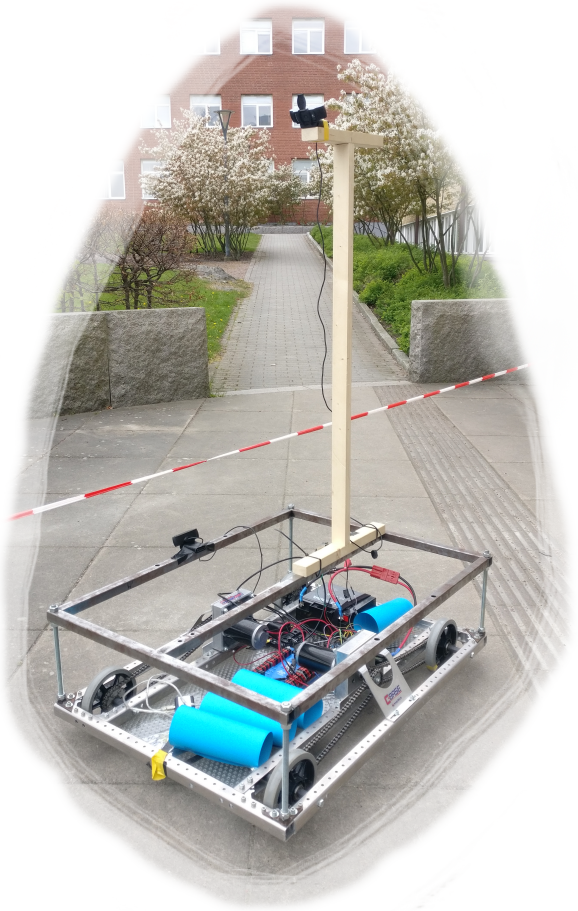
CHALMERS
UNIVERSITY OF TECHNOLOGY



Autonom snöröjare för cykelbanor

Ett projekt för att utveckla ett system för navigering längs cykelbanor

SSYX02 - Kandidatarbete SysCon 2017

Emil Andersson, Fredrik Nilsson, Mathias Nilsson,
Simon Persson, Oscar Sten & Johan Wendel

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Kandidatarbete

Autonom snöröjare för cykelbanor

Ett projekt för att utveckla ett system för navigering längs cykelbanor

Emil Andersson, Fredrik Nilsson, Mathias Nilsson,
Simon Persson, Oscar Sten & Johan Wendel

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

Autonom snöröjare för cyckelbanor
Ett projekt för att utveckla ett system för navigering längs cykelbanor
Emil Andersson, Fredrik Nilsson, Mathias Nilsson,
Simon Persson, Oscar Sten & Johan Wendel

Handledare: Jonas Sjöberg, Department of Signals and Systems
Examinator: Jonas Fredriksson, Department of Signals and Systems

Kandidatuppsats 2017:NN
Department of Signals and Systems

Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Omslag: Prototypen uppställd bredvid avspärrningsband.

Autonom snöröjare för cykelbanor
Ett projekt för att utveckla ett system för navigering längs cykelbanor
Emil Andersson, Fredrik Nilsson, Mathias Nilsson,
Simon Persson, Oscar Sten & Johan Wendel
Department of Signals And Systems
Chalmers University of Technology

Sammanfattning

Syftet med projektet var att ta fram lösningar som grund för utveckling av en autonom snöröjare för cykelbanor. Två alternativ presenteras för att navigera längs cykelbanor, båda baserade på bildanalys och kompass. Första alternativet använder markerade stolpar för navigation, i tester representerade med blåa papperscylindrar och det andra ett band att navigera efter. Kommunikation mellan delsystemen sker via operativsystemet ROS. En simulering och reglersystem skapades i Mathworks Simulink. Prototypen upptäcker fotgängare och stannar tills körbanan är fri. Delsystemen uppvisar tillfredsställande resultat i enskilda tester men kunde inte valideras som ett sammansatt system.

Abstract

The purpose of the project was to develop solutions as a base for development of an autonomous snowplow for bike lanes. Two alternatives were presented for navigation along the bike lanes, both utilised image processing and a digital compass. The first alternative tracked marked posts for navigation, in the tests these were repressented with blue paper cylinders, and the other a ribbon to navigate. Communication between the subsystems was conducted via the ROS operating system. A simulation and a controlsystem was designed in Mathworks' Simulink. The prototype can detect pedestrians and stop til the path is free. The subsystems show satisfying results in isolated tests, but it was not possible to validate the full system.

Nyckelord: snöröjare, autonom, robot, bildanalys, snö, snöröjning, Robot Operating System, OpenCV, simulering

Innehåll

Figurer

Tabeller

1
Inledning

1.1 Bakgrund

Ett ekonomiskt, miljövänligt och praktiskt färdsätt i Göteborg och många andra städer är att cykla. Göteborgs kommun har dessutom gjort stora satsningar på cykeltrafiken under de senaste åren [9]. Dessvärre gör Sveriges klimat det besvärligt att använda detta färdmedel året om. Under vintern är det som mest problem då snöfall och halka både försvårar och ökar riskerna med cyklande.

Enligt Göteborgs stad ska snöröjning av cykelbanor påbörjas innan snölagret hunnit bli tre centimeter och vara färdigt inom sex timmar efter att det slutat snöa, samma kriterier som gäller för de vanliga vägbanorna [7]. Denna mängd innebär inga större problem för biltrafiken men tre centimeter gör markant skillnad i framkomlighet för cyklister och kraftigare snöfall kommer på detta tidsspann göra cykelbanorna oframkomliga. Storstädernas budget idag för snöröjning ligger över 100 miljoner. De inhyrda bolagen klarar dessutom sällan att hålla kommunernas tidskrav vad gäller cykelbanorna[3]. Prioritet läggs istället på viktiga bilvägar och busslinjer.

Begränsad framkomlighet på dagar med kraftigt snöfall leder kan leda till ett minskat intresse för att ta cykeln. Det gäller då även dagar med bra framkomlighet och är ohållbart om städerna är allvarliga med att få bort biltrafiken i centrala delar. Det bör därför ligga i stadens intresse att hitta en lösning som minst kan se till att de viktigaste cykelbanorna alltid är framkomliga året om. En lösning för detta skulle kunna vara en teknisk installation utplacerad på de mest kritiska platserna som kontinuerligt kan hålla dessa rena.

1.2 Syfte

Syftet var att konstruera en prototyp för autonom snöröjning av cykelbanor. Den skall innefatta förslag på hård- och mjukvarulösningar som skulle kunna appliceras i en framtida kommersiell produkt.

1.3 Problemformulering

Då en cykelbana saknar något som hindrar en autonom robot från att köra av måste ett navigeringssystem konstrueras för att kunna hålla sig innanför cykelbanans

gränser. I ett praktikfall kan synliga gränser i den utsträckning som de finns antas vara dolda under snön och kommer inte kunna användas som riktlinjer. Cykelbanor trafikeras samtidigt av fotgängare, cyklar och enstaka djur. Roboten behöver kunna hantera möten med andra trafikanter som inte får orsaka skador på någon av de inblandade. Slutligen saknar Sverige standardisering eller krav på utförandet av cykelbanor, därför saknas en mall att följa men för att ha praktisk tillämpning måste lösningen vara anpassningsbar för flera olika cykelbanor.

1.4 Avgränsningar

Det område som skall trafikeras och röjas av roboten kan vara känt, data gällande omgivningen kan ha samlats in och vara förinläst i roboten.

Prototypen behövde inte ha samtliga lösningar för hårdvara, framdrivning av roboten inkluderat men ej begränsat till motor, hjul, växellåda och energikälla behövde inte dimensioneras. Detta innebär att prototypen inte gavs några specifika effektkrav på dessa för att kunna röja snö i praktiken. En existerande motorplattform kallad NI ROCK (NI Robotics CompactRIO Kit) användes som bas för övrig utrustning. Detta medförde ytterligare avgränsingen att den ej går att framföra på ojämnt underlag.

1.5 Kravspecifikation

Under 1.4 specificerades att projektet inte var att skapa en komplett prototyp för snöröjning. Arbetet har dock förts med målet att lösningarna på delfunktioner skulle kunna användas i kommersiellt syfte. Förslag på kravspecifikationen för en sådan komplett produkt finns i appendix, bilaga A.1. Utifrån denna i linje med projektets syfte formulerades följande krav för prototypen:

Tabell 1.1: Prototypkrav

Beskrivning	Verifieringskrav
Roboten skall kunna följa en förutbestämd bana och inte avvika mer än 20 cm åt något håll	Avståndsmätning vid praktiska tester
Lösningen skall ej göra stora ingrepp i omgivningen	Diskussion, bedömning utifrån estetik samt motiverad kostnad för en kommun
Roboten ska kunna detektera rörliga hinder och undvika att köra på dem.	Låghastighetstest (max 1 m/s) i labbmiljö med fotgängare
Roboten ska kunna detektera stationära hinder och undvika att köra på dem	Tester i labbmiljö

2
Teori

I detta kapitel ges en sammanfattning av de matematiska verktyg och programspråk som utnyttjats i projektet.

2.1 PID-reglering - en sammanfattning

PID-reglering har i projektet använts till att styra roboten på rätt kurs genom att två saker reglerades, robotens riktningsvinkel och avståndet från den önskade banan. Mer om hur står i avsnitt 3.5.

En regulator kontrollerar ett uppmätt värde mot ett referensvärde och hanterar detta enligt en matematisk formel. Vid återkopplad reglering subtraheras det uppmäta värdet från det önskade referensvärdet. Den resulterande differensen eller felet (betecknas e) förstärks enligt formeln och denna reglersignal skickas vidare till nästa delsystem. I en PID-regulator är den matematiska formelen för förstärkningen summan av: felet e multiplicerat med en konstant, integralen av e multiplicerat med en konstant och derivatan av e multiplicerat med en konstant, se ekvation 2.1 [2].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.1}$$

Dimensionering av dessa tre konstanter P, I och D ger regulatorn dess egenskaper. En optimal regulator för alla tillämpningar existerar inte utan det är en balans mellan med vilken hastigheten regulatorn svarar på förändring, hur stora överslagen blir och hur väl den konvergerar.

2.2 Matematisk modell av differentialstyrning

Motorplattformen som användes till prototypen är differentialstyrd. Hur nedanstående modell använts beskrivs i avsnitt 3.4.

Differentialstyrning innebär att olika vinkelhastighet på vardera sida fordonet skapar sväng eller rotation. Detta styrsätt utnyttjas även av bandfordon samt tvåhjuliga fordon och kallas differentialstyrning. Principen kan modelleras genom följande ekvationssystem för sträcka i x- och y-led samt vinkel på fordonet.

$$dx = \frac{r}{2} \cdot (\omega_l + \omega_r \cdot \cos\theta) \tag{2.2}$$

$$dy = \frac{r}{2} \cdot (\omega_l + \omega_r \cdot \sin\theta) \tag{2.3}$$

$$d\theta = \frac{r}{L} \cdot (\omega_l - \omega_r) \tag{2.4}$$

De två ω representerar vinkelhastigheten för hjulen på vänster respektive höger sida. Om vänster och höger hjul har samma hastighet rör sig fordonet rakt fram. Ifall ena sidan är långsammare än den andra kommer roboten svänga åt riktningen med det långsammare hjulet. Om båda hjulen har samma men motriktade vinkelhastigheter kommer roboten rotera på plats[4].

I verkligheten kommer sällan båda hjulen ha samma vinkelhastighet trots att båda styrmotorer erhåller samma signal. Detta beror på en mängd faktorer som olika tröghet i lager och tillverkningsskillnader i motorerna. Pågrund av detta krävs det ofta i praktiken någon form av feedback till reglersystemet för utsignalerna om man vill att roboten skall köra helt rakt.

2.3 Avståndsmätning med kamera

Dessa formler används i avsnitt 3.7.2.
För att beräkna avståndet till ett objekt från en kamera behövs kamerans brännvidd, brännpunktens avstånd från linsen F som ges av ekvationen

$$F = \frac{PD}{W} \tag{2.5}$$

i vilken P är objektets bredd i bilden mätt i pixlar, D är avståndet till kameran och W är den verkligen bredden av föremålet. När brännvidden F beräknats kan denna användas för att kontinuerligt beräkna avståndet till objektet. Detta görs genom att lösa ut avståndet D ur ekvation 2.5 vilket ger

$$D = \frac{WF}{P} \tag{2.6}$$

där pixelbredden P kontinuerligt uppdateras vilket i sin tur leder till uppdatering av avståndet D. Värt att notera är att höjden på föremålet lika gärna kan användas istället för bredden.

2.4 Kalmanfilter för tillståndsuppskattning

Implementeringen av Kalmanfilter sker i avsnitt 3.7.1. Kalmanfilter är ett beräkningsorgan för att uppskatta tillståndet hos en process, något som är användbart då exempelvis data med mycket brus erhålls. Enkelt sammanfattat skapas först en förutsägelse om vad för mätvärde som kommer erhållas samt hur stort felet sannolikt kommer vara. När mätvärdet erhålls beräknas Kalmanförstärkningen som säger hur stor inverkan mätvärdet skall ha och därefter uppdateras de uppskattade värdena till vad som anses vara korrekt. Samma process upprepas för varje nytt mätvärde.

I ”An Introduction to the Kalman Filter” från University of North Carolina beskrivs Kalmanfilter som följande
“Kalmanfiltret är en uppsättning matematiska ekvationer som tillhandahåller ett effektivt beräkningsorgan (rekursivt) för att uppskatta tillståndet hos en process på ett sätt som minimerar medelvärdet av kvadratfelet . Filtret är mycket kraftfull i flera avseenden: den stöder uppskattningar av dåtid, nutid, och även framtida tillstånd, och det kan göra så även när den exakta arten av det modellerade systemet är okänt.”[1]

För att finna en konstant signal med pålagt brus används följande formler. Den linjära differentialekvation

$$x_k = x_{k-1} + w_k \tag{2.7}$$

vilken styr processen med mätningen z

$$z_k = x_k + v_k \tag{2.8}$$

Där w_k och v_k är processens respektive mätningens brus. De ekvationer som uppdateras med tiden och fungerar som en förutsägelse är

$$\hat{x}_k^- = \hat{x}_{k-1} \tag{2.9}$$

$$P_k^- = P_{k-1} + Q \tag{2.10}$$

och de ekvationer som uppdateras varje mätning och ger de korrekta värdena är

$$K_k = P_k^-(P_k^- + R)^{-1} = \frac{P_k^-}{P_k^- + R} \tag{2.11}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{x}_k^-) \tag{2.12}$$

$$P_k = (1 - K_k)P_k^- \tag{2.13}$$

Där Q är processens och R är mätningens brus-kovarians, ett statistiskt mått på samvariationen mellan två slumpvariabler. $\hat{x}_{k-1}$ är det uppskattade mätvärdet och P_k^- är den uppskattade fel-kovariansen. K är Kalmanförstärkningen och $\hat{x}_k$ samt P_k är den uppdaterade mätningen respektive felet.

2.5 OpenCV

Implementeringen av OpenCV förklaras i avsnitt 3.6 och 3.7. OpenCV är ett open source bildanalyseringsbibliotek som bland annat används för att detektera människor. OpenCV biblioteket importeras in det program som kodas och sedan kan samtliga metoder enkelt användas. Användbara metoder som finns i OpenCV är bland andra:

inRange - Metoden tar en bild och ett färg-spann i form av två arrays med tre värden som inparametrar. Detta resulterar i en svartvit bild där allt är svart förutom det inom spannet som istället är vitt.

morphologyEx - En metod som tar en svartvit bild som inparameter och först förstorar allt vitt med en viss faktor, för att sedan förminska allt vitt i den nya bilden med samma faktor och returnera detta. Det leder till att vita områden som är något brusiga blir fylligare och klarare.

dilate - Samma funktion som ”morphologyEX” förutom att ”dilate” inte förminskar de vita områdena efter att ha förstorat dem.

HoughLinesP - Tar en bild och några inställningsparametrar som inparametrar och returnerar en lista med alla potentiella linjer i bilden.

fitLine - Tar en lista med linjer samt inställningsparametrar som inparametrar och resulterar den mest troliga linjen baserad på samtliga linjer.

Camshift - Följer ett objekt i ett videoflöde. Camshift tar hänsyn till att ett objekts storlek och form kan ändras beroende på hur det rör sig i videoflödet.

2.6 Haar Cascades

Genom att passera in en stor mängd positiva (innehållande det som söks) och negativa (innehållande vad som helst utom det som söks) bilder kan en lång textfil med information om vad som ska letas efter i videoflödet skapas. Denna fil kallas en Haar Cascade och med dess hjälp kan sedan de eftersökta objekten letas upp.

2.7 Robot Operation System (ROS)

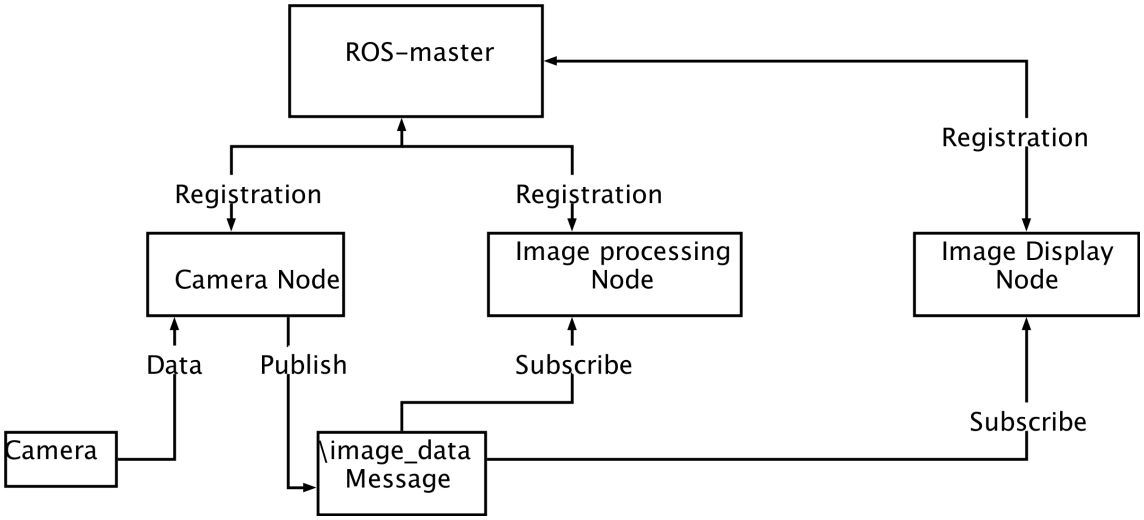
I projektet har Robot Operating System använts för kommunikation mellan delsystemen. Implementationen finns beskriven i avsnitt 3.4.

ROS är ett ramverk för att få olika delsystem av en robot att kommunicera. ROS består av ett antal noder som skickar medelande mellan sig. Dessa noder är program eller sensorer som roboten behöver för att uppnå sin funktionalitet. Dessa noder måste även vara kopplade till en huvudnod som kallas $ROSMaster$, denna nod har en administrativ roll i nätverket och tillhandahåller namn och registreringstjänster för noderna. Noderna skickar information med hjälp av så kallade *topics*. En nod kan antingen skicka information till en topic för andra noder att läsa eller avläsa en topic för att sen processera informationen[6].

ROS tillhandhåller också flera kod-biblotek och plugins för bland annat testning. RQT är ett plugin till ROS för att lätt kunna visualisera data från olika delar av

systemet[10]. Denna visualiseringen ges i form av grafer där data uppdateras i realtid. Detta ger en överblick som underlättar felsöking och testning.

Ett exempel på ett enklare nätverk av noder visas i figur 3.4. Figuren består av tre noder som är markerade med cirklar och är kopplade till $ROSMaster$. $CameraNode$ skriver värden till $/image\_dataMessage$ som de andra två noderna läser ifrån.



Figur 2.1: Exempel på ett enkelt ROS nätverk

3
Metod

Detta metodkapitel presenterar lösningar för delproblem samt material och mjukvara som använts. Tester som beskrivs i detta kapitel har skett under projektets gång för att utvärdera och motivera val av hård och mjukvara. Sluttester finns beskrivna i kapitel fyra.

3.1 Lösningssökning för omvärldsanalys

Utifrån kravspecifikationen som kan ses i tabell 1.1 formulerades de tre delproblemen för lösningssökning som presenteras i tabell 3.1 nedan. Postionering innebär att roboten skall kunna följa en cykelbana. Rörliga hinder innebär roboten skall upptäcka och undvika kollison med andra trafikanter. Stillastående hinder innebär att roboten skall upptäcka och undvika kollision med okända objekt på cykelbanan. I tabellen kan även avläsas de teoretiska lösningsförslag som undersöktes. Följande avsnitt presenterar kort de valda lösningarna, mer information om varför de andra valdes bort finns under avsnitt 5.2

Tabell 3.1: Lösningsmatris

Problem	**Lösningsförslag**			
Positionering	GPS	Kamera	IR-sensor	Linjeföljning
Rörliga hinder	Kamera	IR	Värmekamera	
Stillastående hinder	Kamera	IR	Registrerar stopp	

3.1.1 Lösningsval för positionering och navigering

Roboten skall kunna följa en cykelbana. De fyra potentiella lösningarna i tabell 3.1 jämfördes för navigeringen då det finns exempel där de använts att styra robotar tidigare.

Lösningarna jämfördes utifrån kriterierna:

- Pris: kostnad på den extra hårdvara och mjukvara som behövs köpas.
- Exakthet: Med vilken precision de teoretiskt kan uppfylla problemet.
- Lätthet att implementera: Hur lång tid uppskattas det ta att implementera lösningen.

Möjligheten sågs över att använda flera av lösningarna i kombination.

Det slutgiltiga valet landade på kameran, med hjälp av den kan roboten följa en förutbestämd bana genom att följa riktmärken. Dessa registreras i förväg och kan tillsammans med bildanalys hittas av kameran. Nackdelen är att det är komplicerat att implementera och bildanalys kan kräva mycket processorkraft. Fördelen är att det är en billig lösning relativt de andra. Kamerans bilder kan även användas i andra syften som upptäcka hinder eller ge feedback till operatör.

3.1.2 Lösningsval rörliga hinder

I avsnitt 1.3 beskrivs problemet med andra trafikanter som roboten behöver undvika kollison med. De tre föreslagna tekniska lösningarna i tabell 3.1 bedömdes utifrån samma kriterier som 3.1.1.

Valet blev en webbkamera. Det är kostnadseffektivt och hög upplösning krävs ej för enkla former samt tillämpningar där skarpa kanter inte är viktigt. Ytterligare pixlar ökar även mängden processorkraft som krävs för bildanalys. Valet stärktes av beslutet att använda kameror för positioneringen då samma kan användas. Den största nackdelen ligger i att en vanlig kamera inte har några filter för objekt utan detta måste skötas helt av extern programvara.

3.1.3 Lösningsval stillastående hinder

Ett komplext problem finns i nytillkomna objekt på cykelbanan som klarar att stoppa roboten. Om ett objekt i robotens bana placerats där innan snöfall kan det dessutom vara helt eller delvis täckt av snö. Kriterierna som användes var samma som i avsnitt 3.1.1 med tillägg att även innefatta risken för felaktig detektering av hinder när den inte behöver stanna. De tre hanteringsmöjligheterna var kamera, IR eller endast stanna vid kollision.

Komplexiteten i problemet ledde till att ingen lösning hittades som uppfyllde kravspecifikationen, under 5.2 diskuteras detta mer ingående.

3.2 Lösningssökning datorhårdvara

Roboten är uppbyggd av ett antal delsystem och vissa av dem är prestationstunga processer som videoanalys. Detta ställer ett krav på en dator med hög prestanda. Därav valdes en bärbar persondator som huvuddator till roboten. Motorerna styrs med hjälp av en analog signal. Detta kräver en mikrokontroller med passande utgångar som även har prestanda nog att driva en ROS-nod för att kunna ta in information från resten av roboten. Arduino UNO är en mikrokontroller som tillfredsställer dessa krav och likt huvuddatorn fanns tillgängligt under projektets genomförande.

3.3 Teknisk specifikation för hårdvara

3.3.1 Kameror som använts för bildanalys

Projektet har utnyttjat två stycken kameror av typen Logitech c920 vars specifikationer finns i tabell 3.2 nedan. För projektets bildanalys har en nedskalad upplösning 640x480 från kamerorna använts, mer om implementeringen kan läsas under avsnitt 3.7.

Tabell 3.2: Specifikation på kamerorna

Modell	Logitech C920 HD Pro
Stillbildsupplösning	15 Megapixel(programintepolerat)
Videoupplösning	1920x1080
Mikrofon	Ja

3.3.2 Digital kompass för vinkelmätning

Till projektet användes den digitala kompassen Compass Module 3-Axis HMC5883L [8]. Specifikationerna kan ses i följande tabell

Tabell 3.3: Specifikation på kompassen

Magnetisk fältgräns	+/-8gauss
Säkerhet kompassbäring	1-2°
Strömkrav	2,7 till 6,5 VDC
Kommunikationsgränssnitt	I2C(upp till 400kHz)
Dimensioner	1,9 x 1,7cm
Drifttemperaturer	-30 till +85°C

3.3.3 Den befintliga basplattformen

Plattformen som använts av tidigare kandidatarbeten kallas NI(National Instruments) Robotics CompactRIO Kit. Denna plattform innehåller två PMDC-motorer med tillhörande utväxlingar och impulsgivare. Dessa styrs av två stycken kontrollmoduler och drivs av ett 12V blybatteri på 18Ah.

Tabell 3.4: Specifikation på elmotorerna

Modell:	M4-R0062-12	Effektivitet:	65%
Driftspänning:	6V-12V	Varvtal vid max effektivitet:	4614
Nominellt:	12V	Moment vid max effektivitet:	0.3178 Nm
Varvtal utan last:	5310	Ström vid max effektivitet:	19.8A
Ström utan last:	2.7A	Vikt:	1.3kg
Maxmoment:	2.424 Nm	Längd:	109.6mm
Maxström:	133A	Diameter:	66mm
Kt:	0.018 Nm/A	Axeldiameter:	8mm
Kv:	443 rpm/V	Axellängd:	35.6mm

Tabell 3.5: Specifikation på impulsgivarna

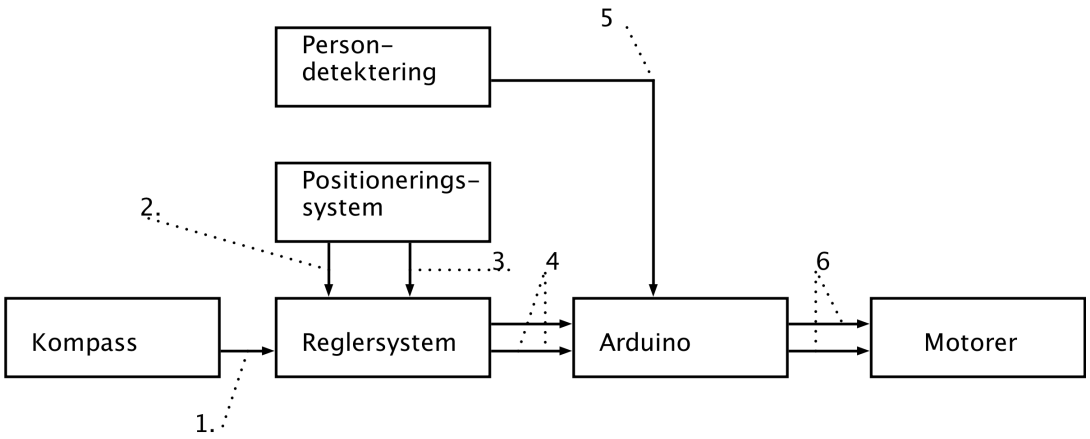
Modell:	E4P-250-250-D-D-D-B	Pin 1:	+5V
Driftspänning:	4.5V-5.5V	Pin 2:	Kanal A
Nominellt:	5V	Pin 3:	Jord
CPR:	250 pulser/varv	Pin 4:	Kanal B

Tabell 3.6: Specifikation på motorkontrollmodulerna

Modell:	MDL-BDC24
Driftspänning:	6V-30V
Styrsignal:	Servo PWM
	CAN (6P6C- och 6P4C-kontakt)
	RS232 (yttersta 2 pinnarna på 6P6C-kontakten)

3.4 ROS-nätverk

ROS-nätverket har strukturerats så att varje nod består av ett delsystem. En skiss över nätverket kan ses i figur 3.1 och en lista med dess komponenter i tabell 3.7. Varje pil i figur 3.1 representerar en *topic*. Pilens början visar vilken nod som publicerar värden på den givna *topicen* och pilens slut visar vilken nod som tar del av informationen. Alla värden som skickas, med stop-signalen som undantag, är av typen $float64$ med anledningen att reglersystemet kräver detta för att kunna utföra sina beräkningar. Stopsignalen är av typen *boolean* för att smidigt kunna signalera arduinon att starta eller stoppa motorerna.



Figur 3.1: Skiss över ROS-nätverket

Tabell 3.7: Komponenter i ROS-nätverket

Nr.	
1.	Robotens vinkel från kompassen
2.	Beräknat avstånd genererat av sidokameran
3.	Beräknat avstånd genererat av frontkameran
4.	Stoppsignal
5.	Reglersystemets utsignal. Vinkelhastighet för respektive motor
6.	Analoga signaler till motorerna

3.5 Simulering av linjeföljning

En simulering skapades för att utvärdera styr och reglersystemet innan det användes för att köra prototypen. Utgångspunkten var att skapa en utsignal i form av en graf eller tabell som påvisar hur bra reglersystemet uppfyller kravspecifikationen i tabell 1.1. MathWorks Simulink användes för att simulera på grund av tidigare vana samt att programmet har en omfattande dokumentation.

Simuleringen plottar robotens position i x och y led utifrån den matematiska modell som beskrivs i avsnitt 2.2. Modellen användes till att dimensionera PID-regulatorerna och bedöma hur mätinstrumentens noggrannhet skulle påverka robotens möjlighet att enligt kravspecifikationen följa en banan inom 20 cm. Som specificerat i avgränsningarna (avsnitt 1.4) är området som roboten skall trafikera alltid känt sedan tidigare och den kan därför ha förprogrammerade kommandon för när den skall svänga som aktiveras vid exempelvis specifika in-signaler.

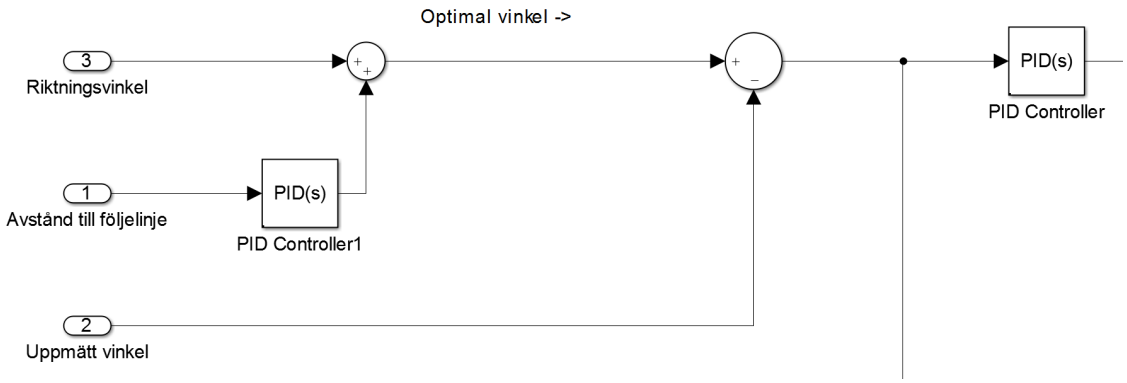
Insignalen till reglersystemet är robotens nuvarande position i x-led gentemot en i programvaran förutbestämd linje, samt robotens vinkel. För att simulera det verkliga avläsningsfelet adderades till insignalerna varje gång en siffra på en normalfördelat

intervall. Genom att ändra detta intervall kunde det i simuleringen undersökas i vilken utsträckning fel i vinkel och positionsavläsning påverkade linjeföljningen. Detta för att undersöka vilken mätosäkerhet som var viktigast för det reglersystem vi skapat och fungera som stöd till val av hårdvara för avläsning. Den slutsats som gavs av simuleringen var att med den styrreglering som skapades är exaktheten på vinkeln mycket viktigare än positionen. En avgränsning för simuleringen gjordes inom mekaniken där accelerationströgheten hos roboten ej simulerats utan hastighetsförändringar sker omedelbart.

3.5.1 Reglersystemet

Kriteriet från kravspecifikationen (avsnitt 1.5) är robotens bana skall hållas inom ±20 cm från ett utsatt linje. För att detta skall uppnås krävs snabbhet i systemet i skarpa kurvor för att klara av kriteriet. Det bör också finnas hantering för överslag och begränsa sick-sack körning då detta sänker prestandan då roboten ej färdas längsta möjliga avstånd från startpunkten per tidsenhet vilket anses strävansvärt. Slutligen måste mätfelen från positionsavläsning hanteras.

För att reglera detta valdes en PID-regulator där parametrarna kan justeras för att balansera mellan tidigare nämda problemområden. PID beskrivs i teoriavsnitt 2.1. En bild av hur systemet ser ut i simulink kan ses i figur 3.2. I robotens reglersystem användes två insignaler. Den första insignalen är avståndet roboten befinner sig ifrån den tänkta linjen den skall följa, vilket i praktiken mäts av kameror. Denna insignal skickas in i den första PID-regulatorn, resultatet subtraheras sedan från den förprogrammerade vinkeln på linjen som skall följas. Resultatet tolkas som en ny vinkel för hur roboten skall svänga in mot linjen. Denna vinkel subtraheras med insignal två som är nuvarande uppmätt vinkel och felet går in i andra PID-regulatorn. Det är sedan denna signal som skickas till styrsystemet.



Figur 3.2: Regleringsdelen av styrsystemet

3.6 Fotgängardetektering

För att undvika att snöröjaren kolliderar med fotgängare på cykelbanan behöver den kunna upptäcka dessa. Genom att använda Haar Cascades med underkroppar kan fotgängare detekteras i snöröjarens väg.

För att undvika att snöröjaren stannar vid falsk positiva träffar implementerades en funktion som utnyttjar en *stack*. Programmet som styr detekteringen utvärderar kontinuerligt de 10 senaste värdena (True, False) och skickar ut en stoppsignal till motorerna ifall minst 5 är träffar (True). Detta innebär att snöröjaren stannar och står still tills det att vägen framför den är fri från fotgängare. Pythonkoden som användes kan ses i A.5.

3.7 Positioneringssystemen

För positioneringssystemet har två metoder använts. Båda använder kameran men erhåller vinkelställning och avstånd i förhållande till den önskade banan på olika sätt. Den första lösning som använts är linjedetektion med kameran (Se Avsnitt 3.7.1) som följer ett band. Denna lösning är inte beroende av en kompass men behöver extern hjälp att hantera skarpa svängar. Den andra lösningen som använts är Duo-kamera-positioneringssystem (Se Avsnitt 3.7.2). Denna lösning är beroende av en kompass men kräver ingen annan extern input. Dessutom finns möjlighet att väga de olika lösningarna mot varandra alternativt kombinera dem.

3.7.1 Linjedetektion med kamera

Lösningen baseras på befintlig teknik i självkörande bilar vilken beskrivs mer ingående i 5.2.2. Snöröjaren skall efter detektion av ett avspärrningsband kunna beräkna snöröjarens avstånd till bandet och dess vinkelställning i förhållande till det.

3.7.1.1 Problematik

Utöver den problematik som beskrivs i avsnitt 1.3 och 5.2.1.1 ställs lösningen med linjer inför vissa specifika problem under de förhållanden som råder med snö och en cykelbana. Eftersom de ofta ligger i anslutning till vägar som i sin tur plogas är det problematiskt att sätta upp någonting temporärt på gränsen mellan bilväg och cykelbana utan att det påverkas av den snö plogbilen plöjer av vägen vid ett kraftigt snöfall. Således måste systemet kunna förhålla sig till endast en linje som den inte ska ligga på, vilket kräver komplexare matematik än att ligga emellan två linjer.

3.7.1.2 Lösningsidé

För att linjedetektion med kamera skall kunna fungera trots problematiken som beskrivs ovan krävs modifieringar.

För en kommersiell lösning hade något bestående behövt användas, men för detta projekt var det enklast att använda ett band. Bandet spändes upp ovanför marken för att lösa problematiken med att linjen kan vara dold nära marken. Bandet som användes är ett avspärrningsband i vitt och rött men endast det röda i bandet används.
Pågrund av problemet med att endast en linje kan användas kommer bandet att spännas upp längs kanten på cykelbanan som inte gränsar mot en bilväg om sådan finnes. För att ta reda på snöröjarens vinkelställning och avstånd analyserades linjens placering och lutning. Linjens lutning är direkt kopplat till snöröjarens vinkelställning och dess placering i kamerabilden är kopplat till avståndet snöröjaren är placerad på.

3.7.1.3 Framtagning av lösning

Bildanalys skedde i steg, där bilden behandlades och resulterade i ut-parametrar. Stegen beskrivs nedan.

För att optimera programmet och minska datorkraften som krävs förminskades bilden till en sextondel och beskars till bara de delar som ansågs intressanta. Det vill säga delen av bilden där banden syns. Beskärningen minskade inte bara den datorkraften som krävs utan också risken från störningar från andra objekt.

Bandet är rödfärgat, därför isolerades rött i bilden genom OpenCV-funktionen *inRange* (se 2.5). Allt rött blev då vitt och allt annat svart (Se Figur 3.3). Detta röda förbättrades genom antingen funktionen *morphologyEX* eller *dilated*, varav den första gjorde det röda mindre brusigt och den andra tydligare, men osäkrare.

Figur 3.3: Kamerabild på band efter utförd inRange-funktion

Alla potentiella linjer togs fram genom funktionen $HoughLinesP$ från bilden som blivit förbättrad med metoden $morphologyEX$. Om denna bild inte var tydlig nog, dvs. det inga linjer hittades, togs linjerna istället fram från bilden förbättrad med *dilate*.

Dessa linjer användes som inparametrar i metoden $fitLine$ (se 2.5) varpå den mest troliga linjen erhölls. Linjen består av lutning och en punkt på linjen. För att denna punkt skall kunna vara jämförbar tas istället den y-koordinat där linjen korsar högra kanten av bilden fram.

För att öka stabiliteten i programmet och minska påverkan av felaktiga mätningar implementerades ett Kalmanfilter (se 2.4) både för lutningen och för y-koordinaten.

Slutligen togs faktorer och formler fram för att räkna om y-koordinaten till ett avstånd samt en önskad lutning som den uppmätta kan avvika från med ett specifikt antal radianer. Detta görs genom att studera värdena för y-koordinat och lutning vid olika vinklar och avstånd och sedan generera en approximerad funktion för att omvandla detta. Detaljerade formler för detta samt koden finns i Appendix A.2.

3.7.1.4 Implementering

Vid den första implementeringen av programmet i prototypen drogs slutsatsen att kameran satt på ungefär samma avstånd från marken som bandet. Detta resulterade i dåliga värden eftersom då kameran befinner sig på samma nivå som bandet spelar det ingen större roll vilken betraktningsvinkel kameran har. Bandet har ändå ungefär lutningen noll. Kameran behövde alltså placeras högre upp och en ställning konstruerades för kameran. Därefter kunde tester på prestanda utföras.

3.7.1.5 Felkällor

Eftersom positioneringen baseras på kamerans detektion av avspärrningsbandet är ljus en potentiell felkälla. Utomhus i gott ljus är körning dock inga problem.

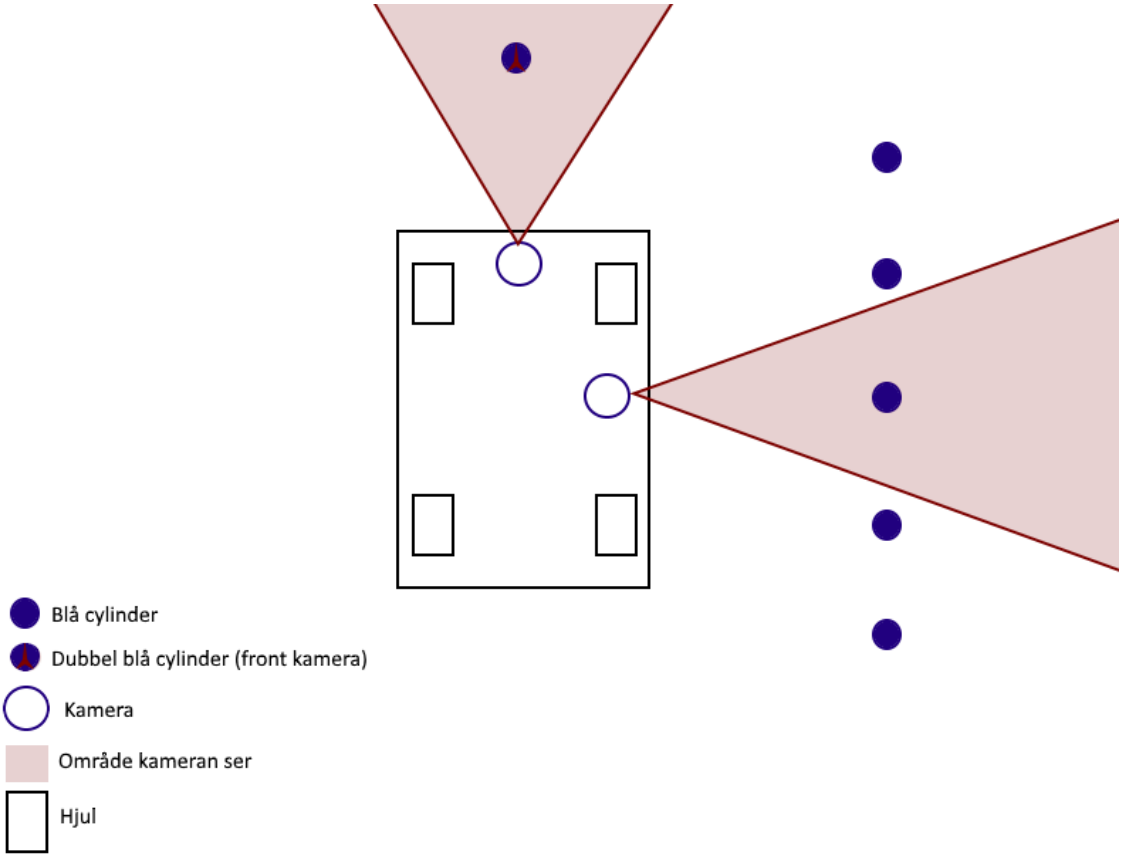
3.7.2 Duo-kamerasystem med digital kompass

Denna lösningen använder två kameror som sitter vinkelrätt mot varanda samt en digital kompass för att entydigt bestämma snöröjarens position. Pythonkoden som användes kan ses i A.6.

3.7.2.1 Lösningsidé

Lösningen baseras på avståndsmätningen som beskrivs i avsnitt 2.3 och används med hjälp av två kameror vinkelrätt placerade för att ge avstånd i x- samt y-led till ett referenssystem. För detta projekt används enkla cylindrar som skapats av blå A4-papper men för en kommersiell produkt kan markeringar på lyktstolpar användas. För att kunna utföra svängar och veta med säkerhet var snöröjaren befinner sig

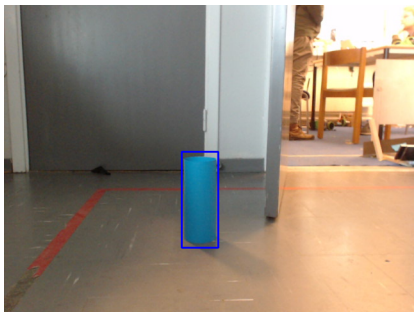
används en digital kompass som ger vinkelavvikningen från referenssystemet. En överblick av systemet kan ses i bilden nedan.



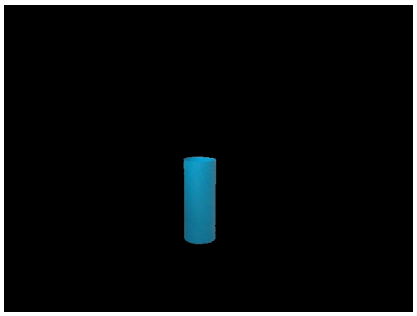
Figur 3.4: Överblick över snöröjaren och dess positionering

3.7.2.2 Bildbehandlingsteknik

För att kunna använda avståndsmätningen krävdes ett sätt för snöröjaren att alltid kunna urskilja de blå cylindrarna. Detta görs genom att först applicera ett filter i OpenCV där endast blå färg släpps igenom.



Figur 3.5: Utan filter



Figur 3.6: Med filter

När endast de blå cylindrarna syns kan tracking med hjälp av OpenCV funktionen

Camshift användas för att kontinuerligt hålla koll på storlek och position på cylindrarna. När kameran inte har sett en cylinder på 10 frames letar den genast upp en ny vilket leder till en i stort sett momentan övergång.

3.7.2.3 Felkällor

Eftersom positioneringen baseras på vad kameran ser, var ljus en potentiell felkälla. Detta löstes dock i stort sätt fullständigt med hjälp av det blå filter som applicerades. Problem uppstod endast vid extrema ljusförhållanden där det dessutom fanns blå golv eller liknande runt omkring.

4

Resultat och Tester

Detta kapitel sammanfattar testerna som gjorts för de slutgiltiga dellösningarna och utvärderar om de uppfyllt kravspecifikationerna. Diskussion kring dessa och förklaring på förbättringar tas upp i efterföljande kapitel 5.

4.1 Testresultat av hårdvaran

Hårdvaran testades initialt med hjälp av en IR-fjärrkontroll, resultatet blev att plattformen gick att fjärrstyra men att den drog lite åt sidan när den skulle köra rakt fram. Dessutom noterades att styrsignaler under 10 % inte gav någon respons från motorerna.

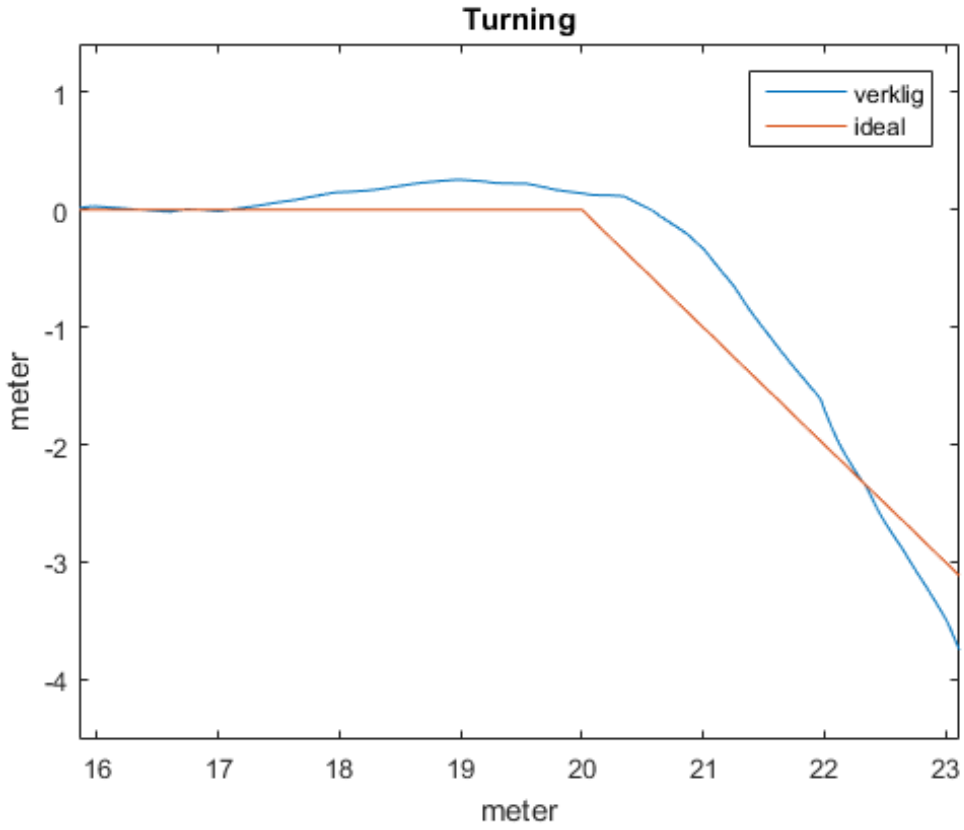
Därefter kördes tester med hjälp av ROS, där data kunde läsas in från pulsgivare samt kompass. Pulsgivarna testades genom att markera hjulen och låta Arduinon räkna hur många pulser som gavs på ett varv. Kompassen testades för exakthet i relativ vinkel, hur nära norr den pekar testades inte då det inte är något som använts. Vid en rotation på 90 grader stämde det nya värdet med en noggrannhet på $\pm$ 1 grad.

4.2 Testresultat av reglersystem i simulering

Simuleringen som beskrivs i avsnitt 3.5 användes för att utvärdera robotens reglersystem. I testerna simulerades den verkliga mätosäkerheten i position som en normalfördelad slumpmässig störning med varians 5cm och i vinkelavläsning med en varians på 0.02 radianer. Simuleringen kördes upprepade gånger och resultaten från den kinematiska modellen plottades i Matlab där största avvikelsen noterades. PID-regulatorn uppdaterades därefter och testades således iterativt. Resultaten kan avläsas i tabell 4.1. Systemet testades även för svängar vid förbestämda tidpunkter en plott av en sväng inzoomad ses i figur 4.1.

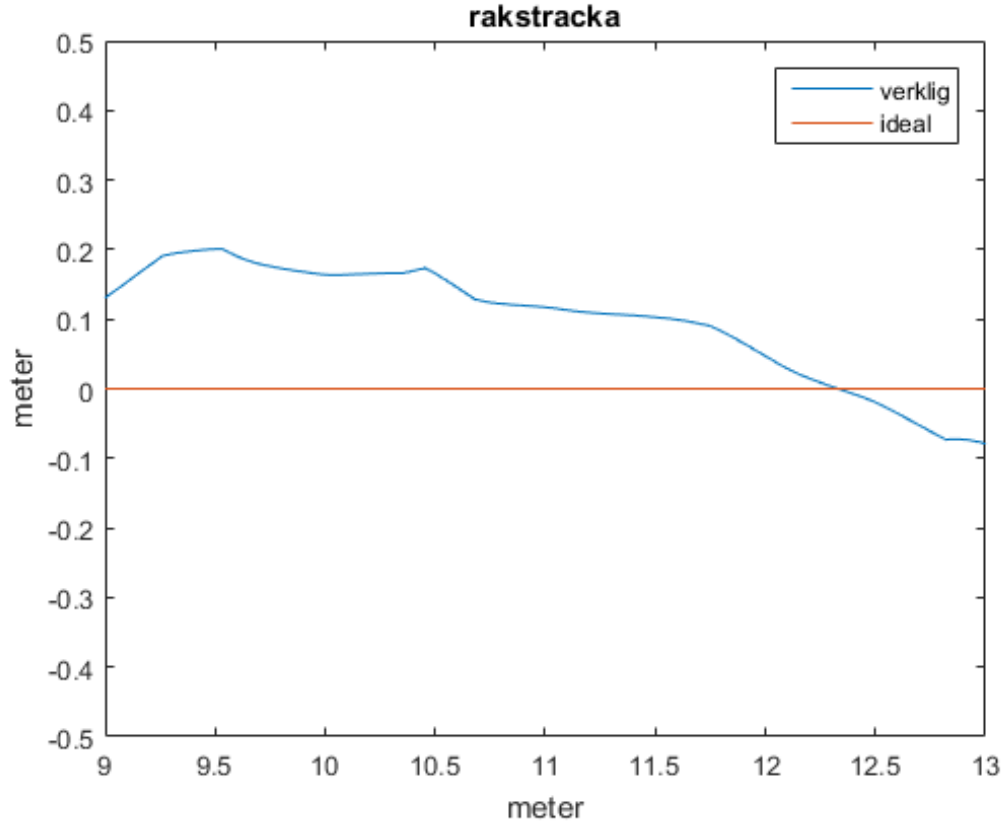
Tabell 4.1: Maximala avvikelse på simulerad 150m racksträcka

Test	1	2	3	4	5	6	7	8	9	10
Avikelse	0.242	0.2844	0.3848	0.2204	0.2326	0.2138	0.3510	0.2121	0.2295	0.2568

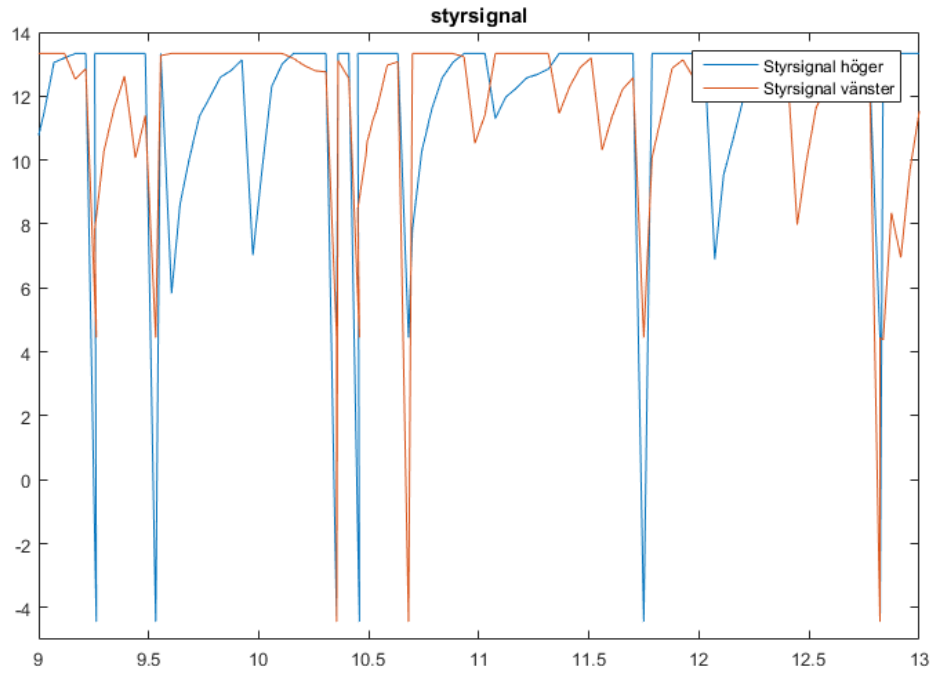


Figur 4.1: Robotens bana vid en 45° sväng

I figur 4.2 visas ett delintervall av den simulerade körningen på raksträcka. Figur 4.3 visar styrsignalerna som samtidigt skickas till hjulen på vänster respektive höger sida på samma intervall. Detta resultat plottas för att undersöka att styrsystemet inte skiftar allt för snabbt. Skulle de fluktuera allt för mycket skulle det skapa problem för de riktiga motorerna.



Figur 4.2: Simulering av hur roboten kör under ett tidsintervall på en raksträcka



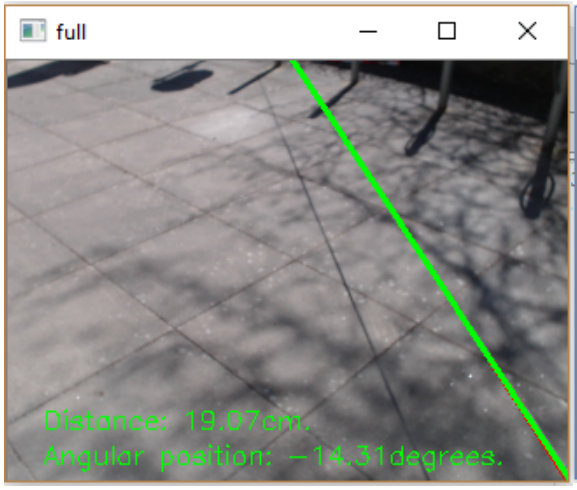
Figur 4.3: Styrsignaler, vänster och höger hjul på raksträcka, samma tidsintervall som figur4.2

4.3 Testresultat av linjedetektion med kamera

För att verifiera att kriteriet ”Roboten ska kunna följa en tänkt bana och inte avvika mer än 20 cm åt något håll” går att uppfylla med positioneringslösningen ”Linjedetektion med kamera” spändes bandet, som roboten skall följa, upp på 30 cm höjd. Kamerans höjd och betraktningsvinkel är mycket viktig och var vid mätningen 152 cm och cirka -30 grader om 0 grader är horisontellt. Roboten placerades i en serie av olika vinklar och avstånd varpå mätvärdena antecknades. Resultatet presenteras nedan. Den gröna linje som är utritad i Figur 4.4-4.7 är den linje som programmet funnit och bör täcka det uppspända bandet.

Figur 4.4: Avstånd: 30 cm. Vinkel: 0.

Figur 4.5: Avstånd: 70cm. Vinkel: 0.



Figur 4.6: Avstånd: 30cm. Vinkel:-10,3.

Figur 4.7: Avstånd: 30cm.Vinkel: 10,3.

Linjedetektion fungerar inte då bandet är utanför kamerans bild. Det sker då avståndet till bandet är stort (Se Figur 4.5) och/eller då vinkeln är stor och positiv (Se Figur 4.7). Vid mätning uppskattades programmet fungera upp till ett avstånd på ungefär 100 cm.

4.3.1 Sluttest

Sluttest av linjedetektion med kamera genomfördes. Arbetsgången för framtagning av kod och omvandling av mätvärden till resultat finns att finna i Appendix A.2. Mätvärden vid test efter att denna förbättring gjorts ser ut som följande.

Tabell 4.2: Samtliga vinkel- & avståndsmätningar från mätning

Vinkel(^{o})	Uppmätt(^{o})	Fel(^{o})	%	Avstånd(cm)	Uppmätt(cm)	Fel(cm)	%
-21.69	-21.88	-0.19	0.88	30	28.17	2.83	9.43
-10.3	-10.14	0.16	1.55	30	30.18	0.18	0.60
-6.48	-6.74	-0.26	4.01	30	30.17	0.17	0.57
-2.6	-3.03	-0.43	18.1	30	29.79	-0.21	0.70
0	-0.27	-0.27	-	30	29.90	-0.10	0.33
2.6	1.87	-0.73	28.1	30	29.93	-0.07	0.23
6.48	4.85	-1.63	25.2	30	27.85	-2.15	7.17
10.3	8.84	1.46	14.2	30	27.67	-2.33	7.77
-6.48	-4.57	1.91	29.5	70	63.39	-6.61	9.44
0	0.07	0.07	-	70	66.54	3.46	4.94
6.48	3.21	-3.27	50.5	70	72.24	2.24	3.20

Tabell 4.3: Osäkerhet för linjedetektion med kamera från Mätning

Typ	Avstånd	Vinkelställning
Största mätosäkerhet	$\pm 6.61cm$	$\pm 3.27^o$
Genomsnittlig osäkerhet	$\pm 1.85cm$	$\pm 0.91^o$
Genomsnittlig osäkerhet vid 30cm	$\pm 1.01cm$	$\pm 0.59^o$
Osäkerhet vid vinkel $\pm 2.6^o$ & avstånd 30cm	$\pm 0.48cm$	$\pm 0.13^o$

4.3.2 Sammanfattning

Programmet fungerar väl för att både ge avstånd till bandet och vinkelställning. Det fungerar bäst utomhus i goda ljusförhållanden. Kamerans begränsade vidd leder till att snöröjaren som mest kan köra 100cm ifrån bandet, men 70cm är rekommenderat max. Då kan en cirka 140cm bred väg skottas med snöröjarens nuvarande bredd.

Programmet mäter avstånd och vinkelställning med en genomsnittlig osäkerhet på $\pm 1.85cm$ och $\pm 0.91^o$. När den kör vid vinkel $\pm 2.6^o$ & avstånd 30cm är säkerheten istället $\pm 0.48cm$ och $\pm 0.13^o$. Denna osäkerhet liknar den för att mäta upp det verkliga scenariot vid mätning och anses bra.

Då hela systemet inte kunnat testas går inte uppfyllandet av kravet på positionering inom 20cm från önskat läge att säkerställa. Programmet för linjedetektion med kamera möjliggör åtminstone att kravet skulle kunna uppfyllas.

4.4 Testresultat av navigering med cylindrar

För att verifiera att avståndsmätningen ger korrekta värden gjordes tester på olika avstånd. En tumstock lades ut längs marken och blå cylindrar placerades längs vägen. Det verkliga och det uppmätta avståndet noterades. Resultatet presenteras i tabellerna nedan.

Tabell 4.4: Sidokamera

Verkligt avstånd (m)	0.6	0.8	1	1.2	1.4	1.6	1.8	2
Uppmätt (m)	0.63	0.80	1.02	1.20	1.37	1.51	1.69	1.9
Fel (m)	0.03	0	0.02	0	0.03	0.11	0.11	0.10
Fel (%)	5.0	0.0	2.0	0.0	2.1	7.3	6.1	5.0

Tabell 4.5: Frontkamera

Verkligt avstånd (m)	1	1.5	2	2.5	3
Uppmätt (m)	0.97	1.47	1.92	2.31	2.69
Fel (m)	0.03	0.03	0.08	0.19	0.31
Fel (%)	3.0	2.0	4.0	7.6	10.3

Resultatet anses bra eftersom frontkameran inte behöver någon direkt noggrannhet på långt avstånd eftersom den bara behövs för att avgöra när en sväng kommer ske. Sidokamerans små fel inom 1.5m är acceptabla men vid större avstånd är felen för stora. Detta går att lösa och behandlas i avsnitt 5.1.4.2.

4.5 Detekterning av rörliga hinder

För att uppfylla kravet för rörliga hinder skall detekteringen fungera då roboten kör och minimumkrav på respons är att stanna. Test utfördes genom att roboten kör längs en testbana och en försöksperson går in framför, om allt fungerar korrekt skall roboten stanna. Testet görs med försökspersoner inträdande i robotens synvinkel på flera olika avstånd och från olika vinklar. Målet bedöms uppfyllt då robotens hastighet inte överstiger 1 m/s.

Testet bedöms som lyckat eftersom snöröjaren korrekt upptäckte och stannade för fotgängare den annars skulle ha kört på. Tiden innan den stannade varierar beroende på avstånd och vinkel.

4.6 Detektering av fasta hinder

Inget lösningförslag som skulle kunna upptäcka en majoritet av ny stationära hinder kunde hittas. I diskussion kapitlet, sektion 5.1 och 5.2 diskuteras framtida arbete med detta problem.

4.7 Ingrepp i omgivningen

I kravspecifikationen i avsnitt 1.5 anges kravet att lösningen inte skall kräva för stora ingrepp längs robotens körbana, samt att dessa inte får kosta för mycket. Tekniken att sätta upp och markera stolpar kräver att avstånd mellan dessa markeringar inte blir för stort för att kameran skall kunna upptäcka dem. I den implementerade lösningen kan detta maxavstånd beräknas med

$$2 \cdot d \cdot \tan(\frac{v}{2}) \tag{4.1}$$

där d är avståndet från kameran till markeringarna roboten skall följa (kameran kan sitta var som helst på roboten) och v är kamerans bildvinkel. Med 68 graders bildvinkel som den använda kameran har och en 1m bred robot ger det ett maxavstånd på 2,70m. I Sverige finns ingen standard på avstånd mellan lyktstolpar, privat mätning har visat sig det varierar mellan 20-35m i tätbebyggelse. Även med andra stolpar, elstationer och annat att använda som markörer kommer fem-sex stolpar extra behövs sättas ner. Dessa behöver endast ha låg höjd men innebär fortfarande ett stort ingrepp, detta diskuteras vidare i sektion 5.1.
För lösningen ”Linjedetektion med kamera” hade röda band behövt spännas upp 30cm över marken längsmed cykelbanan. Det är enkelt att genomföra och dessutom måste banden inte vara helt sammanhängande, utan det kan lämnas öppningar. Dessa öppningar bör dock inte vara större än 2 meter med den vidd som nuvarande kamera har.

5
Diskussion

5.1 Vidareutveckling

Här diskuteras möjlig vidarutveckling av de tekniska lösningarna hos prototypen. Förslag för en kravspecifikation för en komplett produkt finns i bilaga A.1.

5.1.1 Reglersystemet

Reglersystemet uppfyller ännu inte i simulering helt kravspecifikationen, framförallt är hanteringen av skarpa svängar undermålig. Ett förslag är att reglersystemet används på raksträckor och små svängar. Definera problematiskt stora svängar och då roboten skall svänga dessa låt den övergå till ett alternativt körläge där den roterar på plats tills den pekar i rätt vinkel.

Simuleringen som beskrivs i 3.5 för att testa reglersystemet innehåller förenklingar. En mer verklighetsnära simulering hade gett dimensionerade värden hos PID regulatorn som kräver mindre finjusteringar efter praktiska tester. I nuläget är ett uppskattat mätfel inlagt i simuleringen. En uppdatering av de slutgiltiga mätfelet hos kameror och kompass som sköter positionnering och kompassriktning innan dimensionering av PID hade gjort skillnad. Förslagsvis implementeras också acceleration som hindrar momentana hastighetsförändringar.

5.1.2 Fotgängardetektion

Programmet som beskrivs i 3.6 identifierar gångtrafikanter och signalerar åt roboten att stanna tills de passerat. Denna lösning fungerar bäst vid lågtrafikerade tider på dygnet, exempelvis tidig morgon. För att köra effektivt vid mer vältrafikerade tider krävs vidarutveckling. Exempelvis kan hänsyn tas till avstånd för att roboten endast skall stanna nära en trafikant. Optimalt är detta kombinerat med om vägbanans bredd tillåter att låta roboten köra åt sidan för att passera trafikanten utan att stanna

5.1.3 Motorstyrning

Motorstyrningen kan ta emot olika typer av insignaler som nämns i avsnitt 3.3.3, av dessa har CAN och RS232 större möjligheter till en mer exakt reglering då det

går att köra med direkt återkoppling av hastigheten. Dessutom finns möjlighet att övervaka ström och spänning vilket kan användas för att detektera att man kört på ett hinder vilket leder till onormalt hög ström samt avgöra när det är dags att ladda batteriet.

5.1.4 Positioneringssystemen

5.1.4.1 Linjedetektion med kamera

Önskvärt hade varit att arbeta fram en noggrannare funktion för att omvandla den räta linje som programmet finner till ett avstånd och en vinkelställning. De funktioner som finns går att göra bättre genom fler mätdata, upprepade mätningar och justeringar.

Den osäkerhet som nu existerar vid små vinkelskillnader runt avståndet 30 cm är så liten att uppmätningen av det verkliga scenario, vid mätningar, med hjälp av tumstock i princip har samma mätosäkerhet. Osäkerheten vid 70 cm avstånd är inte riktigt lika bra och extra arbete för att optimera denna omvandling hade varit önskvärt.

Programmet har ibland problem att detektera avspärrningbandet. Detta beror i de flesta fall på ljuset och programmet har svårt att hantera mörkläggande skuggor eller bländande sol. En stark lampa hade varit lämplig för körning i mörker och ett helt rödfärgat brett band hade förmodligen förbättrat prestandan. Dock anses inte problemet med bländning så stort vid snöfall. Stark vind har också en viss inverkan vid körning. Samtliga mätningar har genomförts vid, som mest, svag vind. För att minska påverkan av vind rekommenderas användning av ett mer aerodynamisk och mindre elastiskt band än det avspärrningsband i tunn plast som använts. Ett spännband i rött tyg hade exempelvis varit lämpligt.

För tillfället fungerar Linjedetektionen med kamera som mest 100 cm ifrån bandet. Det hade kunnat vara önskvärt att kunna köra på ett längre avstånd för att ploga en bred cykelbana. Då hade kamera med vidare vinkel eller högre upplösning behövt användas.

5.1.4.2 Duo-kamera system med digital kompass

De två huvudsakliga områden för förbättringarna som kan göras till Duo-kamera systemet är högre precision och högre tolerans mot avstånd mellan markeringarna (blå cylindrar eller dylikt).

Precision kan förbättras med att använda en mer högupplöst bild (kamerans maximala upplösning används inte) men detta kräver mer processorkraft.

Högre tolerans mot avstånd mellan markeringarna kan lösas på framförallt två olika sätt. Det första är att kameran kan rotera och därav får kan fånga ett betydligt

större område. Det andra är att utnyttja en tredje kamera (alternativt utnyttja frontkameran) för att positionera utifrån en enskild markering. Detta kräver vinkeln mellan de två kamerorna vilket enkelt kan bestämmas vid montering.

5.1.5 Tester att följa en bana

Systemet klarade aldrig att köra en färdig tesbana men för att slutgiltigt verifiera systemet föreslås en testbana. Positioneringsband eller stolpar placeras på lämpligt avstånd och med tejp eller krita markeras det uppmätta avstånd från banan som ej är tillåtet. Medan roboten följer banan sker samtidigt visuell inspektion som avgör om roboten korsar markeringarna. En enkel men lämplig bana är kvadratisk eller rektangulär för att verifiera att roboten klarar skarpa svängar samt köra en bana 360 grader från startvinkeln.

5.2 Genomförande

5.2.1 Lösningssökning

Komplettring till avsnitt 3.1 om varför de andra alternativen valdes bort. Kriterierna presenterades i samma kapitel. Bakgrunden till priskriteriet var för att hålla nere kostnaderna, exakta prisgränser för varje dellösning sattes ej men projektetbudget var 5000kr. Att hålla nere komponentkostnaden motiverades även av det som nämns i kravspecifikationen 1.5, att målet var att teoretiskt kunna implementera dellösningar i en kommersiell produkt.

5.2.1.1 Navigering

För och nackdelar med den valda lösningen kamera beskrivs i 3.1 här följer en kort förklaring varför de andra alternativen övergavs.

GPS kan positionera roboten oavsett vart den befinner sig i världen och är lätt att implementera som färdig modul. Den föll på exaktheten i kombination med priset. De inom en relevant prisklass har en precisionen mellan 1-10 meter. På marknaden finns GPS med precision ner till centimeternivå men de ännu långt över projektets budget, skulle detta priset sjunka drastiskt är GPS ett motiverat val.

Klassisk linjeföljning med en målad linje utnyttjar IR-sensorer detta ströks då marken kommer vara täckt av snö som täcker linjer och IR-sensorer har också problem med reflexiva material som snö. En lösning markbaserad för linjeföljning sågs initialt hos gräsklipparrobotar där en “linje” istället skapas av en nedgrävd strömförande kabel. Detta kan utnyttjas även för snöröjaren och ger en mycket exakt position. Tekniken för att lägga något liknande i asfalt finns idag som utnyttjas för dragning av fiberkablar. Det bedömdes dock tidskrävande och innebär stor installationskostnad för en eventuell kommersiell produkt samt gör cykelbanan otillgänglig under installation.

5.2.1.2 Rörliga hinder

De rörliga hinder som beskrivs i problemformuleringen, avsnitt 1.3 är levande och varma vilket borde gjort värmekamera till ett självklar lösning i avsnitt 3.1. I kallt väder och snö skulle dessutom varma kroppar relativt framstå tydligare och den värmefiltrerade bilden kräver endast simpel bildanalys på färgintensitet för att utläsa vart någonting varmt befinner sig. Värmekamerans vinst på prestanda förlorade mot att den billigaste värmekameran i skrivande stund på marknaden kostar över 2000kr. Då två webbkameror redan fanns att tillgå utan extra kostnad som dessutom kunde användas som lösning för positionnering valdes det istället.

5.2.1.3 Stillastående hinder

I 3.1 presenterades att ingen lösning för stillastående hinder implementerades. Det föreslagna alternativet IR valdes bort på felrisk kriteriet, IR mäter avstånd genom reflekterat ljus vilket ger risk för felmätningar eller falska positiva vid snö. Med ett säkerhetsavstånd på minst en meter för att kunna stanna finns också en risk att skyltstolpar, soptunnor och annat längs med vägen hade gett utslag. Kamera och bildanalys kan endast hitta förregistrerade objekt genom att leta efter specifika former och färger. Snön gör också att objekt kan vara delvis täckta. Pågrund av att hindren är odefinierade försvinner kameran som alternativ. Lösningen att endast stanna vid kollision visade sig svår att realisera utan trycksensorer, tanken var att stanna då en signal gick till motorerna men inte hjulen rörde sig. Denna situation är dock mindre trolig på fruset underlag och mer troligt är att hjulen slirar.

5.2.1.4 Motorstyrning

Det finns tre olika sätt att styra kontrollmodulerna för motorerna som nämns i avsnitt 3.3.3. Av dessa valdes att styra med vanlig servo-PWM signal eftersom en Arduino enkelt kan skicka ut sådana signaler utan något ytterligare kretskort för att sköta kommunikationen. En nackdel med denna lösning är att vi inte kan använda den inbyggda återkopplingen i kontrollmodulen, istället läses pulsgivarna av arduinon som återkopplar till styrsystemet. En stor fördel med den inbyggda återkopplingen är att den tar hand om problemet som nämns i avsnitt 4.1 angående att den kan dra lite snett och direkt kompenserar för detta.

5.2.2 Linjedetektion med kamera

Lösningen i avsnitt 3.7.1 för positioneringen med hjälp av linjedetektion med kamera baserades på en befintlig lösning i självkörande bilar. Utveckling av system för självkörande bilar har pågått sedan sent 80-tal. Systemen som finns i självkörande bilar idag baserar positioneringen på linjeföljning med kamera. De linjer som systemet förhåller sig till är väglinjerna i asfalten, som finns på var sin sida om bilen. Dessa linjer finnes på många olika sätt, bland annat genom att studera hörnen, kanter, färger och textur. Systemet reglerar för att bilen skall ligga mitt emellan väglinjerna[5].

6
Slutsats

Projektet har dessvärre inte resulterat i en fullt fungerande autonom snöröjare för cykelbanor. Att navigera efter stolpar verkar enligt testerna fungera bra på avstånd mellan 0,6 och 1,4m, på 1,6 meters avstånd blir dock felet för stort. Således som står i avsnitt 4.7 innebär det 2.70 meter mellan stolparna med nuvarande kameralösning. Linjedetektionssystemet mäter avstånd och vinkelställning bra under 1 meters avstånd. Systemet är optimerat för avståndet 30 cm och mätosäkerheten ökar med differensen från detta optimerade avstånd. Detektion av fotgängare fungerar med få fel.

Reglera system har visat sig vara svårt. De störningar som använts i simulation är dock bara uppskattningar baserade på observationer under testningen.

Delsystemen ger lovande resultat i enskilda tester men ett okänt fel gör att när allting sätts samman fungerar inte roboten som förväntat. Under felsökning identifierades ett antal buggar som åtgärdades, dock kvarstod felet. Vår rekommendation för framtida projekt eller vidareutveckling är att utnyttja lösningar, idéer och navigeringssystemen men undvika kedjan från reglersystem till hårdvara då det innehåller okända fel.

Litteratur

[1] Greg Welch och Gary Bishop. "An Introduction to the Kalman Filter". I: (2006). URL: http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

[2] Bengt Lennartson. *Reglerteknikens grunder*. Studentlitteratur, 2000, s. 513.

[3] Johan Hellekant. *Snöröjningen brister trots mångmiljonavtal*. Okt. 2016. URL: https://www.svd.se/bristande-snorojning-trots-mangmiljonavtal.

[4] Steven LaValle. *13.1.2.2 A differential drive*. 2012. URL: http://planning.cs.uiuc.edu/node659.html.

[5] Xin Liu, Xin Xu och Bin Dai. "Vision-based long-distance lane perception and front vehicle location for full autonomous vehicles on highway roads". I: *Journal of Central South University* 19.5 (maj 2012), s. 1454–1465. ISSN: 2095-2899. DOI: 10.1007/s11771-012-1162-7. URL: http://link.springer.com/10.1007/s11771-012-1162-7.

[6] Morgan Quigley m. fl. "ROS: an open-source Robot Operating System". I: (). URL: https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

[7] *Om snöröjning och sandning - Göteborgs Stad*.

[8] Parallax Inc. *Datasheet Compass Module 3-Axis HMC5883L*. URL: http://docs-europe.electrocomponents.com/webdocs/1242/0900766b812421f9.pdf.

[9] Anders Sahlberg. *Göteborg fördubblar cykelsatsning*. Göteborg, 2016. URL: http://www.gp.se/nyheter/g%C3%B6teborg/g%C3%B6teborg-f%C3%B6rdubblar-cykelsatsning-1.3465093.

[10] Blasdel Aaron Thomas Dirk Scholz Dorian. *rqt - ROS Wiki*. URL: http://wiki.ros.org/rqt.

A

Appendix

A.1 Kravspecifikation kommersiell produkt

Tabell A.1: Kravspecifikation för eventuell kommersiell produkt

Beskrivning	**Verifiering**	**Krav/ Önskemål**
Täcka vattenkänsliga delar	Tätslutande för att hindra vatten och fukt att nå kretsarna	Krav
Röja snö	Effekt nog för 5cm djup snö	Krav
Klara av stötar		Krav
Maximal fordonsbredd	80cm	Krav
Inte köra på andra trafikanter	Undvika genom stopp eller manöver	Krav
Inte köra på stillastående trafikanter	Undvika genom stopp eller manöver	Önskemål
Upptäcka nya hinder och bromsa innan kollision	Stanna eller bromsa tillräckligt för att undvika skador på roboten.	Krav
Manövrera runt stationära hinder	Manövrera runt blockeringen om ytan tillåter passage	Önskemål
Kunna särskilja olika stationära hinder	Kan särskilja skräp som ej behöver undvikas t.ex burkar och påsar.	Önskemål
Kunna följa en förutbestämd rutt och återvända till denna	Aviker ej från rutten med mer än 20cm i någon riktning	Krav
Kunna vända på en cykelbana	Maximalt 150cm bredd att vända på	Krav
Kunna själv docka till laddningsstation	Autonomt uppsöker laddningsstation innan batteriet tar slut	Krav
Startar utan behov av operatör	Vet själv när röjning krävs	Önskemål
Skickar felmeddelanden till operatör	Minimum, meddelande indikerar den stannat	Krav
Signalerar svängar	Visuell signal som tydligt indikerar stora svängar för omgivningen	Önskemål
Varningsljus/ljud	Roboten avger ljus och/eller ljud som varnar omgivande trafik	Krav
Ej störa sovande boende	<65dB vid trafikering i bostadsområde	Krav

A.2 Omvandlingsfunktion

När omvandlingsfunktionen togs fram samlades data från en rad olika avstånd och vinklar. Då lästes den räta linje, som erhålls, av vilket är samlat i följande Tabell A.2.

Tabell A.2: Mätdata inför omvandlingsfunktion

Avstånd	Vinkel	m-värde	lutning
30	0.0	206.452	2.0452
30	-2.6	214.952	1.90813
30	-4.87	222.605	1.73669
30	-10.93	232.467	1.48358
30	-21.4	246.5733	1.10156
30	2.6	197.1305	2.27296
30	6.48	177.02446	2.784644
30	10.3	144.124481	3.391929
30	14.03	94.6845	4.279761
70	-5.84	106.8285	1.152076
70	0.0	85.9532	1.27236
70	5.19	49.9571	1.468115

Tabell A.3: Samtliga vinkel- & avståndsmätningar från Mätning 1

Vinkel(^{o})	Uppmätt(^{o})	Fel(^{o})	%	Avstånd(cm)	Uppmätt(cm)	Fel(cm)	%
-25	-34.09	9.09	36.4	30	14.73	15.27	50.9
-10,3	-14.31	4.01	38.9	30	19.07	10.93	36.4
-6.48	-9.45	2.97	45.8	30	22.30	7.70	25.7
-2.6	-4.18	1.58	60.8	30	24.89	5.11	17.0
0.0	-0.07	0.07	-	30	27.71	2.29	7.6
2.6	1.08	1.52	58.5	30	31.42	1.42	4.7
6.48	5.81	0.67	10.3	30	37.29	7.29	24.3
10.3	10.33	0.03	0.29	30	47.42	17.42	58.1
-6.48	-6.81	0.33	5.09	70	59.58	10.42	14.9
0.0	-0.30	0.30	-	70	68.37	1.63	2.3
6.48	3.16	3.32	51.2	70	80.52	10.52	15.0

Tabell A.4: Osäkerhet för linjedetektion med kamera vid Mätning 1

Typ	Avstånd	Vinkelställning
Genomsnittlig osäkerhet	$\pm 8.2cm$	$\pm 2.17^o$
Osäkerhet vid vinkel 0^o	$\pm 2.3cm$	$\pm 0.3^o$

Resultatet från mätning 1 anses bra bortsett från att avståndsberäkningen inte tar hänsyn till vinkelställningen, trots att det är något som avståndet beror på. Detta implementerades.

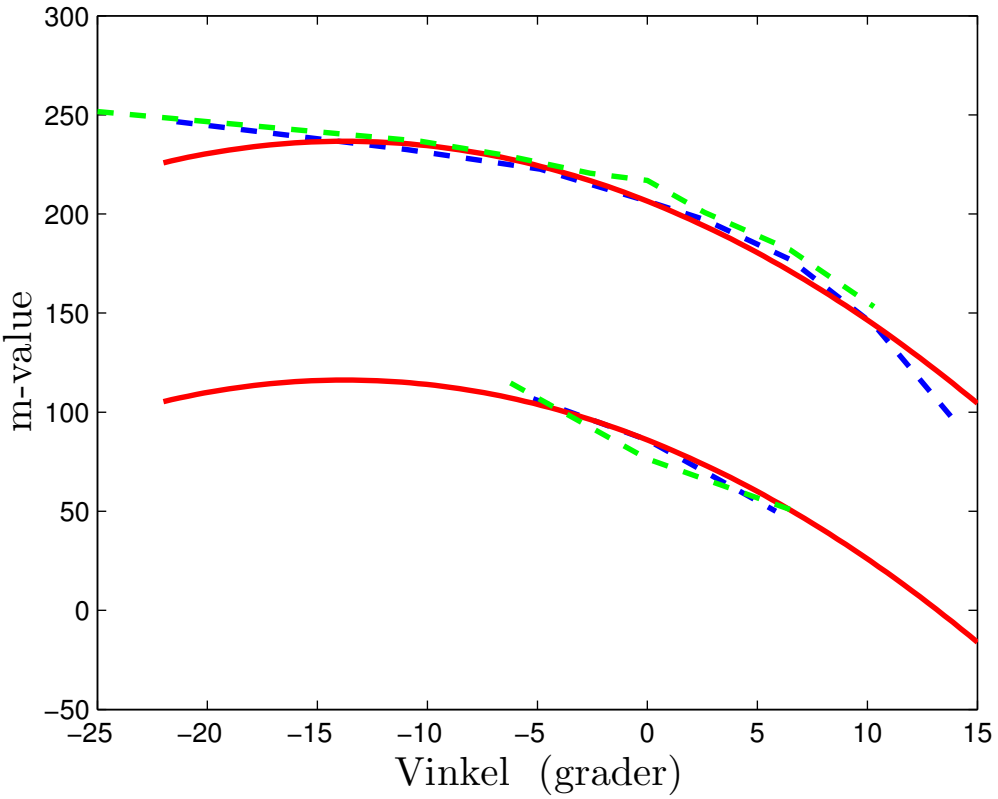
Vinkel kopplat till lutning plottades i MatLab varpå nedanstående logaritmiska funkioner kunde tas fram. De togs fram genom ”trial-error”-metoden och jämfördes med mätdatan.”mhatModify” är också framtagen med ”trial-error”-metoden och är alltså avståndet med hänsyn till vinkelställningen.

$$vinkel = 16 * log(2 * (xhatModify)^2) - 33.9862 \tag{A.1}$$

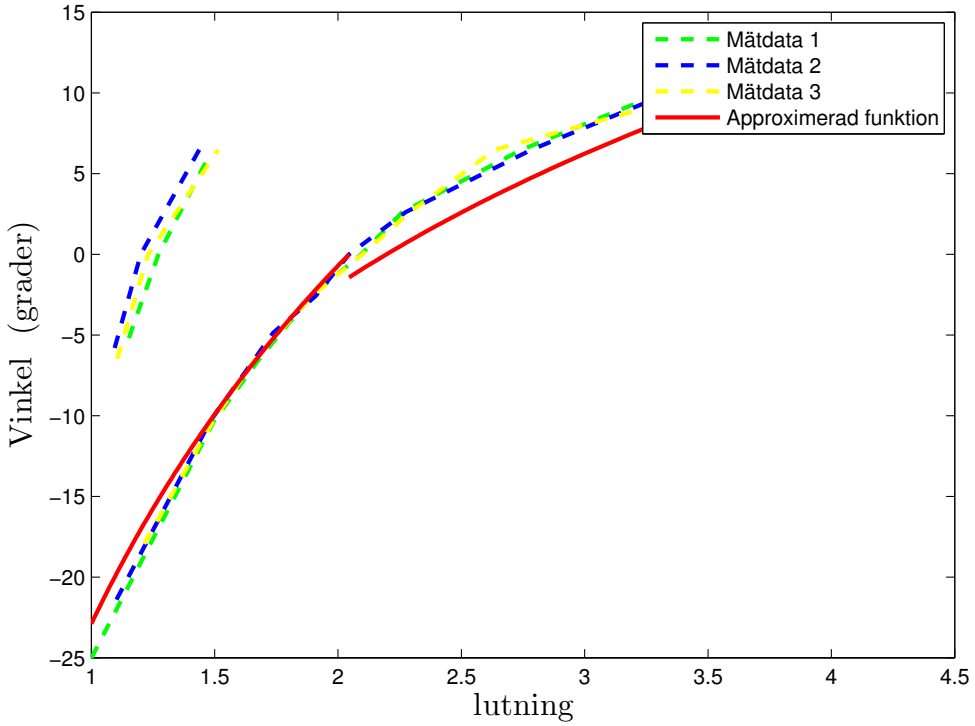
$$vinkel = 10 * log((xhatModify)^2) - 15.7409 \tag{A.2}$$

$$mhatModify = mhat[0] + ((vinkel + 20)/2.5)^2 - 2 * vinkel - 64 \tag{A.3}$$

Där A.1 är för negativa vinklar och A.2 är för positiva. A.3 beräknar vad mhat[0] varit om vinkeln varit 0^o.



Figur A.1: Plot för att ta fram mhatModify. Blåa/Gröna linjer: m-värde beroende på vinkelställning. Röda linjer: mhatModify.



Figur A.2: Plot för att ta fram xhatModify. Blåa/Gröna linjer: Vinkelställning beroende på lutning. Röda linjer: xhatModify.

Funktionen för ”distance” och funktionen inuti ”xhatModify”-funktionen är räta vinkeln mellan två punkter. ”distance” är framtagen från m-värdet hos punkten vid 30cm avstånd och 0^o samt 70cm och 0^o. Denna tar helt enkelt fram avståndet, men tar alltså inte hänsyn till att m-värdet också förändras med vinkelställningen, vilket tas hänsyn till först i ”mhatModify”. Funktionen $-0.019321 * distance + 2.6249$ inuti ”xhatModify” är framtaget från lutningen hos samma punkter som ”distance”. Denna ser alltså till hur det önskvärda värdet på lutningen förändras med avståndet från bandet.

Efter denna framtagna omvandlingsfunktion erhölls mätdatan

Tabell A.5: Mätning 2 - mätdata för vinkel- & avståndsmätningar

Avstånd	vinkel	Uppmätt Avstånd	Uppmätt vinkel	mhat	xhat
30	-17.945	20.833566	-21.734	245.59448242	1.21417701
30	-10.3	29.80295	-10.118	236.16635132	1.49471343
30	-6.48	31.516	-6.01133	228.21482849	1.66572165
30	-2.6	29.69	-2.575510	222.5848999	1.89309549
30	0	28.094	-0.416	213.79699	2.098387
30	2.6	28.0	0.5	208.59321	2.29145622
30	6.48	29.0384	3.5025	184.51455688	2.6360445
30	10.3	22.48	8.715	159.1156311	3.54296851
70	-6.48	66.668	-3.859	116.2508316	1.10443163
70	0	69.7405	-0.74959	90.90525818	1.23010147
70	6.48	75.768	1.755766	55.967453	1.51433945

Då vinkelberoende avståndsmätning ansågs viktigt lades detta till. Framtagning av omvandling finns att finna i Appendix A.2. Mätvärden vid test efter att vinkelberoende avståndsmätning lagts till ser ut som följande.

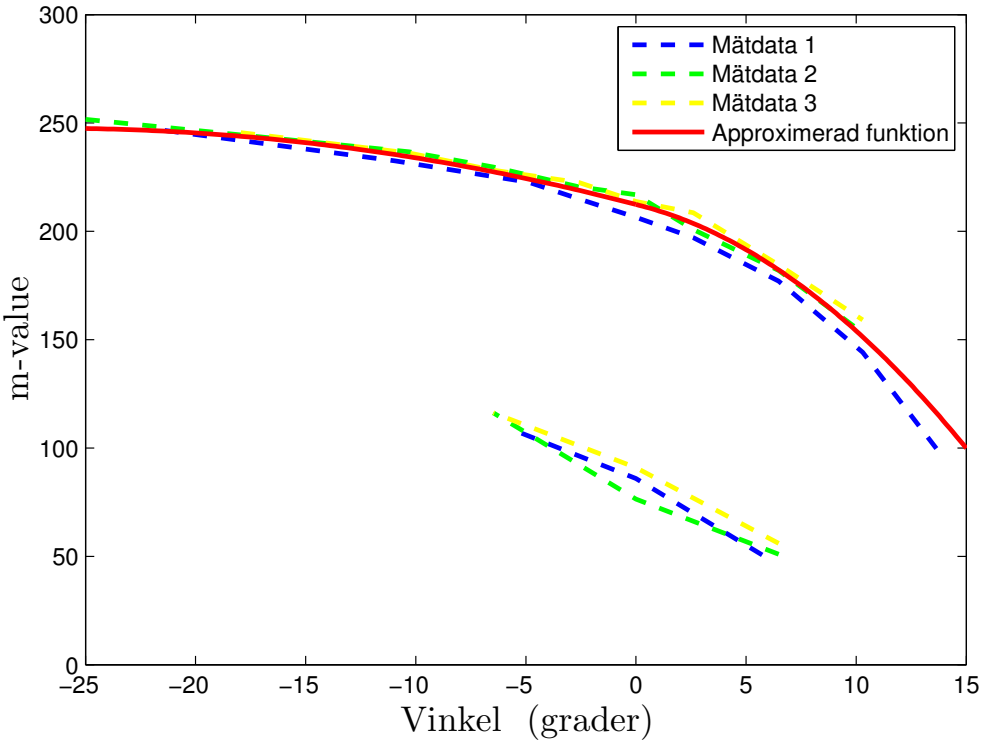
Tabell A.6: Samtliga vinkel- & avståndsmätningar från Mätning 2

Vinkel(^{o})	Uppmätt(^{o})	Fel(^{o})	%	Avstånd(cm)	Uppmätt(cm)	Fel(cm)	%
-17.95	-21.73	-3.78	20.7	30	20.83	-9.17	30.6
-10.3	-10.12	0.18	1.75	30	29.80	-0.2	0.67
-6.48	-6.01	0.47	7.25	30	31.52	1.52	5.07
-2.6	-2.58	0.02	0.77	30	29.69	-0.31	1.03
0	-0.42	-0.42	-	30	28.09	-1.91	6.37
2.6	0.5	-2.1	80.8	30	28.0	-2.0	6.67
6.48	3.50	2.98	46.0	30	29.04	-0.96	3.2
10.3	8.72	1.58	15.3	30	22.48	-7.52	25.1
-6.48	-3.86	2.62	40.4	70	66.67	-3.33	4.76
0	-0.75	-0.75	-	70	69.74	-0.26	0.37
6.48	1.76	-4.72	72.8	70	75.77	5.77	8.24

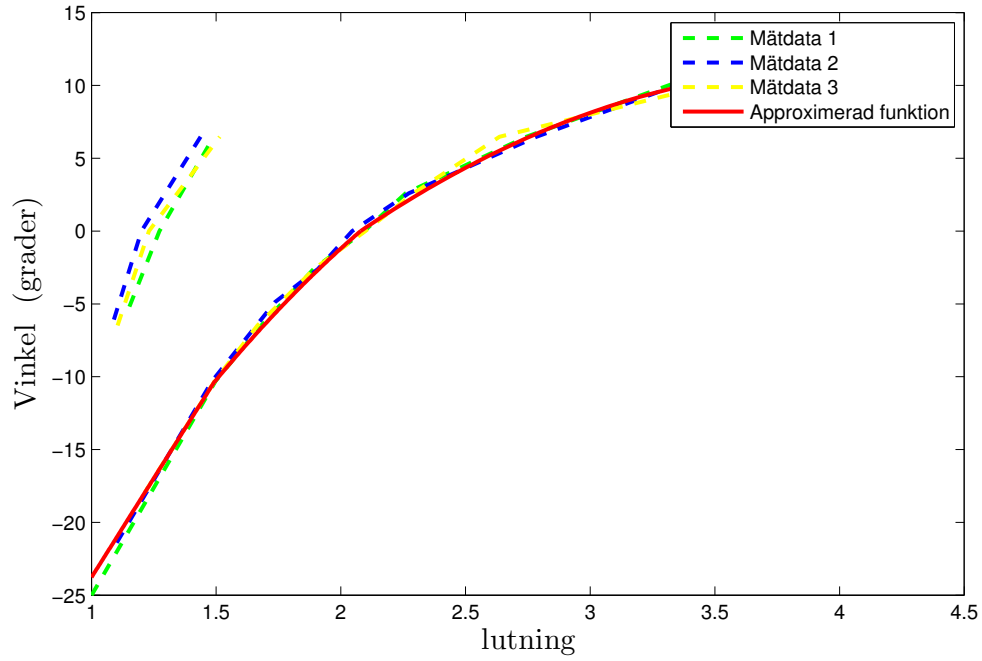
Tabell A.7: Osäkerhet för linjedetektion med kamera från Mätning 2

Typ	Avstånd	Vinkelställning
Genomsnittlig osäkerhet	$\pm 3.00cm$	$\pm 1.78^o$
Osäkerhet vid vinkel 0^o	$\pm 1.91cm$	$\pm 0.42^o$

Efter tillagd vinkelberoende avståndsmätning blir både säkerheten för vinkeln samt avståndet bättre. Dock är funktionen för omvandlingen inte bra nog och valdes att optimera ytterligare. En än mer exakt omvandlingfunktion fram.



Figur A.3: Plot för att ta fram mhatModify. Blåa/Gröna/Gula linjer: m-värde beroende på vinkelställning. Röda linjer: mhatModify.



Figur A.4: Plot för att ta fram xhatModify. Blåa/Gröna/Gula linjer: Vinkelställning beroende på lutning. Röda linjer: xhatModify..

Kod i python för att beräkna vinkel och avstånd:

```
xCorrect = 2.0792 #Onskat varde av xhat[0] vid 30 cm avstand
distance = -0.3126*mhat[0]+96.3961
xhatModify = ((xCorrect - (-0.0211*distance+2.7130)) + xhat[0]) #Vinkeln
    omraknad med avseende p avstandet.
#Formel: (Onskad vinkel vid 30 cm - Onskad vinkel vid specifikt avstand)
#+ nuvarande matvarde for vinkeln
if xhatModify<1.49471343:
    vinkel = 27.2514*xhatModify - 51.0330
elif xhatModify<2.0452:
    vinkel = -(5)*math.pow((xhatModify+1.79),2) + 53.4423*xhatModify
        -36.2636
else:
    vinkel = -(3)*math.pow((xhatModify-2.64),2) + 8.2196*xhatModify
        -16.1466
    #Vinkelutrakning
if vinkel>0:
    mhatModify = mhat[0] + math.pow((vinkel -6.48),2)/3 +6.8204*vinkel -
        13.9968
else:
    mhatModify = mhat[0] + math.pow((vinkel +10),2)/20 +1.6513*vinkel - 5
#Uppdatera varden:
distance = -0.3126*mhatModify+96.3961
xhatModify = ((xCorrect - (-0.0211 * distance + 2.7130)) + xhat[0])
if xhatModify<1.49471343:
    vinkel = 27.2514*xhatModify - 51.0330
elif xhatModify<2.0452:
    vinkel = -(5)*math.pow((xhatModify+1.79),2) + 53.4423*xhatModify
        -36.2636
else:
    vinkel = -(3)*math.pow((xhatModify-2.64),2) + 8.2196*xhatModify
        -16.1466
```

A.3 Pythonkod för linjedetektion

```
#!/usr/bin/env python
#coding=utf-8
#
# license removed for brevity
import numpy as np
import argparse
import cv2
import copy
import matplotlib.pyplot as plt
import math
#import rospy
#from std_msgs.msg import Float32

cap = cv2.VideoCapture('video/30-rak.mp4') #change to cap =
    cv2.VideoCapture(0) for the camera instead (file:
    cv2.VideoCapture('video/30-rak.mp4'))
win = 0
fail = 0

# intial Kalman parameters
#Process noise covariance
Q = 1e-5 # process variance Lutning(Lagre varde = prediktion av stor vikt
    | Hogre varde = matning av stor vikt)
#Default value: 1e-5
Qm = 1e-5 # process variance for m (Lagre varde = prediktion av stor vikt
    | Hogre varde = matning av stor vikt)

# initialize lists for "lutning"
xhat = []#np.zeros(sz) # a posteri estimate of x
P = [] #np.zeros(sz) # a posteri error estimate
xhatminus = [] #np.zeros(sz) # a priori estimate of x
Pminus = [] #np.zeros(sz) # a priori error estimate
K = [] #np.zeros(sz) # gain or blending factor

# initialize lists for "m"
mhat = [] #np.zeros(sz) # a posteri estimate of x
Pm = [] #np.zeros(sz) # a posteri error estimate
mhatminus = [] #np.zeros(sz) # a priori estimate of x
Pmminus = [] #np.zeros(sz) # a priori error estimate
Km = [] #np.zeros(sz) # gain or blending factor

#Measurement noise covariance
R = 0.1 ** 3 # estimate of measurement variance, change to see effect
#Default value: 0.1 ** 2
Rm = 0.1 ** 3 # estimate of measurement variance, change to see effect (m)
```

```
# intial guesses
xhat.insert(0, 0.1)
P.insert(0, 1.0)
mhat.insert(0, 25)
Pm.insert(0, 1.0)
# End of Initial Kalman

# Calc-initial
mcalc = 0
count = 0
xcalc = 0
# Calc-End

#Initiating ROS-stuff
'''dist_Pub = rospy.Publisher('distance', Float32, queue_size=10)
rot_Pub = rospy.Publisher('rotationDeviation', Float32, queue_size=10)
rospy.init_node('positionLinje', anonymous=True)
rate = rospy.Rate(10) # 10hz'''
#End of ROS-stuff

k = 1
while cap.isOpened():
    ret, frame = cap.read()
    frame = cv2.resize(frame, None, fx=0.5, fy=0.5,
        interpolation=cv2.INTER_LINEAR)
    height, width = frame.shape[:2]

    # cropping
    start_row, start_col = int(height * .7), int(width * .5)
    end_row, end_col = int(height), int(width)
    cropped = frame[start_row:end_row, start_col:end_col]
    w, h = cropped.shape[:2]

    #Dilation/Closing kernel
    kernel = np.ones((5,5), np.uint8)

    # Defining the list of boundaries
    boundaries = [
        # ([0, 100, 200], [7, 160, 255]) orange
        #([45, 40, 160], [95, 70, 200]) #Red
        ([4, 14, 120], [95, 85, 200]) # RedExtended
    ]

    # loop over the boundaries
    for (lower, upper) in boundaries:
        # create NumPy arrays from the boundaries
        lower = np.array(lower, dtype="uint8")
        upper = np.array(upper, dtype="uint8")
```

```
        # find the colors within the specified boundaries and apply
        # the mask
        mask = cv2.inRange(frame, lower, upper)
        cv2.imshow('red', mask)
        # output = cv2.bitwise_and(image, image, mask = mask)
        #cv2.imshow('bild', mask)
        dillad = cv2.dilate(mask, kernel, iterations=1)
        closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
        #cv2.imshow('dilated', dillad)
        #cv2.imshow('closed', closing)
        # cv2.waitKey(0)



        # cv2.destroyAllWindows()

    #Find all lines in the picture
    lines = cv2.HoughLinesP(closing, cv2.HOUGH_PROBABILISTIC, np.pi /
        180, threshold=20, minLineLength=1, maxLineGap=50)

    if lines is None:
        lines = cv2.HoughLinesP(dillad, cv2.HOUGH_PROBABILISTIC, np.pi /
            180, threshold=20, minLineLength=1,
                                maxLineGap=50)

    #Find the correct line
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                pts = np.array([[x1, y1], [x2, y2]], np.int32)
                points = np.vstack([pts])
        vx, vy, x0, y0 = cv2.fitLine(np.float32(points), cv2.DIST_L2, 0,
            0.01, 0.01)
        cv2.line(frame, (int(x0 - vx * width), int(y0 - vy * width)),
            (int(x0 + vx * width), int(y0 + vy * width)),(0, 0, 255), 1)
            #Alla linjer typ (ROD)
        lutning = (vy/vx)
        #print('vx: ', vx, 'vy: ', vy, 'x0: ', x0, 'y0: ', y0)
        #m = (y0 - x0 * lutning) #Denna utrakning kan behova ses over! Nu
            kollar den vart linjen slutar i vanstra kant, men hogra hade
            eventuellt varit battre? ev. y0 + (width-x0)*lutning
        m = (y0 + (width - x0) * lutning)
        #print('m: ',m, ' + lutning: ', lutning, 'punkt1:
            ',int(lutning*20+m), 'punkt2: ',int(lutning*300+m), 'width: ',
            width, 'height: ', height)
        #cv2.line(frame, (width, int(m)), (0,
            int(m-lutning*width)),(255,0,0),2) #Alla fitLines (BLA)
        win = win + 1
```

```
        # Kalman: time updateD
        xhatminus.insert(0, xhat[0])
        Pminus.insert(0, P[0] + Q)
        mhatminus.insert(0, mhat[0])
        Pmminus.insert(0, Pm[0] + Qm)

        # measurement update
        K.insert(0, Pminus[0] / (Pminus[0] + R))
        xhat.insert(0, xhatminus[0] + K[0] * (lutning - xhatminus[0]))
        P.insert(0, (1 - K[0]) * Pminus[0])
        Km.insert(0, Pmminus[0] / (Pmminus[0] + Rm))
        mhat.insert(0, mhatminus[0] + Km[0] * (m - mhatminus[0]))
        Pm.insert(0, (1 - Km[0]) * Pmminus[0])
        cv2.line(frame, (width, int(mhat[0])), (0,
            int(mhat[0]-xhat[0]*width)),(0,255,0),2) #Ratta linjen
        #cv2.imshow('fotot', cropped)
        k=0
    else:
        if k < 20:
            cv2.line(frame, (20, int(xhat[0] * 20 + mhat[0])), (width,
                int(xhat[0] * width + mhat[0])), (0, 255, 0), 2)
            #cv2.imshow('fotot', cropped)
        k = k+1
        if k > 20:
            print ('Error occurred')
            #break #Bor eventuellt kommenteras bort vid reell korning.


    #Output: (Optimerad for 30cm fran bandet)
    xCorrect = 2.0452 #Onskat varde av xhat[0] vid 30 cm avstand
    distance = -0.331*mhat[0]+98
    xhatModify = ((xCorrect - (-0.019321*distance+2.6249)) + xhat[0])
        #Vinkeln omraknad med avseende pa avstandet. Formel: (Onskad
        vinkel vid 30 cm - Onskad vinkel vid specifikt avstand) +
        nuvarande matvarde for vinkeln
    #print('Ratt: ', -0.01603*distance+2.83285,'xCorrect: ', xCorrect,
        'xhat[0]: ', xhat[0], 'xhatModify: ', xhatModify)
    if xhatModify<2.0452:
        vinkel = 16*math.log(2*math.pow(xhatModify,2))-33.9862
    else:
         vinkel =
            10*math.log(math.pow(xhatModify,2))-15.7409#14,25*xhatModify-29.15
            #100/math.exp(xhatModify)-9.7 #Vinkelutrakning
    mhatModify = mhat + ((vinkel + 20) / 2.5) ^ 2 - 2 * vinkel - 64
    #Uppdatera varden:
    distance = -0.331*mhatModify+98
    xhatModify = ((xCorrect - (-0.019321 * distance + 2.6249)) + xhat[0])
    if xhatModify<2.0452:
        vinkel = 16*math.log(2*math.pow(xhatModify,2))-33.9862
```

```
    else:
        vinkel = 10*math.log(math.pow(xhatModify,2))-15.740
    print('Avstand fran band: ', distance, ' centimeter. | Grader fran
        optimal kurs: ', vinkel, ' grader. (pos = pekar bort fran bandet,
        neg = pekar mot bandet)')

    vink = "%.2f" % vinkel
    dist = "%.2f" % distance #"{:.5f}".format(distance)
    #cv2.putText(frame, ("Avstand: ", distance, "cm. Vinkelstallning: ",
        vinkel, "grader"), (20, 250), cv2.FONT_HERSHEY_SIMPLEX, 2,
        (0,255,0), 2)
    cv2.putText(frame, ("Distance: " + dist +"cm."), (20, 210),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    cv2.putText(frame, ("Angular position: " + vink + "degrees."), (20,
        230), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    cv2.imshow('full', frame)

    #Grader till radianer
    radianer = vinkel*math.pi/180

    #dist_Pub.publish(distance)
    #rot_Pub.publish(radianer)
    #rate.sleep()

    mcalc = mcalc + mhat[0]
    count = count + 1
    xcalc = xcalc + xhat[0]

    if cv2.waitKey(1) == 13: # 13 is the Enter Key ; Ett annat krav for
        avbrott kan vara rekommenderat vid reell korning.
        print('mTot: ',mcalc/count,'xTot: ',xcalc/count)
        break
print(win*100/(win+fail), 'procent funnet!')
cap.release()
cv2.destroyAllWindows()
```

A.4 Matlabkod för omvandlingsfunktion för linjedetektion med kamera

```
%% Ny Avstandsmatning 1.0
clear all
clc
X=[-21.4,-10.93,-4.87,-2.6,0.0,2.6,6.48,10.3,14.03];
Y=[246.5733,232.467,222.605,214.952,206.452,197.1305,177.02446,144.124481,94.6845];
Z=[-5.19,0,5.84];
A=[106.8285,85.95,49.957];
B=linspace(0,15,100);
C=2000./log(B-15)-80;
mhat=0;
%D=270.4520-((B+20)./2.5).^2+2*B;
D=276.3963-((B+20)./2.5).^2+2*B;
H=149.9532-((B+20)./2.5).^2+2*B;
212.3963-(276.3963-((0+20)./2.5).^2+2*0);
% avstand
k=(-40)/(270.452-149.9532);
G=0;
F=270.4520-((G+20)./2.5).^2+2*G;
E= -0.3305785*F+98;
% Find the coefficients.
mhatValue = (206.452+216.94+213.79699)/3;
mhatValue70 = (85.95+76.4625+90.90525818)/3;
distanceK=((30-70)/(mhatValue-mhatValue70));
distanceM=(70-distanceK*mhatValue70);
tenValue = (144.124481+153.2+159.1156311)/3;
extralutning = 2;
k = ((251.68-mhatValue+extralutning)/(-25-0));
Xm = linspace(-25,0,100);
mTen = (1/20)*(0+10).^2 -k*0 +mhatValue;
approx = -(1/20)*(Xm+10).^2 + k*Xm + mTen;

extralutningP = 10;
kp = ((mhatValue-tenValue+extralutningP)/(0-10.3));
Xp = linspace(0,15,100);
mmTen = (1/3)*(0-6.48).^2 -kp*0 +mhatValue;
approxP = -(1/3)*(Xp-6.48).^2 + kp*Xp + mmTen;

% Nya varden
vinkel=[-25,-10.3,-6.48,-2.6,0,2.6,6.48,10.3];
mhat=[251.68,236.68,229.65,220.54,216.94,200.8,181.9,153.2];
vinkel70=[6.48,0,-6.48];
mhat70=[50.944,76.4625,116.27];
% Varden Test 2
vinkel2=[-17.945,-10.3,-6.48,-2.6,0,2.6,6.48,10.3];
mhat2=[245.59448242,236.16635132,228.21482849,222.5848999,213.79699,208.59321,184.5145568
```

```
vinkel702=[6.48,0,-6.48];
mhat702=[55.967453,90.90525818,116.2508316];

%Plottar
plot(X,Y,'--','linewidth', 2);
hold on;
plot(vinkel,mhat,'g--','linewidth', 2);
plot(vinkel2,mhat2,'y--','linewidth', 2);
%plot(B,H,'r','linewidth', 2);
plot(Xm,approx,'r','linewidth', 2);
plot(Xp,approxP,'r','linewidth', 2);
plot(Z,A,'--','linewidth', 2);
hold on;
%plot(B,D,'r','linewidth', 2);
hold on


plot(vinkel70,mhat70,'g--','linewidth', 2);
hold on
plot(vinkel702,mhat702,'y--','linewidth', 2);
fontSize=15;
xlabel(['Vinkel \ (grader)'],'interpreter','latex','fontsize',fontSize)
ylabel(['m-value'],'interpreter','latex','fontsize',fontSize)
%set(gca,'TickLabelInterpreter','latex','fontsize',17);
h=legend('Mtdata 1','Mtdata 2','Mtdata 3','Approximerad funktion');
%set(h,'Interpreter','latex');


%% Avstandsberoende vinkel | Ny Vinkel 1.0 (Vinkeln beroende p xhat)
close all
clc
%Test 2
vinkel2=[-17.945,-10.3,-6.48,-2.6,0,2.6,6.48,10.3];
xhat2=[1.21417701,1.49471343,1.66572165,1.89309549,2.098387,2.29145622,2.6360445,3.542968
vinkel702=[-6.48,0,6.48];
xhat702=[1.10443163,1.23010147,1.51433945];
%Ngt test
vinkel1=[-21.4,-10.3,-4.87,-2.6,0.0,2.6,6.48,10.3,14.03];
xhat1=[1.10156,1.48358,1.73669,1.90813,2.0452,2.27296,2.784744,3.391929,4.279761];
vinkel701=[-5.19,0,5.84];
xhat701=[1.152076,1.27236,1.468115];
%Test 1
vinkel=[-25,-10.3,-6.48,-2.6,0,2.6,6.48,10.3];
xhat=[1.0,1.4979,1.6809,1.891,2.094,2.26,2.7443,3.362];
vinkel70=[6.48,0,-6.48];
xhat70=[1.4376,1.200,1.0823];
%Koefficient
xhatValue=(2.098387+2.0452+2.094)/3
xhatValue70=(1.200+1.23010147+1.27236)/3
```

```
kVinkel = (xhatValue-xhatValue70)/-40
mVinkel = xhatValue-(30*kVinkel)
%Aproximerad
Xp = linspace(2.0452,4.4,100);
Xm = linspace(1.495,2.0452,100);
Xmm = linspace(1.0,1.495,100);
vinkelm = 16*log(2*Xm.^2)-33.9862;
vinkelp = 10*log((Xp.^2))-14.6397;
vinkelmm = (-10.3-(-17.945))/(1.49471343-1.21417701)*Xmm -
    (((-10.3-(-17.945))/(1.49471343-1.21417701))*1.49471343 - (-10.3));
Fel = (10*log((2.0792.^2))-14.6397)%14.3099
kmm = (-10.3-(-17.945))/(1.49471343-1.21417701);
mmm = (((-10.3-(-17.945))/(1.49471343-1.21417701))*1.49471343 - (-10.3));
%Ny Aproximerad
tenValue = (3.362+3.391929+3.54296851)/3;
extralutningP = 0.1;
kP = ((0-10.3)/(xhatValue-tenValue+extralutningP))
XpNy = linspace(xhatValue,4.3,100);
mTen = (3)*(xhatValue-2.64).^2 -kP*xhatValue +0
approxP = -(3)*(XpNy-2.64).^2 + kP*XpNy + mTen;

tenMValue = (1.49471343+1.48358+1.4979)/3;
extralutning = 0.3944;
k = ((0+10.3)/(xhatValue-tenMValue-extralutning))
XmNy = linspace(1.495,xhatValue,100);
mMTen = (5)*(xhatValue+1.79).^2 -k*xhatValue +0
approxM = -(5)*(XmNy+1.79).^2 + k*XmNy + mMTen;
%Gammal Aproximation
%GXp = linspace(2.0452,4.4,100);
%GXm = linspace(1.0,2.0452,100);
%Gvinkelm = 16*log(2*GXm.^2)-33.9862;
%Gvinkelp = 10*log((Xp.^2))-15.7409;
%Plottar
plot(xhat,vinkel,'g--','linewidth',2);
hold on
plot(xhat1,vinkel1,'--','linewidth',2);
plot(xhat2,vinkel2,'y--','linewidth',2);
%plot(Xm,vinkelm,'r','linewidth',2);
plot(XmNy,approxM,'r','linewidth',2);
plot(XpNy,approxP,'r','linewidth',2);
%plot(Xp,vinkelp,'r','linewidth',2);
plot(Xmm,vinkelmm,'r','linewidth',2);
plot(xhat70,vinkel70, '--','linewidth',2);
plot(xhat701,vinkel701, 'g--','linewidth',2);
plot(xhat702,vinkel702, 'y--','linewidth',2);
fontSize=15;
xlabel(['lutning'],'interpreter','latex','fontsize',fontSize)
ylabel(['Vinkel \ (grader)'],'interpreter','latex','fontsize',fontSize)
%set(gca,'TickLabelInterpreter','latex','fontsize',17);
```

```
143 h=legend('Mtdata 1','Mtdata 2','Mtdata 3','Approximerad funktion');
144 %set(h,'Interpreter','latex');
145
146 %% Osakerhetsutrakning
147 Avstandssakerhet = (9.17+0.2+1.52+0.31+1.91+2+0.96+7.52+3.33+0.26+5.77)/11
148 Vinkelsakerhet =
        (3.78+0.18+0.447+0.02+0.42+2.1+2.98+1.58+2.62+0.75+4.72)/11
149 Avstandssakerhet3 = (2.83+0.18+0.17+0.21+0.10+0.07+2.15+2.33)/8
150 Avst70 = (6.61+3.46+2.24)/3
151 Vinkelsakerhet = (0.19+0.16+0.26+.043+0.27+0.73+1.63+1.46)/8
152 Vinkel70 = (1.91+0.07+3.27)/3
153 TotAvst = (2.83+0.18+0.17+0.21+0.10+0.07+2.15+2.33+6.61+3.46+2.24)/11
154 Totvinkl = (0.19+0.16+0.26+.043+0.27+0.73+1.63+1.46+1.91+0.07+3.27)/11
155 minAvst = (0.43+0.27+0.73)/3
156 minVinkl = (0.21+0.10+0.07)/3
```

A.5 Pythonkod för fotgängardetektering

```
import cv2
import rospy
from std_msgs.msg import Bool

body_classifier =
    cv2.CascadeClassifier('./Haarcascades/haarcascade_lowerbody.xml')
cap = cv2.VideoCapture(0)
pub2 = rospy.Publisher('stop', Bool, queue_size=10)
rospy.init_node('pedestrianStop', anonymous=True)
rate = rospy.Rate(10) # 10hz


class Stack:
     def __init__(self):
         self.items = []
         for i in range(10):
            self.items.append(0)

     def __str__(self):
         return str(self.items)

     def isEmpty(self):
         return self.items == []

     def push(self, item):
         self.items.append(item)

     def pop(self):
         return self.items.pop(0)
```

```
    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)

    def sum(self):
        return sum(self.items)


count = Stack()
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640,480))

while True:
    ret, frame = cap.read()
    frame = cv2.resize(frame, None, fx=1, fy=1,
        interpolation=cv2.INTER_LINEAR)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    bodies = body_classifier.detectMultiScale(gray, 1.2, 4)
    if len(bodies) > 0:
        count.push(1)
        count.pop()
    else:
        count.push(0)
        count.pop()

    if count.sum() >= 3:
        pub2.publish(True)
        cv2.putText(frame, "stop", (50,50), cv2.FONT_HERSHEY_SIMPLEX, 2.0,
            (0,255,0),3)




    for (x, y, w, h) in bodies:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)

    cv2.imshow('Pedestrians', frame)
    out.write(frame)

    rate.sleep()
    if cv2.waitKey(1) == 13:
        break
```

```
cap.release()
cv2.destroyAllWindows()
```

A.6 Pythonkod för avstånd till blå cylindrar

```
#!/usr/bin/env python
# coding=utf-8
import numpy as np
import cv2
import rospy
from std_msgs.msg import Float64

def distance_to_camera(knownWidth, focalLength, perWidth):
    # compute and return the distance from the maker to the camera
    return (knownWidth * focalLength) / perWidth


cap = cv2.VideoCapture(0)
# take first frame of the video
ret,frame = cap.read()
# setup initial location of window
r,h,c,w = 0,480,0,640 # simply hardcoded the values
track_window = (c,r,w,h)
# set up the ROI for tracking

hsv_roi = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((95., 100.,100.)),
    np.array((130.,255.,255.)))
roi = frame[r:r+h, c:c+w]
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
print(roi_hist)
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
# Setup the termination criteria, either 10 iteration or move by atleast
    1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )


KNOWN_DISTANCE = 1
KNOWN_HEIGHT = 0.21
FOCAL_LENGTH = 700

dist_Pub = rospy.Publisher('distSide_camshifttrack', Float64,
    queue_size=10)
rospy.init_node('camDist_Side', anonymous=True)
rate = rospy.Rate(10) # 10hz

#fourcc = cv2.VideoWriter_fourcc(*'XVID')
#out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640,480))


while(1):
```

```
    ret ,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
        # apply meanshift to get the new location
        ret, track_window = cv2.CamShift(dst, track_window, term_crit)
        if track_window[3] < 10:
         track_window = (c,r,w,h)
         pass
        # Draw it on image
        pts = cv2.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv2.polylines(frame,[pts],True, 255,2)
        distance = distance_to_camera(KNOWN_HEIGHT, FOCAL_LENGTH,
            track_window[3])
        cv2.putText(img2, str(distance), (50,50),
            cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0,255,0),3)
        cv2.imshow('img2',img2)

        #out.write(img2)


        distance = distance_to_camera(KNOWN_HEIGHT, FOCAL_LENGTH,
            track_window[3])-0.64
        #print(distance)

        dist_Pub.publish(distance)
        rate.sleep()


        if cv2.waitKey(1) == 13: # 13 is the Enter Key
            break

    else:
            break

cv2.destroyAllWindows()
cap.release()
```