dot_length,'y'),
        "type": "line"}
    line_to_penalty_arc_up2 = {
        "end": (width / 2 +  penalty_arc_radius, lenght -
        ↪  penalty_dot_length),
        "after_end": turn270neg(width / 2 +  penalty_arc_radius,
        ↪  lenght - penalty_dot_length,'-x'),
        "type": "line"}
    penalty_arc_up = {
        "end": (width / 2 -  penalty_arc_radius, lenght -
        ↪  penalty_dot_length),
        "center": (width / 2, lenght - penalty_dot_length),
        "radius": penalty_arc_radius,
        "type": "circle",
        "direction": "negative"}

    line_to_penalty_dot_up = {"end": (width / 2, lenght -
    ↪  penalty_dot_length), "type": "line"}

    midline_dot = {"start": (0, lenght/2), "end": (width/2, lenght /
    ↪  2), "type": "line"}

    test_circle = {
        "end": (1, 1),
        "center": (1, 0),
        "radius": 1,
        "type": "circle",
        "direction": "negative",
    }
```

```
    paint_order = [
        shortside_down,
        longside_right,
        shortside_up,
        longside_left,
        line_to_penalty_area_down_1,
        penalty_area_down_1,
        penalty_area_down_2,
        penalty_area_down_3,
        line_to_goal_box_down_1,
        goal_box_down_1,
        goal_box_down_2,
        goal_box_down_3,
        line_to_penalty_arc_down1,
        line_to_penalty_arc_down2,
        penalty_arc_down,
        line_to_penalty_dot_down,
        line_after_dot,
        line_to_midline,
        midline,
        line_to_mid_circle,
        mid_circle1,
        mid_circle2,
        line_after_mid_circle,
        line_to_penalty_area_up,
        penalty_area_up_1,
        line_to_goal_box_up1,
        goal_box_up_1,
        goal_box_up_2,
        goal_box_up_3,
        line_to_penalty_area_up2,
        penalty_area_up_2,
        penalty_area_up_3,
        line_to_penalty_arc_up1,
        line_to_penalty_arc_up2,
        penalty_arc_up,
        line_to_penalty_dot_up,]

    print("paint_order", paint_order)
    return paint_order
def turn270neg(x, y, travel_dir):
    if travel_dir == 'x':
        x1 = x+1
        y1 = y
        x2 = x
        y2 = y-1
```

```
        x3 = x
        y3 = y
        xcenter = x+1
        ycenter = y-1
    elif travel_dir == 'y':
        x1 = x
        y1 = y+1
        x2 = x+1
        y2 = y
        x3 = x
        y3 = y
        xcenter = x+1
        ycenter = y+1
    elif travel_dir == '-x':
        x1 = x-1
        y1 = y
        x2 = x
        y2 = y+1
        x3 = x
        y3 = y
        xcenter = x-1
        ycenter = y+1
    elif travel_dir == '-y':
        x1 = x
        y1 = y-1
        x2 = x-1
        y2 = y
        x3 = x
        y3 = y
        xcenter = x-1
        ycenter = y-1

    return  [{
        "start": (x, y),
        "end": (x1, y1),
        "type": "line",
    },
    {
        "start": (x1, y1),
        "end": (x2,y2),
        "center": (xcenter,ycenter),
        "radius": 1,
        "type": "circle",
        "direction": "negative"
        },
        {
```

```
        "start": (x2,y2),
        "end":(x3,y3),
        "type": "line"
        },]
def turn270pos(x, y, travel_dir):
    if travel_dir == 'x':
        x1 = x+1
        y1 = y
        x2 = x
        y2 = y+1
        x3 = x
        y3 = y
        xcenter = x+1
        ycenter = y+1
    elif travel_dir == 'y':
        x1 = x
        y1 = y+1
        x2 = x-1
        y2 = y
        x3 = x
        y3 = y
        xcenter = x-1
        ycenter = y+1
    elif travel_dir == '-x':
        x1 = x-1
        y1 = y
        x2 = x
        y2 = y-1
        x3 = x
        y3 = y
        xcenter = x-1
        ycenter = y-1
    elif travel_dir == '-y':
        x1 = x
        y1 = y-1
        x2 = x+1
        y2 = y
        x3 = x
        y3 = y
        xcenter = x+1
        ycenter = y-1
    return  [{
        "start": (x, y),
        "end": (x1, y1),
        "type": "line",
    },
```

```
    {
        "start": (x1, y1),
        "end": (x2, y2),
        "center": (xcenter,ycenter),
        "radius": 1,
        "type": "circle",
        "direction": "positive"
        },
        {
        "start": (x2,x2),
        "end":(x3,y3),
        "type": "line"
        },]

def turn270flag(x, y, travel_dir):
    if travel_dir == 'x':
        x1 = x-1
        y1 = y-1
        x2 = x-1
        y2 = y
        x3 = x
        y3 = y+1
        x4 = x+1
        y4 = y+1

        x5 = x
        y5 = y-1
        x6 = x+1
        y6 = y
    elif travel_dir == '-x':
        x1 = x+1
        y1 = y+1
        x2 = x+1
        y2 = y
        x3 = x
        y3 = y-1
        x4 = x-1
        y4 = y-1

        x5 = x
        y5 = y+1
        x6 = x-1
        y6 = y
    elif travel_dir == 'y':
        x1 = x+1
        y1 = y-1
```

```
        x2 = x
        y2 = y-1
        x3 = x-1
        y3 = y
        x4 = x-1
        y4 = y+1

        x5 = x+1
        y5 = y
        x6 = x
        y6 = y+1
    elif travel_dir == '-y':
        x1 = x-1
        y1 = y+1
        x2 = x
        y2 = y+1
        x3 = x+1
        y3 = y
        x4 = x+1
        y4 = y-1

        x5 = x-1
        y5 = y
        x6 = x
        y6 = y-1



    return  [
        {
        "end": (x1,y1),
        "center": (x5,y5),
        "radius": 1,
        "type": "circle",
        "direction": "negative"
        },
        {
        "end":(x2,y2),
        "type": "line"
        },
        {
         "end":(x3,y3),
         "center":(x,y),
         "radius": 1,
         "type": "circle",
         "direction": "negative"
```

```
        },
        {
        "end": (x4,y4),
        "type": "line"
        },
        {
        "end": (x,y),
        "center":(x6,y6),
        "radius": 1,
        "type": "circle",
        "direction": "negative"
        }
        ]

def turn180(x,y):
    x1 = x+1
    y1 = y
    x2 = x
    y2 = y+1
    x3 = x
    y3 = y-1
    x4 = x+1
    y4 = y
    x5 = x
    y5 = y
    return [
        {"end": (x1,y1),
         "type": "line"
        },
        {"end": (x2,y2),
         "center": (x+1,y+1),
         "radius": 1,
         "type": "circle",
         "direction": "positive"
        },
        {"end": (x3,y3),
         "type": "line"
        },
        {"end": (x4,y4),
         "center": (x+1,y-1),
         "radius": 1,
         "type": "circle",
         "direction": "positive"
        },
        {"end": (x5,y5),
         "type": "line"
```

```
        }


    ]
```

A.6 comb_cut.py

Det är denna filen som beskriver för klipparen hur den ska åka för att kamma planen.

```
import sys
import json
import os

current_file = __file__
parent_dir = os.path.dirname(current_file)
sibling_dir = os.path.join(parent_dir, "userInput")
child_file = os.path.join(sibling_dir, "paintDimension.json")
child_file_rel = os.path.relpath(child_file, current_file)

def get_comb_cut_order():
    with open(child_file_rel) as paintDimensions:
        user_input = json.load(paintDimensions)
    width = round(float(user_input["shortside"]), 4)
    lenght = round(float(user_input["longside"]), 4)
    print("width: ", width)
    print("lenght: ", lenght)
    num_comb_lines = round(int(user_input["numCombLines"]), 4)

    width_combing = lenght / num_comb_lines
    length_combing = width
    width_comb = 0.3
    num_combs_per_area = int(width_combing / width_comb)

    comb_cut_order = []
    for k in range(num_combs_per_area):
        for i in range(num_comb_lines):
            if not i % 2:
                x1_e = 0
                x2_e = width
                y1_e = i*width_combing + k*width_comb
                y2_e = y1_e
                trip_e = {"start": (x1_e, y1_e), "end": (x2_e, y2_e),
                ↪  "type": "line"}
                comb_cut_order.append(trip_e)
                to_next_e = {"start": (x2_e, y2_e), "end": (x2_e,
                ↪  y2_e+width_combing), "type": "line"}
                comb_cut_order.append(to_next_e)
```

```
            else:
                x1_o = width
                x2_o = 0
                y1_o = i*width_combing + k*width_comb
                y2_o = y1_o
                trip_o = {"start": (x1_o, y1_o), "end": (x2_o, y2_o),
                 ↪  "type": "line"}
                comb_cut_order.append(trip_o)
                if i != (num_comb_lines-1):
                    to_next_o = {"start": (x2_o, y2_o), "end": (x2_o,
                     ↪  y2_o+width_combing), "type": "line"}
                    comb_cut_order.append(to_next_o)
                else:
                    to_start = {"start": (x2_o, y2_o), "end": (0,
                     ↪  0+k*width_comb), "type": "line"}
                    comb_cut_order.append(to_start)


    return comb_cut_order
```

A.7 README.md

Detta är README.md filen för projektet vilket beskriver hur klipparen ska sättas igång.

```
# painting automower
This project succesfully steers an autommower, a 2D robot
   , to a path with a precition of about 0.1-0.5 m.
This was accomplished by using odometry and GPS with a
   Kalman filter.

To change lengths of lines:

1. Start local host by running the server.py.
2. Go to <http://localhost:8000/userInput/>
3. Input the dimensions of your soccer field
4. Click submit

To start control mower:

1. run this anywhere in terminal: "source ~/450/devel/
   setup.bash"
2. run this anywhere in terminal: "sudo chmod 666 /dev/
   ttyACM0" (if error, check if automower is on)
3. Enter password: "kandidatarbete"
```

```
4. run this anywhere in terminal: "roslaunch
   am_driver_safe automower_hrp.launch"
5. Open a second terminal, run "cd ~/450/src/calculations
   "
6. run "source ~/450/devel/setup.bash"
7. run "rosrun am_driver main.py" (If error, run this in
   450/src/hrp/am_driver/scripts: "chmod 777 main.py")

To run simulation:

1. Change trip to paint or comb

- Paint:
  - 1 Change paint_order in paint.py to desierd trip
  - 2 Change self.order = paint_order in simulate.py
- Comb and cut
  - 1 Change comb_order in comb_cut.py to desierd trip
  - 2 Change self.order = comb_cut_order in simulate.py

2. Run "cd calculations", then run "python3 simulate.py"
3. Check you simulated result in calculations/plots/plot-
   latest.png

To get position run this in terminal "cd 450/src/hrp/
   am_driver/scripts", then: "rostopic echo /pose"
To get GPS run this in termrminal: "cd 450/src/hrp/
   am_driver/scripts", then: "rostopic echo /GPSfix"
To see all rostopics run this in termrminal: "cd 450/src/
   hrp/am_driver/scripts", then: "rostopic echo /GPSfix"


SSH

1. Install vscode and the plugin ssh to vscode
2. Connect to the same wifi as the jetson nano (SSID:
   CASELAB, Password:CaseLocalNet)
3. Open ssh extention (bottom left green), connect to
   host, "ssh kandidatarbete@192.168.1.110"
4. Open 450/src

To change wifi

1. sudo nano /etc/netplan/01-network-manager-all.yaml
2. change to correct wifi, copy one of these:

network:
```

```
  version: 2
  renderer: networkd
  wifis:
    wlan0:
      dhcp4: no
      addresses:
        - 192.168.1.110/24
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
      access-points:
        "CASELAB":
          password: "CaseLocalNet"

network:
  version: 2
  renderer: networkd
  wifis:
    wlan0:
      dhcp4: no
      addresses:
        - 192.168.1.110/24
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
      access-points:
        "GalaxyS20":
          password: "Input the password of your phones
             internet"

3. sudo netplan apply
```

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS