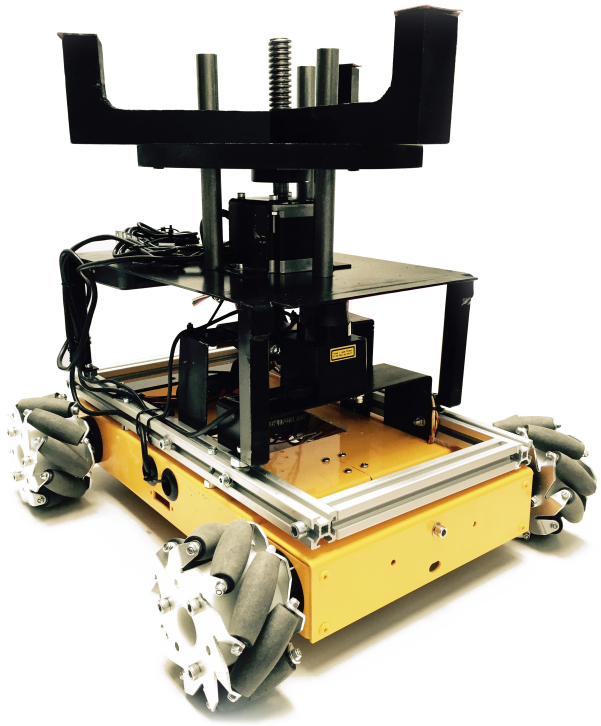
CHALMERS



Autonom lagerhantering

Lösningar för den moderna lagermiljön

Kandidatarbete inom civilingenjörsprogrammet

Victor Andersson
Gunnar Bolmvall
Daniel Eriksson
Jonathan Hasselström
Pedro Josefsson
Daniel Pettersson

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

Autonom lagerhantering

Lösningar för den moderna lagermiljön

-Kandidatarbete inom civilingenjörsprogrammet-

Victor Andersson
Gunnar Bolmvall
Daniel Eriksson
Jonathan Hasselström
Pedro Josefsson
Daniel Pettersson

Handledare: Martin Fabian
Examinator: Knut Åkesson

Kandidatarbete SSYX02-15-17
Autonom lagerhantering, lösningar för den moderna lagermiljön

Chalmers tekniska högskola
SE-412 96 Göteborg, Sverige
Telefon +46(0) 31-772 1000
Fax 031-772 5944
chalmers@chalmers.se

Tryck: Signaler och system
Omslag: Foto på AGV

Förord

Följande rapport beskriver kandidatarbetet *Autonom lagerhantering*. Syftet med arbetet var att utveckla ett autonomt lagerhanteringssystem och att ta fram en fungerande prototyp. Projektet har utförts på Chalmers tekniska högskola vid institutionen för Signaler och System. Projektet omfattar 15 högskolepoäng och genomfördes vårterminen 2015. Vi vill tacka alla som varit inblandade i projektet och särskilt tack riktas till vår handledare Martin Fabian. Stort tack riktas även till Janne Bragée och Reine Nohlborg i Prototyplabbet som varit behjälpliga under framtagningen av projektets prototyp.

Abstract

In order to create an efficient warehouse management system and minimize the related work injuries, systems with an ever-increasing degree of automation is pursued.

This report adresses the development of an autonomous warehousing system and an associated prototype with the aim to facilitate the work and reduce the economic and social costs associated with conventional methods.

An AGV (Automated Guided Vehicle) was developed and with the help of a supervisory system it's made possible to retrieve and return warehouse furnishing autonomously, following given orders. A grid of QR codes has been established in a test area, which the AGV can scan and thereby obtain position feedback. The route planning, provided by the supervisory system, is based on the A* algorithm that calculates an optimal route for the AGV to reach its target destination. The supervisory system also supports multiple AGVs and thereby enabling adaptation to different sizes of storage systems.

In a warehouse environment, unpredictable elements such as people and misplaced items can appear and therefore a collision avoidance system has been implemented. A laser sensor is placed at the forefront of the AGV, thus enabling obstacle detection in the direction of travel. In order to be able to move the warehouse furnishing the AGV is equipped with a mechanical lifting device which makes it possible to transport articles.

Sammanfattning

För att skapa effektiva lagerhållningssystem och minimera de relaterade arbetsskadorna eftersträvas en ständigt ökande automatiseringsgrad av nämnda system. Rapporten behandlar utvecklingen av ett autonomt lagerhållningssystem och en tillhörande prototyp med syftet att underlätta arbetet samt minska de ekonomiska och sociala kostnader förknippade med konventionella metoder.

En AGV (Automated Guided Vehicle) har utvecklats, som tillsammans med ett överordnat system klarar att hämta och lämna material autonomt efter given order. I lagermiljön har ett rutnät av QR-koder upprättats, vilka AGV:n kan avläsa och därigenom bestämma sin position. Ruttplaneringen, som sker i det överordnande systemet, görs med algoritmen A* som beräknar den optimala rutten för AGV:n. Det överordnade systemet har stöd för flera AGV:er och därigenom möjliggörs anpassning till olika storlekar på lagersystem.

I en lagermiljö förekommer oförutsägbara element såsom människor och felplacerade objekt, därför har ett kollisonsundvikningssystem implementerats. En lasersensor placeras i framkanten av AGV:n och därmed möjliggörs avskanning av området i AGV:ns färdriktning och följaktligen kan eventuella hinder upptäckas. För att kunna förflytta lagerinredningen utrustas AGV:n med en mekanisk lyftanordning.

Beteckningar

AGV	Automated guided vehicle.
QR-kod	Quick Response-kod.
CATIA	Program för att digitalt skapa kontruktioner och ritningar.
JavaScript	Prototypbaserat Scriptspråk som används främst i webbtillämpningar.
HTML5	Senaste standarden för HTML, vilket används för skapandet av webbsidor.
Kompilator	Datorprogram som skapar ett lågnivåprogram utifrån en programtext.
Kommandotolk	Gränssnitt där användare kan köra kommandon eller enkla program.
Netcat	Tjänst för att läsa/skriva till nätverksanslutningar.
API	Specificerar hur olika applikationsprogram kan använda specifik programvara.
Full-duplex	Beskriver att kommunikation kan ske i båda riktningar samtidigt.
Heuristik	Procedur som ger tillfredställande men ofullständiga svar.
Encoder	Enhet som omvandlar information från ett format till ett annat.

Innehåll

Kapitel 1

Inledning

För att öka effektiviteten och säkerheten samt minska kostnaden för lagerhantering har institutionen för Signaler och System på Chalmers skapat kandidatarbetet *SSYX02-15-17, Autonom lagerhantering*. Arbetet syftar till att skapa en prototyp för en automatiserad lösning för materialhantering i lagermiljö. Denna rapport redogör utförande samt resultat av nämnt kandidatarbete som genomförts under vårterminen 2015. Som grund till arbetet finns tidigare års kandidatarbeten, där syftet var att autonoma AGV:er skulle transportera lagerinredning åt ett överordnat system [16, 12].

1.1 Bakgrund

Att hushålla med resurser är en viktig del av företagandet oavsett om det rör sig om pengar, ytor, personal eller miljön. Inom tillverkningsindustrin tvingas företag ofta att lagerhålla de producerade produkterna vilket skapar stora lager som binder kapital och tar upp stor platsyta. Detta görs dels för att företag ska kunna hålla höga servicenivåer men även för att den ständigt föränderliga marknaden är svår att förutspå. Stora svenska företag arbetar ofta aktivt med att minska dessa kostnader genom att utarbeta effektiva logistiklösningar, exempelvis via ”Supply chain management” [20]. Problemet med ”Supply chain management” är att fokus ligger på in- och utflöden från lager och fabriker som behandlas. Mindre hänsyn tas således till materialflödena inom lagret.

Oftast sker materialhanteringen i lagret manuellt med hjälpmedel som truckar, pallastare och vagnar. Hos vissa företag används ibland paternosterverk[1], men

[1]Paternosterverk syftar här till en automatiserad lagerhylla.

dessa är ofta inte flexibla gällande nya lagerlösningar och är begränsade till mindre artiklar [20].

Konsultbolaget Boston Consulting Group menar att det finns möjligheter att minska kostnaderna inom industrin med 16% genom en ökning av automatiseringen under de kommande 10 åren [14]. Med tanke på den stora andelen manuellt arbete i materialhanteringen finns det kostnadsminskningar att göra även här.

Utöver materialhanteringskostnaderna tillkommer också andra sociala kostnader i form av arbetsolyckor. Nästan 800 personer skadas och sjukskrivs från jobbet varje år i truckrelaterade olyckor, vilket är den vanligaste typen av arbetsolyckor i Sverige [10]. Genom att minimera människors närvaro i lagerhållningsområdet är förhoppningen att arbetet blir betydligt säkrare än med konventionella lagerhållningssystem.

En uppmärksammad automationslösning för lagerhantering, som projektet hämtat inspiration ifrån, har utvecklats av företaget Kiva Systems. Det baseras på ett integrerat system där operatörer, AGV:er och lagerhyllor samspelar. Kortfattat utgörs systemet av flexibla lagerplatser där ett överordnat system övervakar och håller koll på inventarier. Under det överordnade systemet verkar individuella AGV:er med uppgift att transportera lagerhyllorna mellan operatörerna och lagerplatsen. Resultatet är ett lagerhanteringssystem med minimerad olycksrisk då personalen befinner sig vid plockstationer och AGV:erna sköter transporten av lagerinredningen. Detta system möjliggör även en flexiblare användning av lagerytorna då utrymmet lätt kan omorganiseras [3].

1.2 Syfte

Avsikten med projektet är att genom vidareutveckling av föregående års kandidatarbeten utarbeta och implementera en lösning för autonom materialhantering med hjälp av AGV:er. Via ett överordnat system ska AGV:er koordineras till att transportera förutbestämd lagerinredning till bestämda platser för att till exempel en montör ska slippa hämta detta själv. Systemet ska vara möjligt att implementera i ett lagersystem med flera AGV:er och varje AGV utrustas med ett kollisionsundvikningssystem för att undvika hinder. Målet är att öka effektiviteten och på ett effektivt och tillförlitligt sätt minimera kostnader samt reducera personskador och höja ergonomin i anknytning till materialhantering i lagermiljöer.

1.3 Avgränsningar

För att kunna hålla projektet inom given tidsram måste avgränsningar i projektuppgiften göras. Till en början avgränsas arbetet till att enbart behandla och framställa en enda AGV eftersom det är vad gruppen har att tillgå i projektet. Detta skall dock inte begränsa det överordnade systemet, som bör konstrueras för att hantera fler AGV:er.

På liknande sätt kommer lagermiljön utformas så att den endast innehåller en operatör. Detta för att spara tid samt minimera och fokusera uppgiften på att först få den huvudsakliga funktionen, hämta och lämna lagerinredning, att fungera korrekt. Här beslutas det att fortsätta med föregående års tanke med att lagerinredningen skall ha en förutbestämd struktur och design.

För att optimera ruttplaneringen antas den tillgängliga arbetsmiljön vara känd för det överordnade systemet. Miljön avgränsas också till att verka i ett plan och ordnas så att det ej existerar hinder på höjder som inte kan detekteras.

Två andra områden som projektet ej behandlar är felplacering av lagerinredning samt lagersaldohållning. Felplacering av lagerinredning innebär alltså att lagerinredningen förflyttas manuellt vilket gör att systemet tappar kontrollen på var de befinner sig i lokalen. Det förutsätts här att all personal är väl införstådd med systembegränsningen. Företag har ofta redan en extern saldohållningsfunktion vilket gör utvecklandet av en sådan funktion inte innefattas i projektet.

1.4 Problemformulering

De huvudsakliga målen för projektet är enligt följande att AGV:n skall:

- Autonomt navigera mellan kända positioner.
- Ta order om att utföra leveranser från ett överordnat datorsystem.
- Undvika att kollidera med människor och annan rörlig utrustning inklusive andra AGV:er.
- Utrustas med ett system för att automatiskt hämta och lämna lagerinredning.

Det överordnade systemet skall också hantera flera AGV:er. Prioritet sätts på att primärt klara de nämnda målen och därefter addera ytterligare funktionalitet samt optimering. För att få en överblick över alla de problem som definierats har följande delmål upprättats:

1. Lagerinredningen skall vara utformad så att:
   (a) Den kan transporteras av AGV:n utan att varor tappas eller går sönder.
2. Plockstationen skall:
   (a) Ge order till det överordnade systemet om vad som skall hämtas via ett lämpligt gränssnitt.
   (b) Godkänna att rätt order blivit plockad så att AGV:n kan återlämna lagerhyllan.
3. Navigationssystemet skall:
   (a) Möjliggöra för AGV:n att bestämma sin position.
   (b) Vara grund till ett kommersiellt gångbart system.
4. AGV:n skall:
   (a) Navigera till rätt plats.
   (b) Möjliggöra transportera av lagerhyllan.
   (c) Kontrollera att rätt lagerhylla hämtas.
   (d) Undvika kollision med rörliga och stationära hinder.
   (e) Placera lagerhyllan på angiven plats.
5. Överordnade systemet skall:
   (a) Hålla koll på vilken hylla som skall hämtas.
   (b) Ruttplanera AGV:ns väg till och från målet.
   (c) Planera AGV:ernas rutt så att kollisioner undviks .
   (d) Tillhandahålla ett gränssnitt som möjliggör orderhantering och uppdatera användaren med status av orderhämtning.
   (e) Hålla reda på lagerhyllornas position.

En kravspecifikation upprättades även för att kvantifiera målen i projektet, vilken kan hittas i appendix A. Under arbetets gång gjordes kontinuerlig återkoppling till kravspecifikationen för att säkerställa att alla krav verkligen uppnåtts.

1.5 Rapportstruktur

För att uppnå de uppsatta målen delades arbetet upp i mindre, mer konkreta, delsystem. Det som föreföll mest naturligt vid projektstart var att dela upp projektet i fyra delsystem. Dessa är navigering, överordnat system, kollisionsundvikning samt anordning för transport av material och därför återfinns dessa delsystem som enskilda kapitel i rapporten. Inledningsvis ges en inventariebeskrivning som efterföljs av de fyra delsystemkapitlen som beskriver först det mer teoretiska underlaget för att sedan gå in på implementeringen av arbetet, det vill säga en konkretisering av vad som gjorts. Innan resultat och diskussion, hittas ett kapitel som beskriver hur delsystemen interagerar med varandra och ger läsaren en bättre överblick över hela arbetet.

Kapitel 2

Lagermiljö och robot

Hur lagermiljön ser ut och vilken hårdvara som används påverkar självklart det resultat som kan uppnås av projektet. För att ge läsaren en tydlig bild av de delar som ingår i det slutgiltiga resultatet följer nedan en beskrivning av hur lagermiljön är utformad samt vilka komponenter som ingår i AGV:n.

2.1 Lagermiljö

Som beskrivs i kapitel 1.3, har flera avgränsningar gjorts med avseende på lagermiljön. Utöver dessa bör lagermiljön även vara utformad på ett standardiserat sätt för att minska antalet scenarion som AGV:n kan utsättas för. Därför upprättas, utöver de avgränsningar som redan gjorts, riktlinjer kring hur lagret ser ut.

2.1.1 Lagerinredning

AGV:n skall transportera varor i ett lager. I kapitel 6 beskrivs den anordning som används för uppgiften. Lösningen liknar den som Kiva systems använder idag [3]. I projektet används ett kvadratiskt bord med sidan 55 cm och höjden 45 cm som väger drygt 3 kg för att enkelt, snabbt och billigt efterlikna någon typ av lagerinredning med produkter, se figur 2.1.

Denna inredning används inte vanligen i lagermiljöer men den utgör en god grund för att testa funktionaliteten i projektet. Vid eventuell vidareutveckling finns här stor potential för att effektivisera systemet samt öka dess användarvänlighet, se kapitel 9.

Figur 2.1: Bordet som används som lagerinredningen i projektet [21]

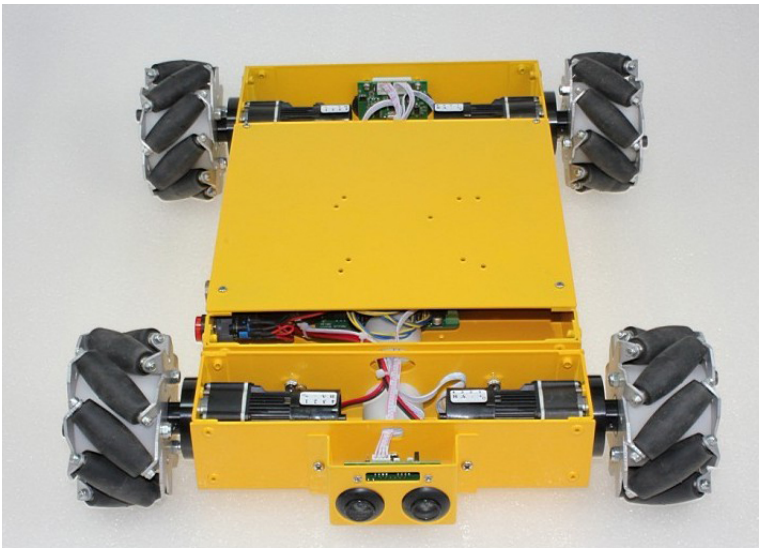
2.1.2 Rutnät

Metoden som valts för att återkoppla AGV:ns position i rummet sker via ett rutnät av QR-koder(Quick response) på golvet, se avsnitt3.2.4. För att AGV:n ska klara av att tolka information från rutnätet förses den med en kamera som kan läsa av QR-koder. Punkterna i rutnätet utgörs av QR-koder som innehåller information om punktens koordinater. Punkterna befinner sig 80 cm från varandra och utifrån koderna kan även AGV:ns vinkel i förhållande till koden utläsas. I kapitel 3 beskrivs beslutet av navigationssystem.

2.2 Hårdvarubeskrivning

Tillgängligt för projektet finns en i figur 2.2 illustrerad Nexus *4WD mecanum wheel mobile robot kit 10011* som är utrustad med fyra ultraljudssensorer, en mikrokontroller(Arduino), ett aluminiumchassi samt fyra mecanumhjul med tillhörande 12 V motorer [1]. Nettovikten för denna grundstomme är drygt 4 kg. Utöver Nexus

roboten och dess tillbehör används också i projektet en router, server, styrenhet(myRIO), kamera, batteripack, anordning för transport av material och lasernavigationsutrustning. En kortfattad beskrivning följer nedan för dessa delar bortsett från anordningen för transport av material som istället beskrivs under kapitel 6.



Figur 2.2: Nexus 4WD Mecanum wheel mobile robot kit 10011

2.2.1 Chassi

Chassit är tillverkat av aluminiumlegering och är både lätt och starkt med dimensionerna 380mm x 215mm x 50mm. Metallen är två millimeter tjock vilket anses tillräckligt för att fästa utrustning på.

2.2.2 Arduino

För att kunna länka mjukvaran med hårdvaran används en Arduino mikrokontroller som är sammankopplad med hjulmotorerna vilket gör att AGV:n kan styras enligt önskemål. För att möjliggöra styrning av robotens alla fyra motorer samtidigt är Arduino:n utrustad med ett Arduino IO Expansion Board. Arduino:n som används är av modellen Deumilanove 328 och den styr utöver hjulmotorerna även den motor som höjer och sänker lyftstycket. Den största begränsningen i Arduino:n är att den endast har en port för seriell kommunikation. Ursprungligen användes denna port för att interagera med ultraljudssensorerena, men för att kommunicera

med myRIO:n är seriell kommunikation den enda möjligheten. Detta innebär att de ultraljudssensorer som finns förmonterade på roboten inte kan användas [33].

2.2.3 myRIO 1900

National Instruments har utvecklat en mikrokontroller, myRIO 1900, för att erbjuda studenter ett verktyg att konstruera avancerade system snabbt och enkelt [28]. Enheten har bland annat inbyggd accelerometer och WiFi-mottagare. Fördelen med nätverksåtkomsten i detta projekt är att den ger möjlighet för användaren att följa programmet i realtid och att den tillåter trådlös kommunikation med det överordnade systemet. Mikrokontrollern har utöver flertalet I/O portar även en USB port som kan användas för att koppla in komponenter.

National Instruments har utvecklat programspråket LabVIEW som används i bland annat myRIO:n. Det är ett grafiskt programmeringsspråk med möjligheter att interagera med andra språk såsom MATLAB, C, C++ och C# [42].

2.2.4 LiDAR

LiDAR (Light Detection and Ranging) är ett optiskt mätinstrument som används för att bestämma avstånd till objekt vilket behövs för att upptäcka hinder innan kollision sker. Den LiDAR som finns tillgänglig är modell URG-04LX-UG01, tillverkad av Hokuyo, se figur 2.3. LiDAR:n drivs och kommunicerar via en USB 2.0 anslutning [36].

2.2.5 Batteripack

För att driva roboten och dess komponenter används ett batteripack med spänningen 12 V och kapaciteten 9500 mAh.

2.2.6 Överordnat system

Det överordnade systemet kommer köras på en dator som är kopplad till samma trådlösa nätverk som AGV:n. Här sker de huvudsakliga beräkningarna bland annat eftersom myRIO kortets processkraft inte är tillräcklig för att hantera det med tillfredställande snabbhet [16]. Efter bearbetning kan systemet sedan skicka kommandon till AGV:n om vad som skall göras. Ett överordnat system krävs också om flera robotar skall kunna arbeta effektivt tillsammans.

Figur 2.3: LiDAR, Hokuyo modell URG-04LX-UG01 [36]

2.2.7 Kamera

För att läsa QR-koder används en Logitech C930 webbkamera, se figur 2.4. Kameran kan ta upp till 30 bilder per sekund i full HD 1080p (upp till 1920 x 1080 bildpunkter) och har dimensionerna 94 x 29 x 24 millimeter. All strömförsörjning av kameran sker via dess USB 2.0 anslutning.

Figur 2.4: Logitech C930 kamera [25]

2.2.8 Ram

För att underlätta montering av exempelvis kameran och anordningen för transport av material, samt minimera åverkan på Nexusroboten, finns en ram av aluminiumprofil fastsatt på AGV:ns ovansida längsmed dess ytterkanter.

2.2.9 Mecanumhjul

En fördel med roboten är dess mecanumhjul, vars design tillåter färd i olika riktningar, det vill säga inte bara framåt och bakåt som traditionella hjul, utan även i sidled. Fördelen kan vara till stor nytta i trånga utrymmen då exempelvis roboten kan rotera kring sin egna centrumpunkt. Skälet till att hjulen har nämnd förmåga är att de är uppbyggda av cylindrar vinklade mot hjulaxeln vilket kan ses i figur 2.2. Dock är precisionen vid förflyttning inte helt exakt vilket leder till att åtgärder måste tas för att minimera risken för att tappa rutten, mer om detta under kapitel 3.

2.2.10 Styrmotorer

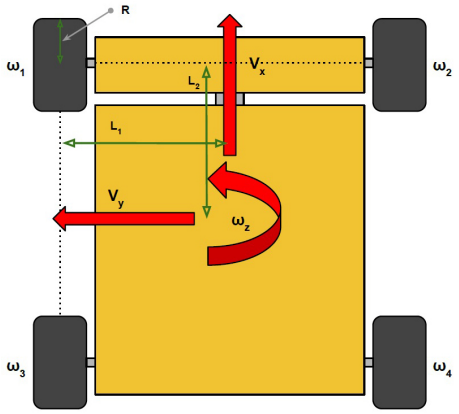
De fyra meccanumhjulen styrs individuellt av varsin motor, varje motor kräver en effekt på 17 W och vid en belastning som ungefär motsvarar AGV:ns vikt fås en maxhastighet runt 70 RPM. På varje motor sitter också en optisk enkoder som mäter varvtalshastighet vilket används som är-värde till PID-regulatorn i mikrokontrollern.

2.3 Robotens kinematik

Som tidigare nämnts tillåter robotens mecanumhjul färd åt olika riktningar tack vare dess utformning. Varje hjul har en egen motor som är oberoende av de andra vilket gör att ekvationer krävs för att bestämma hur varje hjul skall rotera för att uppnå en särskild rörelse.

Matriserna för att kunna beräkna hjulrotationen för en önskad rotation eller rörelse i X/Y-led av roboten har tagits fram av Doroftei, Grosu och Spinu på Technical University of Iasi, Rumänien och visas i ekvation 2.1 och figur 2.5 [11].

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = \frac{R}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ -\frac{1}{l_1+l_2} & \frac{1}{l_1+l_2} & -\frac{1}{l_1+l_2} & \frac{1}{l_1+l_2} \end{bmatrix} \tag{2.1}$$



Figur 2.5: Förtydligande av de ingående variablerna i ekvation 2.1 [16]

2.4 Motorkontroll

Arduino:n som medföljer roboten vid inköp är modifierad med ett expansionskort för att kunna kontrollera fyra motorer med hjälp av PWM (Pulse Width Modulation). PWM möjliggör att på ett snabbt sätt kontrollera spänningen över en motor. Parametrar som även behöver specificeras är hastighet, accelerationstid samt PID-reglering vilket förklaras mer utförligt under kapitel 7. För Nexusroboten finns det färdiga programbibliotek som används för att få roboten att kunna åka i standardrörelser.

Kapitel 3

Navigation

För att AGV:n skall ha möjlighet att hämta lagerinredning autonomt krävs en navigeringsmetod som hjälper AGV:n att kontinuerligt finna sin position i rummet. Det är svårt att skapa ett system som kan sköta lagerhanteringen utan någon form av återkoppling av positionen. Detta eftersom verkligheten inte alltid stämmer överens med teorin och det uppstår små fel som med tiden ackumuleras tills systemet inte längre fungerar som tänkt. Exempel kan vara varierande friktion mellan hjul och underlag.

I projektet värdesätts precisionen hos systemet eftersom gruppens uppfattningen är att med hög precision följer oftast hög funktionalitet och effektivitet. Därför söks en navigeringsmetod som tillåter systemet att arbeta med en hög precision och då är en kontinuerlig återkoppling en viktig komponent.

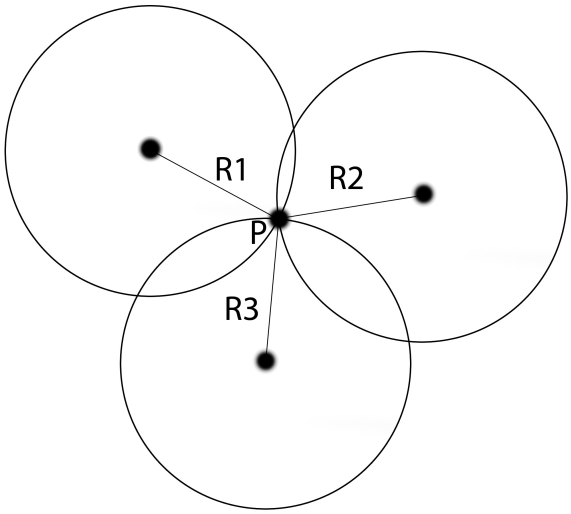
Genom att studera vad som används kommersiellt i området och liknande områden har ett antal olika metoder tagits fram för utvärdering [2, 22, 30, 29]. Krav som ställs på navigationen i projektet återfinns i den upprättade kravspecifikationen, se appendix A. Nedan följer en sammanställning av de olika navigationsmetoderna samt motivering för valet av navigation med hjälp av QR-koder.

3.1 Trilaterationsmetoder

För att navigera en AGV så kan man med hjälp av trilaterationsmetoder positionsbestämma roboten, vilket är en mer modern version av triangulering. Trilateration bygger på avståndsmätning från en position till tre andra punkter vars positioner är kända. Därifrån dras cirklar runt de kända punkterna med en radie motsvarande det uppmätta avståndet och där alla cirklar tangerar varandra är AGV:ns

position [35]. Figur 3.1 visar hur trilateration fungerar för att positionsbestämma en punkt P och varför det krävs tre kända referenspositioner för att få en exakt position.

Normalt används en elektronisk distansmätare som utnyttjar den kända ljus- eller radiovågshastigheten för att få ut en distans till en punkt tack vare dess höga precision och snabbhet. GPS är troligen det mest kända exemplet på användning av trilateration men utöver det kan också exempelvis WiFi eller laser användas.



Figur 3.1: En förklarande illustration på hur triliterationsmetoden går till

3.1.1 GPS och WiFi

GPS har överlägset bäst yttäckning men kan uteslutas i inomhusmiljöer eftersom det inte går att få fri siktlinje till satelliter och därmed inte heller en fungerande navigation.

Att använda WiFi-navigation är i många fall enkelt eftersom det ofta redan finns routrar tillgängligt som kan användas och det ger en relativt bra precision särskilt om fler än 3 routrar är tillgängliga för mätning. Dock är precisionen som bäst cirka 1m och påverkas kraftigt av hinder, särskilt människor, som är i vägen för kommunikationen vilket gör att precisionskraven inte kan uppnås, se appendix A [15].

3.1.2 Lasernavigation

Lasernavigation är en noggrann men dyr metod som används i stor utsträckning i kommersiella autonoma lagerhanteringssystem och bygger på att man sätter ut referenspunkter i form av reflektorer som en laser på AGV:n kan detektera samt mäta avståndet till [22].

Med lasernavigation kan en mycket hög flexibilitet uppnås eftersom metoden inte är begränsad till förutbestämda rutter på samma sätt som markeringsföljningsmetoder, se kapitel 3.2. AGV:n kan teoretiskt åka vart som helst med kontinuerlig positionsfeedback så länge minst tre olika reflektorer finns inom synfältet. För projektets ändamål är AGV:ns operationsmiljö dock sannolikt begränsad. Det eftersom de öppna ytorna vanligtvis är utformade som ett rutnät för att hålla en hög platseffektivitet vilket indirekt leder till förutbestämda rutter.

Tidigare kandidatarbete har använt lasernavigation och därigenom visat att metoden fungerar. Samma arbete har även visat att metoden kräver många reflektorer för att ej tappa position och att den kan störas av andra reflektiva ytor. Detta indikerar att metoden och den befintliga laserutrustningen eventuellt inte är optimal för projektet i och med den miljö AGV:n skall verka i [12].

Det kan också ifrågasättas hur väl laseravläsningen kommer fungera när flera AGV:er är verksamma, eftersom lasrarna kan störa ut varandras navigeringsförmåga med en försämring av systemets precision som följd.

3.2 Markeringsföljning

Det finns flera olika navigationssystem som bygger på att AGV:n följer markeringar av något slag. Nedan följer en redogörelse för de som anses ha störst relevans för projektet.

3.2.1 Linjeföljning

Linjeföljning är en relativt simpel metod som i den enklaste formen bygger på att AGV:n utrustas med lampor samt fotodioder som är sammankopplade med en transistor, och att den planerade rutten märks upp med exempelvis tejp. Tejpen belyses av lamporna och det reflekterade ljuset tas upp av fotodioden och om det är en hög intensitet på det reflekterade ljuset ökar fotodiodens resistans. Då ökar spänningen via transistorn som i sin tur låter motorn driva. Är det istället en

lägre intensitet på det reflekterade ljuset kommer fotodiodens motstånd minska vilket gör att spänningen sjunker och därmed minskar drivet i motorn. För att linjeföljningen ska vara så följsam som möjligt är det vanligt att varje hjul utrustas med sin egen krets. Genom att utnyttja systemet går det att kontrollera motorerna på AGV:n så att den hela tiden driver lite mer på det hjul som är på väg att tappa kontakten med tejpen, och därmed följs rutten på ett smidigt vis [24].

Fördelar med linjeföljning är att det är en relativt billig metod med en hög precision men det kan bli problematiskt att implementera metoden när det krävs ett rutnät av linjer. Att då få AGV:n att följa exakt den rutt som är planerad samt att få någon sorts positioneringsfeedback kan eventuellt visa sig svårt.

3.2.2 Magnetföljning

Navigation med hjälp av magneter är ett navigationssystem där en AGV utrustad med magnetsensorer utnyttjas för att följa magneter installerade i golvet. Genom att sensorerna kan känna av hur starkt magnetfältet är får AGV:n information om den följer sin rutt på ett tillfredställande sätt. Om den inte gör det kompenseras färdriktningen för att sedan återkopplas än en gång vid nästa magnet i golvet. Ett gyroskop finns installerat på AGV:n och det utnyttjas till att kontinuerligt lämna feedback på AGV:ns riktning. Vid oplanerade riktningsförändringar som kan inträffa mellan två magneter informerar gyroskopet AGV:n som då korrigerar riktningen.

Vanligtvis är det tillräckligt med en magnet i golvet var femte till tionde meter längs AGV:ns planerade rutt för att bibehålla en tillräcklig hög precision [27].

Fördelen med magnetföljning är att det är ett system med hög precision och billigare utrustning än exempelvis laserbaserade navigationssystem.

3.2.3 RFID

Radio Frequency Identification (RFID) tekniken kan användas genom att en RFID-tagg placeras på en plats och programmeras med information om den punkt den är placerad på, vilket sedan kan hämtas med hjälp av en avläsare. Ett RFID-system innehåller taggar, en avläsare samt mjukvara för tolkning av avläsning. Med hjälp av avläsarens antennen läses taggarna genom antingen en induktiv koppling eller med elektromagnetiska vågor. Ett sätt är att antennspolen inducerar ett magnetfält i taggens spole och taggen använder sedan energin som skapas för att återkoppla till avläsaren. Ett annat sätt är att avläsaren skapar energi som

elektromagnetiska vågor och energin reflekterar tillbaka från taggen till avläsaren för att återkoppla informationen i taggen. RFID-taggar används ofta tillsammans med andra system som till exempel GPS eller WiFi där taggarna utgör referenspunkter för att säkerställa en korrekt rutt. Om en avläsning av en RFID-tagg sker och referenspunkten inte stämmer överens med det övriga navigationssystemet så kalibreras navigationssystemet om efter RFID-taggens referenspunkt.

Den stora fördelen med RFID-avläsning är att det går fort att läsa av taggarna om kvalitétsprodukter används. Att läsa av taggarna i AGV:ns topphastighet är inget problem. Den stora nackdelen med systemet är att det inte finns möjlighet att avläsa taggens orientering och korrigera för eventuella vinkelfel [6, 18].

3.2.4 QR-koder

Streckkoder är ett sätt att lagra information så att den blir läsbar för en maskin. De har under lång tid används för att märka upp varor och var ursprungligen endimensionella. QR-koder är tvådimensionella streckkoder som kan innehålla betydlig mer information på mindre yta än de konventionella endimensionella streckkoderna. Koden är uppbygg av ljusa och mörka fält som benämns ”moduler” där antalet moduler kan variera mellan 21 x 21 upp till 177 x 177 beroende på mängden data. Avläsningen görs med hjälp av laser eller kamera och i jämförelse med endimensionella streckkoder kan avläsningen genomföras snabbare och mer tillförlitligt. Den asymmetriska utformningen av QR-koden i kombination med möjligheten att läsa av koden i 360 grader möjliggör uppskattning av storlek, position och lutning [31]. I figur 3.2 syns en bild på en QR-kod.

Figur 3.2: En QR-kod innehållandes texten ”Autonom lagerhantering”

Det finns flera fördelar med att implementera QR-koder i navigationssystem, bland annat är de är enkla och billiga att tillverka på egen hand då det finns mängder av

hemsidor som erbjuder dessa tjänster. Att avläsningen sker snabbt och i 360 grader möjliggör positionsbestämning av en AGV i rörelse och därmed kan korrigeringar utföras för eventuella fel. Nackdelen är att navigeringssystemet inte blir särskilt flexibelt, vilket är det stora problemet med alla de olika markeringsföljningsmetoderna.

3.3 Val av navigationsmetod

Generellt för navigationsmetoder som följer markeringar är att de i utgångspunkten är inflexibla, då alla markeringar måste sättas upp. Därigenom måste det område som AGV:n verkar i redan innan vara bestämd men i gengäld fås bra precision och snabb arbetshastighet. Trilatereringsmetoderna är ofta inte lika precisa men däremot smidigare och mer flexibla än markeringsföljningsmetoderna, även om uppsättning av till exempel reflektorer till laseralternativet skapar en viss inflexibilitet då de måste placeras strategiskt och överallt där AGV:n skall köra.

Tabell 3.1: Den första iterationen av en Pughmatris i projektet, med QR-kod som referenslösning

		Alternativ						
Kriterier:	**Viktning**	**QR-kod**	**Laser**	**Linje**	**RFID**	**Magnet**	**WiFi**	**GPS**
Navigeringsprecision	3	Referens	0	0	-1	0	Ej Tillr.	Ej Tillr.
Används kommersiellt	1	Referens	0	0	0	0	-	-
Ruttflexibilitet	2	Referens	1	-1	0	0	-	-
Systemflexibilitet	2	Referens	-1	-1	0	-1	-	-
Implementeringssvårighet	1	Referens	-1	1	-1	0	-	-
Snabbhet	1	Referens	1	1	1	1	-	-
Pris	2	Referens	-1	0	0	-1	-	-
Hårdvarustorlek/Vikt	1	Referens	-1	0	0	0	-	-
Störningskänslighet	1	Referens	-1	0	0	-1	-	-
Genomförbarhet	2	Referens	0	0	0	0	-	-
							-	-
Summa		Referens	-5	-2	-3	-5	-	-
Rangordning		1	4	2	3	4	-	-
Elimineras		Nej	Ja	Nej	Nej	Ja	-	-

I tabell 3.1 visas den Pughmatris som gjorts, vilket är ett sätt att poängsätta lösningar för att kvantitativt jämföra olika alternativ sinsemellan med avseende på olika kriterier för vad som ska åstadkommas med alternativen. Detta görs genom att en lösning tas som referens och de andra lösningarna värderas gentemot referenslösningen i ett antal olika kriterier. Varje lösning tilldelas antingen 1,0 eller -1 beroende på om lösningen är bättre respektive lika bra eller sämre än referenslösningen och därefter multipliceras värdena med viktningen för respektive kriterie och summeras till en totalpoäng för varje lösning. Efter att ha eliminerat de tydligt

sämsta alternativen byts referenslösningen ut mot en annan lösning och processen itereras med kvarvarande lösningar.

För att ge en rättvis värdering av varje lösning bör kriterierna täcka alla möjliga dimensioner, det vill säga allting som kan skilja lösningarna emellan och som är betydelsefullt hos lösningen. Utifrån uppskattningar av vad som är viktigt för lösningen, tillsammans med en upprättad kravspecifikation med krav och önskemål som ställs på det system som skall väljas, skapas kriterierna för Pughmatrisen, se appendix A. Viktningen och varje lösningsvärdering på respektive kriterie är baserat på uppskattningar.

Enligt Pughmatrisanalysen i figur 3.1 är magnet och laserföljning sämst värderade för ändamålet i jämförelse med QR-koder och elimineras därför i urvalsprocessen tillsammans med WiFi och GPS navigering som inte har tillräcklig precision som tidigare nämnts. Detta innebär att processen itereras men med ett annat alternativ som referens och endast QR-koder, linjeföljning samt RFID som alternativ vilket visas nedan i tabell 3.2.

Tabell 3.2: Sista iterationen i Pughmatrisen med linjeföljning som referenslösning

		Alternativ		
Kriterier:	**Viktning**	**Linjeföljning**	**QR**	**RFID**
Navigeringsprecision	3	Referens	0	-1
Används kommersiellt	1	Referens	0	0
Ruttflexibilitet	2	Referens	1	1
Systemflexibilitet	2	Referens	1	1
Implementeringssvårighet	1	Referens	-1	-1
Snabbhet	1	Referens	-1	0
Pris	2	Referens	0	0
Hårdvarustorlek/Vikt	1	Referens	0	0
Störningskänslighet	1	Referens	0	0
Genomförbarhet	2	Referens	0	-1
Summa		Referens	2	-2
Rangordning		2	1	3
Elimineras		Ja	Nej	Ja

Enligt analysen i Pughmatrisen är QR-kod den bästa navigationsmetoden med linjeföljning som näst bäst och därefter RFID. Detta stämmer relativt väl med den undersökning som gjorts på vad som används kommersiellt, där QR-kod, linje/magnetföljning och lasernavigering verkar vara de vanligast förkommande metoderna.

För projektet finns utrustning tillgänglig för QR-koder, laser samt linjeföljningsnavigation vilket gör att de alternativen är att föredra. Eftersom QR-metoden får klart bättre resultat i Pughmatrisen samt att linjeföljning är mer inflexibelt än QR-koder och laserns störningskänslighet är en oroande faktor, väljs metoden med QR-koder. Svagheten i ruttflexibilitet tros vara begränsad på grund av att lagermiljön troligen ändå ordnas i ett rutmönster vilket begränsar flexibiliteten oavsett.

3.4 Implementering

Nedan följer implementeringen av navigationssystemet. För att få systemet att fungera såsom uppvisas i det slutliga resultatet genomfördes en rad modifieringar på hårdvaran med anpassade mjukvarumässiga implementeringar. För mjukvarudelen användes LabVIEW vilket är ett grafiskt programmeringsspråk utvecklat av National Instruments som har stöd för parallellexekvering samt färdiga bibliotek som kan användas då samma företag har utvecklat myRIO:n som används i projektet.

3.4.1 Rotation

I ett av de tidigare kandidatarbetena användes QR-koder för att navigera [16]. En tidig förbättring som diskuterades var placeringen av kameran på AGV:n. En centrerad placering valdes. Tanken bakom att placera kameran centrerat med avseende på hjulen var att underlätta navigationen, exempelvis genom att inte behöva använda transformationsmatriser för att transformera positioner. En stor fördel är också den minskade ytan som krävs vid rotationen om man roterar kring centrum istället för en utomstående punkt. Centrerad rotation möjliggjorde även att redan existerande bibliotek kunde användas.

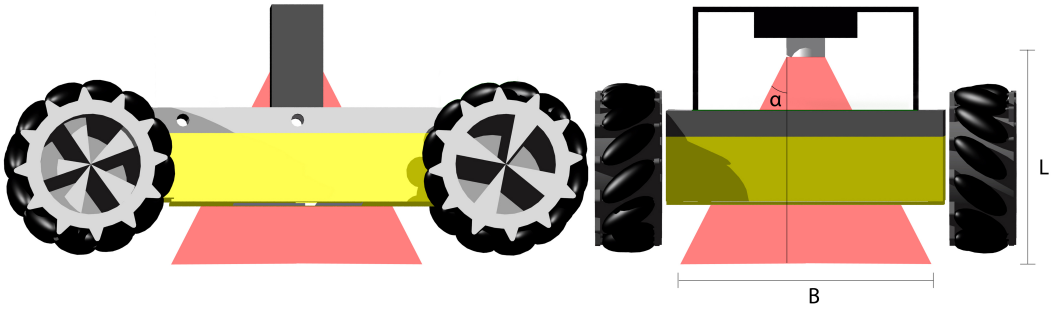
Eftersom AGV:n är relativt låg kunde inte kameran få fokus om den placerades undertill. Som lösning gjordes ett hål i chassit för att den skulle kunna placeras på ett längre avstånd från marken där den också fick ett större synfält att upptäcka koderna på, se figur 3.3. En enkel metallbygel gjordes för att fästa kameran ovanför hålet. Avståndet beräknades via chassits inre bredd, vilket är vad som avgör synfältets maximala bredd. Avståndet som valdes var 145 mm och beräknades således via ekvationer 3.1 och 3.2

$$L = \frac{B}{2} tan\,(\alpha) \tag{3.1}$$

Med värdena $B = 250$ mm och $\alpha = 30°$ fås L enligt ekvation 3.2:

$$L = 145 \text{ mm} \tag{3.2}$$

En justerbar infästning utvecklades för att kunna finjustera kamerans vinkel i förhållande till chassit. Detta för att på ett smidigt sätt kunna korrigera kameran och säkerställa kameramätningarnas riktighet.



Figur 3.3: En illustration som visar kamerans placering och dess synfält

3.4.2 Rörelsemönster

Nexusroboten kan via sina meccanumhjul röra sig i ett flertal riktningar. För att skapa en systematisk ordning i lagerlokalen där hyllor kan placeras i långa rader valdes ett korridorförfarande. Detta innebär att AGV:erna effektivast rör sig i korridorer och därför endast behöver kunna förflytta sig framåt, bakåt, höger, vänster och rotera på sin plats. Dessa rörelser kom att utgöra grunden för de olika navigationsfall som AGV:n antas kunna försätta sig i. Diagonala rörelser undviks på grund av begränsningar i kollisionsundvikningssystemet, se kapitel 5.

3.4.3 Läsning av koder

Läsningen av QR-koder sker, via kameran, i myRIO:n. I LabVIEW finns färdiga drivrutiner, så kallade block, för videoupptagning och urskiljning av QR-koder i bilder vilket användes. Ur blocket för läsning av QR-koder utvinns bland annat information som finns lagrad i koden samt placeringen av QR-kodens olika hörn.

Från början var tanken att bildupptagning skulle ske i rörelse så att AGV:n skulle kunna köra tills en ny QR-kod hittas och samtidigt reglera eventuell felplacering. Precis som nämns i tidigare kandidatarbete påträffades vissa begränsningar i hårdvaran. De bilder som togs under rörelse blev suddiga och ingen kod kunde upptäckas. I ovan nämnt kandidatarbete skrivs att den maximala hastigheten som AGV:n kunnat hålla för att klara av att läsa QR-koder i samtliga testförsök var 100 mm/sek [16]. Då detta ansågs vara en för låg hastighet utvärderades alternativa lösningar.

Två lösningsalternativ uppmärksammades, uppgradering av hårdvara eller en förändring av metodik. Då uppgraderingen av hårdvaran både skulle ta tid och öka kostnaderna föll valet på en ny metodik.

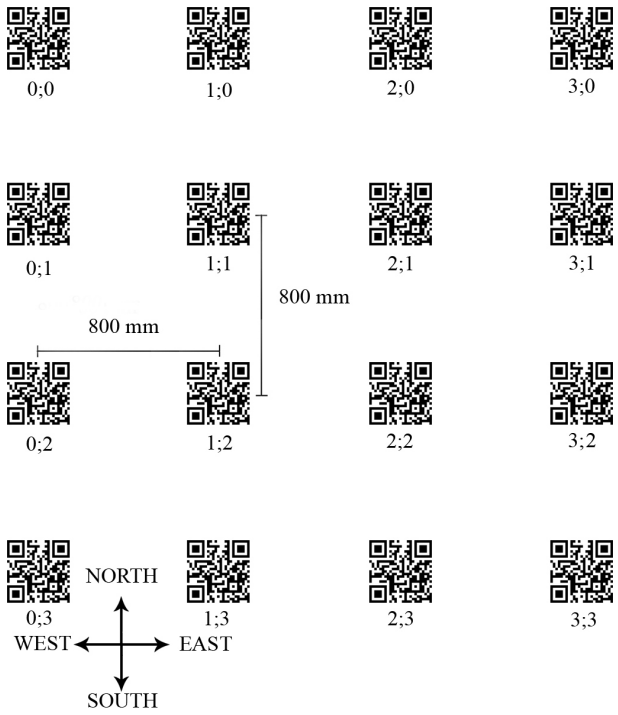
Istället för att försöka läsa koder i rörelse ändrades metodiken så att AGV:n står still över koden när bilden tas. En klar fördel med detta sätt är att bildens information, såsom placering och vinkel, fortfarande är aktuell. Därför behövs inga vidare beräkningar för att omvandla datan. Vad som är negativt är att AGV:n behöver stanna vid varje punkt vilket leder till längre tider för avläsning av QR-koderna. Denna tidsförlust kompenseras dock av den ökade hastigheten och systemets ökade precision.

3.4.4 Rutnät

Lagermiljön behövde även den anpassas för systemet. QR-koder placerades över lagerytan för att definiera det globala koordinatsystemet. Punkterna i koordinatsystemet motsvaras av en QR-kod som innehåller information om punktens x- och y-koordinat. För att förenkla programmeringen av ruttplaneringsalgoritmen utformades rutnätet som en matris med de första elementet (0,0) i matrisens övre vänstra hörn. Det globala koordinatsystemets axlar definierades sedan parallellt med matrisen, se figur 3.4, dessa bör inte förväxlas med de vanliga väderstrecken.

För att i varje punkt kunna avläsa AGV:ns orientering i förhållande till det globala koordinatsystemet behöver även QR-kodernas riktning tas hänsyn till. För att utnyttja QR-kodernas asymmetri till att beräkna vinkelfel placerades de åt samma håll.

En kortare distans mellan QR-koderna i rutnätet leder till en mer frekvent positionsåterkoppling och därmed ett mer precist navigationssystem. Bredden på det fyrkantiga bordet, som AGV:n skall lyfta, används för att bestämma längden mellan QR-koder eftersom det förutsätts att borden ska placeras intill varandra. En kortare distans mellan QR-koderna gör att återkoppling sker oftare vilket leder till att det kommer ta längre tid för AGV:n att förflytta sig från start till slutpunkt.



Figur 3.4: En schematisk bild på hur lagermiljön är uppbyggd

Detta rättfärdigas med att navigeringsprecision avses vara av högre prioritet än snabbheten som AGV:n utför uppgiften med. En tät placering av hyllor ökar också utnyttjandet av golvet och möjliggör yteffektiv lagring.

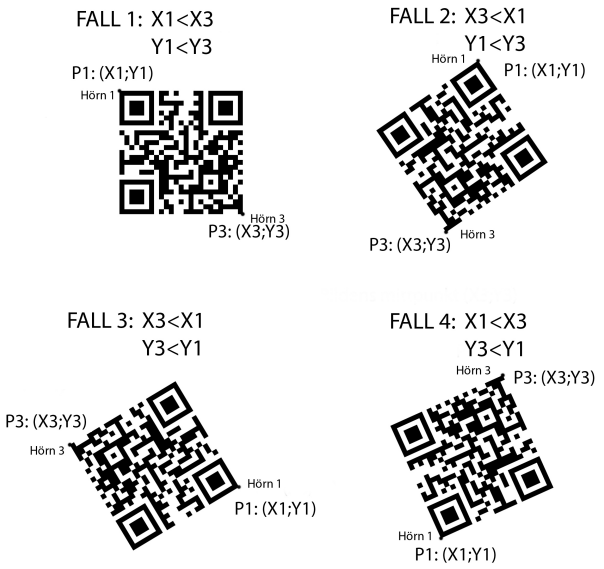
Om borden inte står helt rätt kan det skapa oönskade händelser vid transporten, framförallt vid rotation då ett felplacerat bord tar upp större plats än planerat. I det värsta fallet är båda borden vridna 45 grader så att de står hörn mot hörn vilket innebär att det är en diagonalbredd mellan de två bordens mittpunkter. Denna diagonal ges av ekvation 3.3 vilket gör att avståndet mellan QR-koder sätts till 80cm.

$$\sqrt{2 \cdot 55^2} \approx 77.78cm \tag{3.3}$$

Rutnätet av QR-koder som upprättades för projektet var i storleksordningen 4x4 eftersom platsen var begränsad samt att det ansågs tillräckligt för att påvisa systemets funktionalitet.

3.4.5 Offset och vinkel

Varje gång AGV:n får en återkoppling om sin position via en bild kan felaktig positionering och avvikelse från rutt kontrolleras och beräknas. För att göra detta användes den information om QR-kodernas hörnpositioner som erhölls ur LabVIEW-blocket.



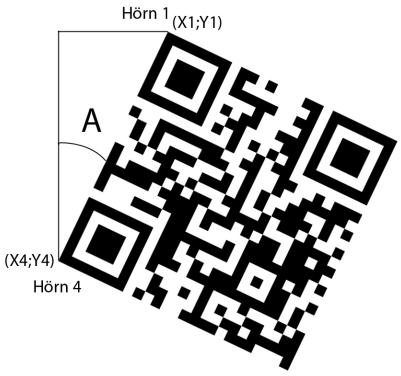
Figur 3.5: En bild på de fyra fallen för offset som finns

För att beräkna QR-kodens position i förhållande till bildens mittpunkt användes positionerna för hörnen som ges nummer 1 och 3, se figur 3.5. Från dessa positioner kan sedan QR-kodens mittpunkt och således antalet pixlar till kamerans mittpunkt beräknas. Antalet pixlar omvandlas sedan till ett avstånd i millimeter. För beräkningar, se appendix B.

Vid implementeringen upptäcktes att dessa beräkningar inte var tillräckliga utan hänsyn behövde tas till de fyra möjliga fall som kunde uppstå. Beroende på QR-kodens vinkel varierade hörnens x- och y-värde vilket ledde till en felaktig mittpunkt. Hänsyn togs till de olika fallen enligt figur 3.5.

För att sedan beräkna vilken vinkel QR-koden har i förhållande till kameran, användes som ovan positionerna av två hörn. Vid detta tillfälle definierades QR-kodernas koordinatsystem vilket sedan användes vid fastställandet av rutnätet som nämns ovan, se figur 3.6

Det globala systemets y-axel definierades som 0° och QR-koderna placerades med



Figur 3.6: Visar hur vinkelfelet mäts

sidan mellan hörn 1 och 4 parallellt med y-axeln, se figur 3.6. Vinkelfelet utgörs av den vinkel som uppstår mellan y-axeln (north) och den linje som uppstår mellan punkterna.

Den vinkel som AGV:n avviker från QR-koden kunde sedan i alla punkter beräknas via ekvation 3.4.

$$\alpha = arctan\left(\frac{\Delta y}{\Delta x}\right) \tag{3.4}$$

Vinkeln definierades som positiv vid högerrotation och negativ vid vänsterrotation. När AGV:n roterar och riktar sig mot south, se kapitel 5, behövde vinkeln minskas med en radian.

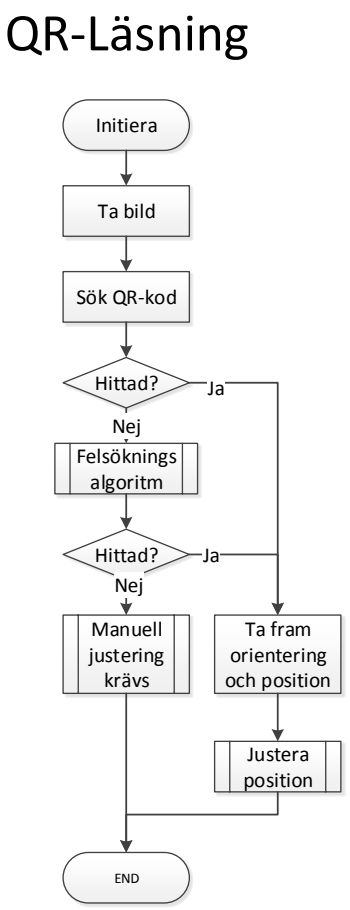
3.4.6 Processbeskrivning

Nedan följer en avskärmad överblick över implementeringen. Denna del är tänkt att skapa en överblick för att underlätta fortsatt läsning. För den fullständiga integreringen av systemet, se kapitel 7.

Vid varje uppstart av AGV:n kommer en bild att tas. Utifrån denna bild justerar sedan AGV:n in sig i förhållande till det globala koordinatsystemet. Här används beräkningarna som görs med avseende på offset och vinkel. Efter justeringen kommer alltså AGV:n att befinna sig på en utav koordinatsystemets punkter, vilket

den sedan kommunicerar till det överordnade systemet. Vid injusteringen så vänder sig AGV:n alltid mot det fördefinierade north, se figur 3.4. Värt att nämna är att den initierande justering inte kan göras om AGV:n inte står över en kod. Om AGV:n inte hittar en kod kommer ett felmeddelande levereras till operatören.

Eftersom hastigheten på AGV:n överstiger de 100 mm/s som var den maximala hastigheten för att klara att läsa av QR-koderna på ett lämpligt sätt, ansågs bildupptagning i rörelse som onödigt moment. Därför slutar bildupptagningen i samband med att kommando för rörelse mottas och startar i samband med AGV:ns stoppkommando. Detta frigör processorkraft till andra operationer.



Figur 3.7: Flödesschemat för QR-läsningen

Vid varje stopp börjar alltså bildupptagningen och därmed sökandet efter en QR-kod. Om en QR-kod finns i bilden tar systemet således tillvara på informationen lagrad i koden, som är offset och vinkelfel. Direkt kommer AGV:n att självmant justera offset och vinkelfel samt kommunicera AGV:ns position till det överordnade

systemet. I figur 3.7 visar ett flödesschema hur processen vid läsningen av QR-koderna går till.

Om bilden som behandlas inte innehåller en QR-kod genomförs inga beräkningar utan en ny bild tas direkt och analyseras. Detta förfarande upprepas tills en QR-kod finns i bild eller tills ett förutbestämt antal bilder tagits i behandling. Utifall ingen QR-kod hittas betyder det att AGV:n är ur kurs. För att hitta tillbaka till en känd position i koordinatsystemet genomför AGV:n en felsökningssekvens. Eftersom AGV:ns förflyttningar inte är helt exaka är sökningssekvensen en viktig komponent i det autonoma systemet för att minimera risken för att AGV:n tappar rutten. Felsökningssekvensen beskrivs närmre i kapitel 7.3.

Kapitel 4

Överordnat system

Efter valet av navigeringsmetod påbörjades arbetet med ett överordnat system. För att möjliggöra ett autonomt lagerhanteringssystem krävs ett överordnat system som hanterar och tar emot ordrar från personal, men även kontrollerar en uppsättning av AGV:er autonomt på ett säkert och effektivt sätt. I detta kapitel görs en genomgång av ruttplanering och motiveringen till varför A* väljs som ruttplaneringsalgoritm samt beskrivning av gränssnitt och hur det överordnade systemet angriper problemet.

4.1 Systemuppbyggnad

Systemet ska enligt kraven kunna ta emot ordrar, hantera dessa och autonomt styra flera AGV:er i en lagerlokal för att frakta varor till och från olika positioner. Systemet måste alltså klara av att ha flera kommunikationer samt beräkningar igång samtidigt. Detta kan ställa till stora problem med integreringen mellan ruttplanering, styrning samt kommunikation. För att lösa det måste alla delar dirigeras rätt för att uppnå ett effektivt och responsivt system.

Operatören ska inte behöva få vänta flera sekunder på svar från gränssnittet utan det måste svara snabbt för att inte verka förvirrande eller leda till osäkerhet. Inte heller får beräkningarna i ruttplaneringen, styrning eller kommunikationen med AGV:erna vara för långsamma, då systemet kommer verka väldigt ineffektivt. Med detta som utgångspunkt måste ett flexibelt och kraftfullt, men effektivt system byggas upp.

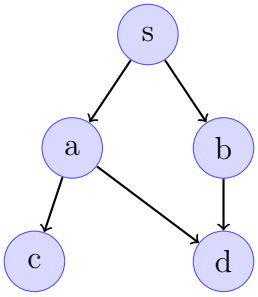
4.2 Gränssnitt

Operatören måste på ett enkelt och smidigt sätt kunna lägga till samt hantera ordrar via gränssnittet. Det måste också vara möjligt att via gränssnittet få status på systemet, samt få notifikationer när oväntade eller ödesdigra händelser inträffar.

4.3 Ruttplanering

För att AGV:erna på ett effektivt sätt ska hitta från start- till slutpunkt krävs en algoritm som räknar ut vilken väg de skall ta. För att åstadkomma detta krävs någon form av ruttplanering och för det finns flera olika metoder att tillämpa. Metoderna som valts att utvärderas är A*, Dijkstras algoritm, potentialfält, och Flood Fill. Nedan följer en utförlig utvärdering av varje metod samt en motivering till varför valet föll på A*.

Flera av dessa algoritmer använder sig av grafer för att representera datan i problemet. En graf $G(V, E)$ består av noder V, samt kanter, E som beskriver ”vägar” mellan noderna. En kant kan vara riktad, vilket betyder att den bara går åt ena hållet. Detta är ett väldigt enkelt men effektivt sätt att beskriva till exempel ett rutnät, där varje ruta då är en nod och kanterna är de olika sätten som du kan röra dig mellan rutorna. I figur 4.1 visas ett exempel på en enkel graf.



Figur 4.1: En riktad graf

4.3.1 A*

A*, utläses ”A-stjärna”, är en vanligt använd algoritm vid t.ex. datorspel [19]. A* hittar vägen från en startpunkt till flera slutpunkter, där vägen går förbi flera noder och A*-algoritmen väljer bästa väg vid varje nod. Det krävs alltså en fördefinierad gradering av varje väg mellan varje nod för att A* skall kunna finna den bästa

vägen. Graderingen benämns ofta kostnad, olika vägar blir alltså olika ”dyra” för roboten. Den totala kostnaden för en väg bestäms av ekvation 4.1, detta kallas algoritmens heuristik och bestäms utifrån problemets karaktär.

$$f(x) = g(x) + h(x) \tag{4.1}$$

I ekvation 4.1 representerar $g(x)$ kostnaden från startpunkten till noden x och $h(x)$ representerar den uppskattade kostnaden från noden till målet. I och med $h(x)$ erbjuder alltså A* en möjlighet för roboten att överväga framtida kostnader för olika vägar vilket möjliggör en kostnadsavvägning för hela vägen mellan start- och målpunkt.

Det finns två vanliga sätt att beräkna kostnaden, Manhattanmetoden och den Euklidiska metoden [26]. Manhattanmetoden har som utgångspunkt i att vägarna ses som vägarna på Manhattan, New York, alltså ett rutnät. Den euklidiska metoden handlar om att den kortaste vägen hittas, den så kallade ”fågelvägen” [17].

4.3.2 Visibility graph/Dijkstras algoritm

Visibility graph är inte en algoritm på samma sätt som A*, utan den sätter upp en visuell graph som sedan andra sökalgoritmer som Dijkstras algoritm kan använda för att ruttplanera. Ett visst antal noder definieras samtidigt som Visability graph försöker dra räta linjer mellan noderna där AGV:n kan köra. Ifall det är ett hinder för den räta linjen, dras den inte. Därefter beräknar Dijkstras algoritm ut bästa vägen mellan start- och slutpunkt.

Noderna utgår ofta från fasta hinders kanter och några andra få punkter. Vad som kan bli ett problem är att vissa noder kan finnas utan att ha andra noder synliga. Om rutten går via någon av dessa noder kommer systemet stanna. Detta problem kallas för ”The art gallery problem” och grunden i det kända problemet ligger i att bestämma hur många noder som krävs i ett rum för att alla delar av rummet skall vara synligt för någon av noderna [39, 40, 26, 43, 9].

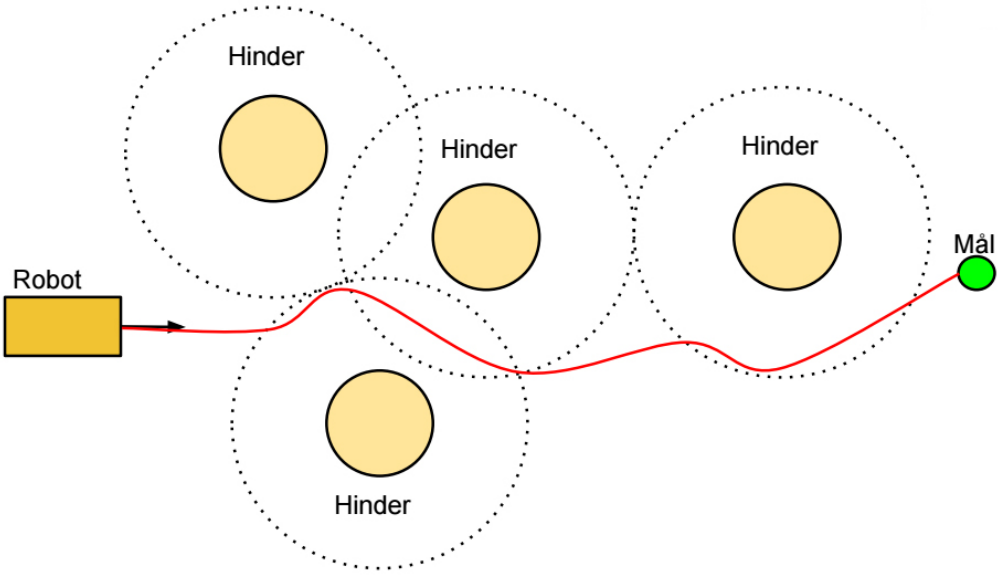
Skillnaden på Dijkstra och A* är att A* lägger till den uppskattade kostnaden för resterande delen av vägen. Detta gör att A* möjliggör en effektivare ruttplanering eftersom A* koncentrerar sig på de vägarna som uppskattas som bäst, medan Dijkstraalgoritmen utvärderar varje väg vid varje nod [26, 43].

4.3.3 Potentialfält

Potentialfält innebär att man ger vissa punkter i miljön attraherande kraft och vissa punkter repellerande kraft. Förslagsvis gör man målpunkten attraherande kraft och fysiska hinder i miljö repellerande kraft. Detta medför att roboten alltid dras till målet och undviker hinder.

Problemet med den här metoden är att det inte rör sig om någon ruttplanering i ordets rätta bemärkelse, utan roboten kommer röra sig närmare och närmare målet samtidigt som den undviker hinder succesivt när de uppkommer. Det kommer därför vara svårt att förutsäga vilken rutt roboten kommer ta.

Ett annat problem som kan uppkomma är att de repellerande krafterna tar ut de attraherande, vilket gör att den resulterande kraften som verkar på roboten blir noll. Dessutom finns risken att roboten inte kommer ta den kortaste vägen eftersom roboten kommer påverkas av ett repellerande hinder oavsett ifall den inte hade kollisionskurs med hindret, se figur 4.2 [5, 34].

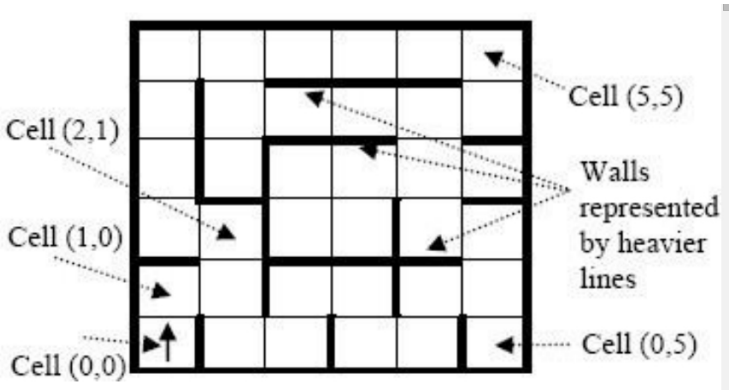


Figur 4.2: Ett exempel på att AGV:n inte nödvändigtvis behöver ta närmasta vägen [12]

4.3.4 Flood fill

Algoritmen Flood fill bygger på konceptet att vatten alltid fyller från en högre vattennivå. Området som skall ruttplaneras delas in i celler som ges ett värde beroende på hur långt ifrån cellen är centrum, alltså där vattnet ”fylls på”. Ju längre ifrån centrum desto lägre värde får cellen.

Flood fill är ursprungligen en labyrintlösare. Väggarna i labyrinten är fördefinierade och algoritmen upptäcker dem efterhand som labyrinten löses [32]. Se figur 4.3 för en bild av en labyrint som är löst med flood fill.



Figur 4.3: Labyrint löst med flood fill [32]

4.4 Val av ruttplaneringsmetod

I och med tidigare erfarenheter så blev valet av ruttplaneringsalgoritm lätt. A* är väldigt flexibel, på grund sin heuristik kan man skräddarsy den för sitt problem. Om heuristiken väljs rätt så garanterar A* att den optimala vägen returneras, vilket går snabbt. Det realiserade rutnätet i projektet är relativt litet och A* kan då verka väldigt onödig och överflödig, men systemets bakomliggande tanke är att det ska fungera på betydligt större lagerutrymmen än så. Därför anses valet av A* rättfärdigat.

4.5 Implementering av överordnade systemet

Med klara krav på vad det överordnade systemet måste klara av så började uppgiften med att välja på vilket sätt systemet skulle byggas upp. Detta inkluderar

val av programmeringsspråk, men även konceptuella och arkitekturella val.

Det finns många programmeringsspråk som skulle passa in i projektet, men med tanke på tidigare erfarenheter samt kompetens valdes *Clojure*, som är ett dynamiskt programmeringsspråk med fokus på funktionell programmering. Clojure är ett språk som är en dialekt ur familjen *Lisp*. Språket valdes också för den interaktiva utvecklingsprocessen, stödet för parallell programmering samt interopabiliteten med det stora programmeringsspråket Java [7].

En stor fördel med Clojure är att det existerar en kompilator, *ClojureScript* som har JavaScript som målplattform [8]. Detta betyder att man kan exekvera Clojurekod i en webbläsare vilket gör det utmärkt att bygga gränssnittet som en webbapplikation i HTML5 och ClojureScript (JavaScript).

Implementeringen av systemet började med att upprätta och säkerställa kommunikationen till flera samtidiga AGV:er. Ett enkelt textbaserat protokoll över TCP togs fram, detta beskrivs i kapitel 7.2.1. Efter manuella tester samt några iterationer så existerade den funktionaliteten som behövdes för att upphålla realtidskommunikation med AGV:er. Under utvecklingsfasen så användes virtuella AGV:er där bara kommunikationen fanns, detta för att underlätta utveckling och få en effektiv återkoppling till utvecklingen. De ”virtuella AGV:erna” bestod egentligen bara av enkla skript i en kommandotolk som använder sig av ett unix-program som kallas netcat, vilket används för att kommunicera över TCP-anslutningar. En asynkron tillståndsmodell sattes också upp i servern, denna har koll på all status för de anslutna AGV:erna och exponerar ett API för att kunna göra atomiska skrivningar och på så sätt slippa problem som kan uppstå vid asynkrona skrivningar.

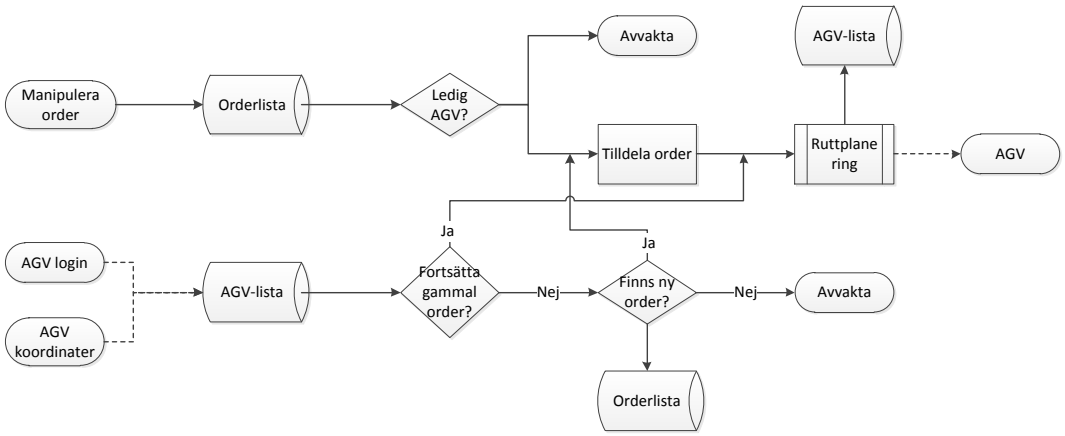
När implementeringen av kommunikationen samt tillståndslagringen var klar påbörjades en enkel skiss på gränssnittet. De tidigare kraven användes som en specifikation för vad som behövdes finnas med, en slutgiltig beskrivning av gränssnittet återfinns i kapitel 4.5.1. Som tidigare nämnt utvecklades gränssnittet i Clojure och kompilerades till JavaScript via kompilatorn ClojureScript, detta medför en stor fördel då det är möjligt att använda samma sorters datastrukturer både i servern och i gränssnittet. Datan som som skickas i realtid mellan servern och gränssnittet faller under formatet edn (Extensible Data Notation), vilket gör det väldigt smidigt att utbyta information[13]. För att skicka data i realtid mellan servern och gränssnittet används en modern webbteknik som kallas WebSockets. WebSockets fungerar i princip på samma sätt som en vanlig TCP-socket, en tillförlitlig full-duplex datakanal [41].

Den sista funktionaliteten att implementeras var den största och viktigaste, själva planeringen. Som nämnt i teoridelen i detta kapitel så valdes A*-algoritmen för att hitta kortaste vägen i rutnätet. Vid varje steg i processen som AGV:n utför

så beräknas den kortaste vägen för varje AGV som är ansluten. Planeraren tar övriga AGV:er, blockerade rutor samt hyllor i beräkning i varje steg för att se till att en väg inte korsar en hylla eller annan AGV, allt för att undvika kollision. En position kan bli blockerad genom att AGV:n meddelar att den har stött på ett hinder, se kapitel 5.4.2, då markeras positionen i kartsystemet som en blockerad ruta. För att få bort ett hinder krävs en omstart av systemet.

I figur 4.4 visas ett flödesschema på hur det överordnade systemet fungerar. Det finns två vägar in i schemat, antingen manipuleras en order via gränssnittet, eller så ansluts alternativt skickar en existerande AGV data till systemet. Varje gång en AGV har slutfört ett steg så meddelar den sin position och status till systemet, som tar den informationen i beräkning till vad som ska skickas tillbaka. Antingen fortsätter den med en order som den håller på att leverera, eller så kollar systemet om det finns en ny order att behandla. Detta resulterar i att antingen så skickas kommandon till AGV:n om vad den ska göra i nästa steg, eller så avvaktar systemet tills ny order inkommer.

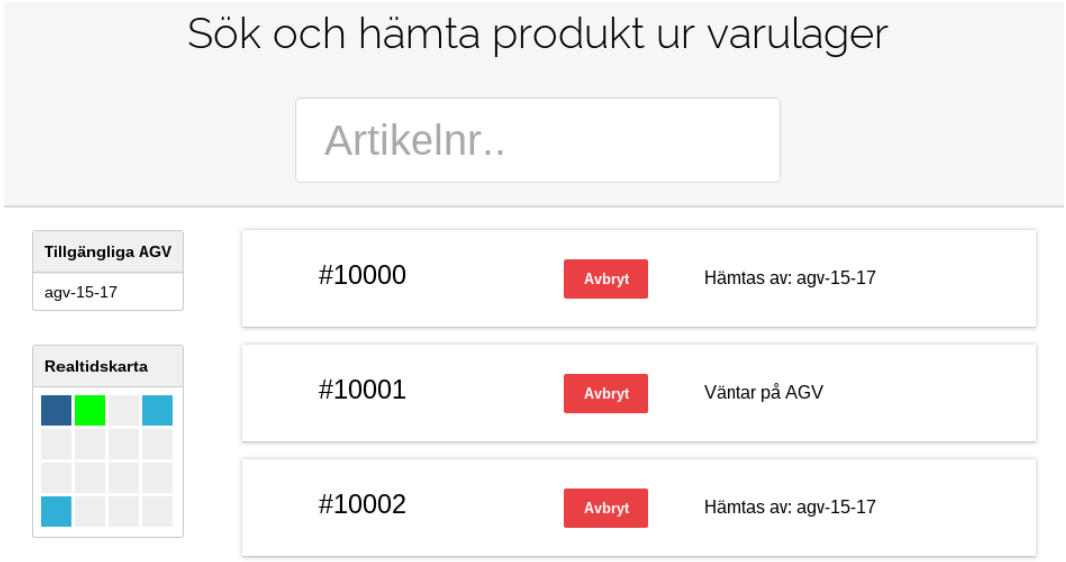
Systemet är dynamiskt och det är fullt möjligt att ansluta flera AGV:er under tiden systemet körs. Det har däremot valts att inte hantera när en AGV försvinner då det kan betyda att batteriet tagit slut och att den då står mitt i lagerutrymmet och blockerar. Valet blev då istället att göra operatören uppmärksam på problemet via gränssnittet och därefter är manuell hantering nödvändig.



Figur 4.4: Flödesschema på det överordnade systemet

4.5.1 Beskrivning av gränssnittets utseende

I figur 4.5 ses en bild av hur det överordnade systemets webbaserade gränssnitt ser ut. I nedersta vänstra hörnet syns en karta över lagersystemet som uppdateras



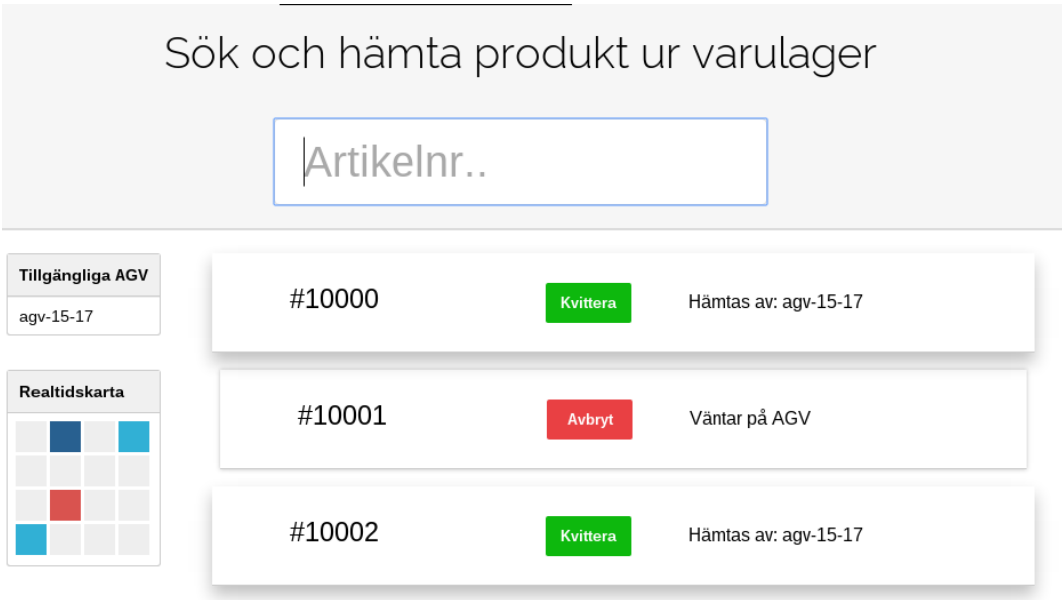
Figur 4.5: Skärmbild på ordrar i gränssnittet

i realtid, där varje ruta motsvarar positionen av en QR-kod. Olika färger i kartan motsvarar de ingående delarna i systemet. En grön ruta är plockstationens position, det vill säga AGV:ns målposition, mörkblå motsvarar en AGV, ljusblå för bord och röd indikerar på ett hinder.

När en AGV kopplas upp mot systemet läggs den in i listan till vänster, under rubriken tillgängliga AGV. Användaren skriver in sökt artikelnummer i sökfältet och då läggs en order in i systemet. Det överordnade systemet vet var varje artikel befinner sig och ruttplanering genomförs. Därefter tilldelas vald AGV information om färdriktning och därmed hämtas artikeln, för mer ingående information om hur navigeringen sker se kapitel 3.

Om flera order läggs där olika artiklar återfinns på samma lagerplats så kommer systemet tilldela samma AGV ordern att hämta de båda artiklarna, vilket kan ses i figur 4.5. Där har tre ordrar lagts in på olika artiklar och eftersom artiklarna med artikelnummer 10000 och 10002 återfinns på samma lagerplats så tilldelas samma AGV bägge ordrarna. Då ingen AGV finns tillgänglig kommer helt enkelt systemet invänta en tillgänglig AGV och därefter tilldela den ordern. Vilken AGV som hämtar vilken artikel syns tydligt i användargränssnittet och om utifall en order avbryts via knappen ”Avbryt” så kommer ett av två scenarion inträffa. Om AGV:n inte hunnit hämta det aktuella bordet så kommer den helt enkelt stanna vid nästa QR-kod och invänta en ny order. Men om AGV:n redan har hämtat bordet och är på väg tillbaka till plockstationen så kommer kommandot avbryta

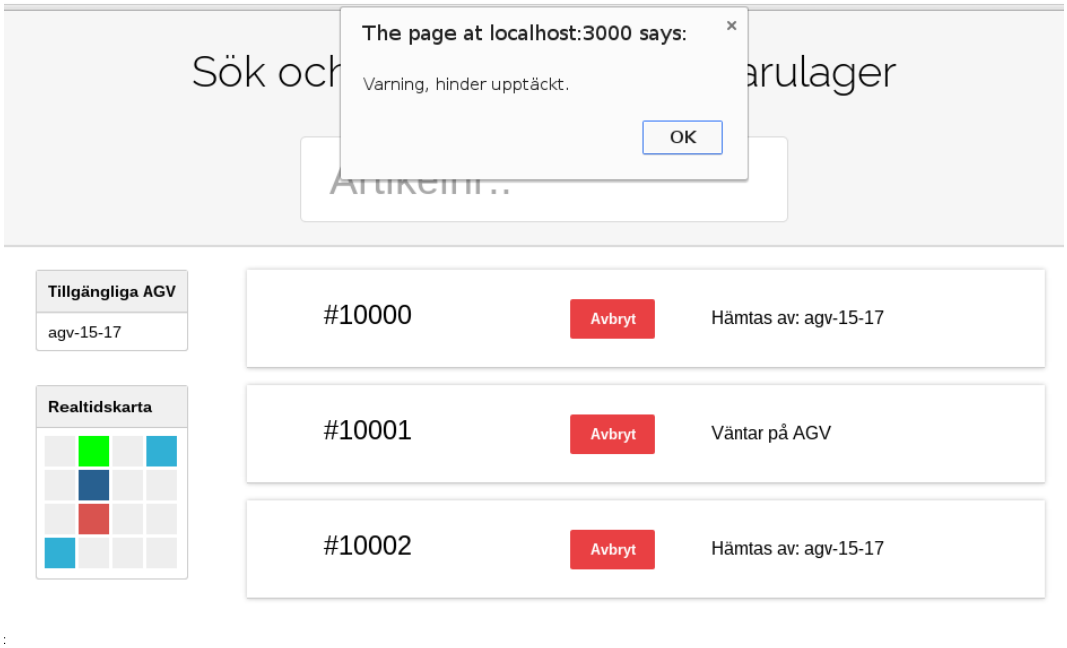
innebära att AGV:n kör tillbaka med bordet och därefter invänta order.



Figur 4.6: Skärmbild på kvitterbara ordrar i gränssnittet

När en AGV väl har hämtat en artikel och nått plockstationen kommer alternativet ”Kvittera” upp i systemet, se figur 4.6. När ”Kvittera” - kommandot ges är det en signal till systemet att artikeln är avplockad från bordet och AGV:n lämnar därefter tillbaka bordet på sin ursprungliga plats för att sedan invänta order.

Utifall ett hinder påträffats av AGV:ns kollisionsundvikningssystem så skickas information om vilken den aktuella QR-koden är och det överordnade systemet markerar följaktligen motsvarande QR-kod med rött i realtidskartan. Detta indikerar på att den aktuella QR-koden är indisponibel och en ny rutt beräknas. En varningsruta syns då i gränssnittet för att göra operatören uppmärksam på det påträffade hindret, se figur 4.7.



Figur 4.7: Skärmbild på när AGV:n har skickat blockeringssignal

Kapitel 5

Kollisionsundvikning

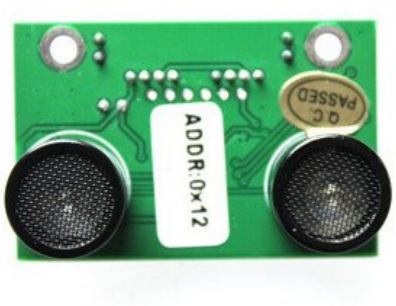
För att kunna använda AGV:erna i en lagermiljö krävs det att det finns system för kollisionsundvikning. Detta för att inte riskera personalens säkerhet men även för att effektivisera lagerhanteringssystemet. Det överordnade systemet håller ordning på AGV:erna i förhållande till varandra och på så vis undviker AGV:erna att kollidera med varandra, se kapitel 4.5.

För att undvika kollisioner med objekt som inte finns loggade i det överordnade systemet, exempelvis människor som rör sig inne i lokalen, måste ett ytterligare system implementeras. För att utreda vilket system som är bäst lämpat för uppgiften görs en utvärdering kring olika metoder som kan användas för kollisionsundvikning. Utvärderingen har begränsats till tillgängliga system från tidigare arbeten. Detta görs för att de tillgängliga systemen anses väl lämpade för uppgiften och därmed finns ingen anledning att köpa in något ytterligare. De tillgängliga systemen ultraljudssensorer och en lasersensor, en så kallad LiDAR.

För att möjliggöra ett effektivt kollisionsundvikningssystem krävs en metod för att skanna av omgivningen för att upptäcka eventuella hinder. Skanningen bör ske med hög frekvens så att rörliga hinder som uppkommer nära AGV:n kan upptäckas så fort som möjligt och ge AGV:n möjlighet att stanna i tid. I kravspecifikationen återfinns de krav som ställs på kollisionsundvikningssystemet i projektet, se appendix A. Nedan följer en utvärdering om de tillgängliga systemen samt en motivering till varför LiDAR väljs.

5.1 Ultraljud

Sonar (Sound Navigation and Ranging) är en teknik som med hjälp av ljudets fortplantning gör det möjligt att avståndsbestämma objekt i omgivningen. Ultraljudssensorer utnyttjar denna teknik och skickar ut pulser av ultraljud som reflekteras mot ett objekt vilka sedan fångas upp av sensorn. Genom att beräkna tiden det tar för ljudet att nå tillbaka kan ett objekt avstånd bestämmas [38]. Ultraljudssensorerna som är aktuella i projektet är av modellen *Arduino Ultrasonic Distance Sensor* (RS485 interface),se figur 5.1. Fyra stycken följer med roboten och monteras en per sida. Ultraljudssensorerna kan mäta avstånd på upp till fem meter med en upplösning på en centimeter[4].



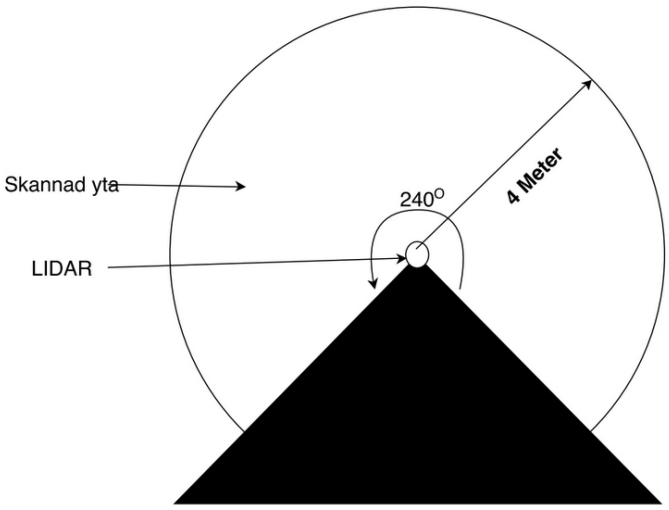
Figur 5.1: Ultraljudsensorn, *Arduino Ultrasonic Distance Sensor* [4]

Fördelen med att tillämpa ultraljudssensorerna är att AGV:n då har kollisionsundvikningsystem som fungerar i alla fyra riktningar samt att AGV:n redan har hårdvaran monterad. Eftersom det dock inte finns information kring hur brett ultraljudfältet är för den specifika sensorn, finns det risk att objekt som kommer från sidan inte upptäcks. Ultraljudssensorerna drivs och kommunicerar via den seriella anslutningen RS485 men eftersom myRIO:n redan kommunicerar via Arduino:ns enda seriella port finns det inte möjlighet att samtidigt använda ultraljudssensorerna. Det är dock möjligt att via mjukvara implementera en seriell port, men detta låser upp processorn och förhindrar annan kod att exekveras korrekt vilket i princip gör det omöjligt att tillämpa ultraljudssensorerna i projektet.

5.2 LiDAR

LiDAR (Light Detection and Ranging) är ett optiskt mätinstrument som kan tillämpas för att avståndsbestämma objekt, se figur 2.3. Den fungerar likt en sonar men istället för ljudvågor skickas ljusvågor ut och genom att mäta tiden det tar

för dem att reflekteras tillbaka kan avståndet bestämmas till det objekt som ljuset träffat [23]. Den klarar av att skanna 240° med en uppdateringsfrekvens på 10 Hz, se figur 5.2. LiDAR:n kan upptäcka föremål på upp till fyra meter med en felmarginal på tre procent av uppmätt avstånd, dock minst 30 millimeter. Under skanning görs 683 olika avståndsmätningar vilket innebär att mätpunkterna är separerade med 0.36° [36].



Figur 5.2: Scanningsspannet för LiDAR:n [12]

Informationen som skickas från LiDAR:n är en sträng med 683 punkter innehållandes avståndsmätningarna. Värdena representerar det uppmättat avståndet för varje vinkelintervall från LiDAR:n till ett objekt, om inget objekt påträffas returneras en nolla.

LiDAR:n drivs via en USB 2.0 port vilket är fördelaktigt då det är enkelt, med hjälp av en USB-hub för att få portar till både LiDAR och kameran, att sammankoppla den med myRIO:n. En nackdel är det faktum att LiDAR:n endast tillåter avskanning i 240° vilket innebär att om AGV:n backar finns inte möjligheten för LiDAR:n att detektera objekt som kan vara i vägen.

5.3 Val av kollisionundvikningssystem

En avvägning görs mellan de två tillgängliga teknikerna och utifrån det väljs LiDAR:n som kollisionsundvikningssystem. Både ultraljudsensorerna och LiDAR:n uppfyller de krav som ställs på kollisionundvikningssystemet vilket är det som är

mest essentiellt, se appendix A. Istället för att grunda beslutet på en jämförelse av hur väl kriterierna uppfylls av de olika lösningarna, likt det som gjordes för navigationssystemet, togs ett beslut baserat på det faktum att bristen på seriella anslutningar på Arduino:n gör att det blir väldigt komplext att implementera ultraljudssensorerna. Nackdelen med att välja den befintliga LiDAR:n är att den endast kan läsa av 240° av omgivningen vilket dock går komma runt genom att programmera AGV:ns rörelsemönster så att den aldrig tillåts röra sig baklänges utan istället rör sig i framåt samt i sidled och roterar vid behov vilket alltså anses vara lättare än att använda ultraljudssensorerna.

5.4 Implementering

För att implementera kollisionsundvikningssystemet utnyttjades LabVIEW vilket möjliggjorde en smidig integration med styrningen av AGV:n, se kapitel 3.

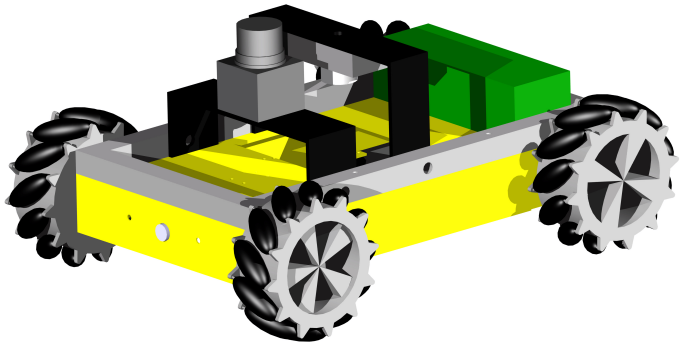
5.4.1 Synfält

LiDAR:n kopplades ihop med myRIO:n via USB och placerades på den främre delen av AGV:n för att minimera antalet delar på AGV:n som skymmde sikten. Dock skymde benen på anordningen för transport av material samt benen på bordet fortfarande LiDAR:ns synfältet. För att komma runt det problemet programmerades myRIO:n till att ignorera mätvärden från LiDAR:n som var på ett avstånd kortare än 20 cm. Detta löste även de problem som tidigare arbeten har haft med inkorrekta mätvärden på korta avstånd.

För att undvika kollision med hinder som kan uppstå programmerades LiDAR:n i LabVIEW att skanna av omgivningen i färdriktning, se figur 5.4. När ett objekt påträffats inom ett förutbestämt område skickas informationen om vinkel samt avstånd till myRIO:n som då stoppar AGV:n.

Eftersom LiDAR:n endast kan skanna av omgivningen i 240° programmerades AGV:n till att, istället för att backa, rotera 180° och köra framåt. Dock så backar AGV:n i de fall som ett fast hinder påträffats, mer om detta under kapitel 5.4.2. För att förbättra LiDAR:ns synfält gjordes en upphöjning för LiDAR:n, se figur 5.3.

Sikten för LiDAR:n var ständigt delvis skymd av anordningen för transport av material vilket ledde till en dödzon bakom benen. Detta kunde blivit en begränsande faktor i kollisionsundvikningssystemet men tester indikerade på att det inte

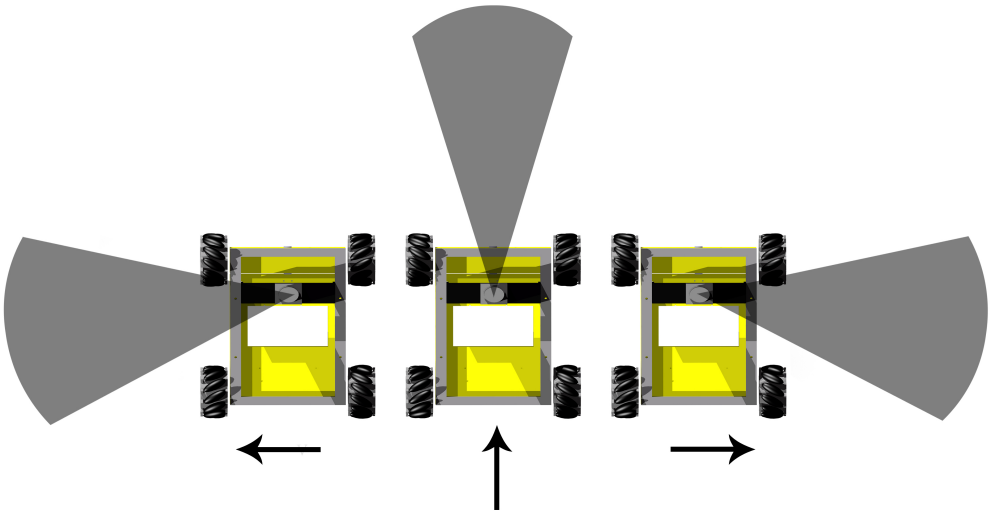


Figur 5.3: Bild som visar hur LiDAR:n är fäst på AGV:n

gjorde någon skillnad i praktiken. Då diagonala rörelser inte tillämpades, minskade problematiken med dessa dödzoner.

För att bestämma bredden på skanningsområdet från LiDAR:n betraktades två fall: när AGV:n var olastad och när den var lastad. När AGV:n var olastad kommer ett mindre sökfält än vid lastat läge att behövas. Ett sökfält motsvarande ekipagets bredd utgör minimimåttet. Ett bredare sökfält kan dock komma att hindra AGV:n från att manövrera in under bordet då dess ben kan tolkas som hinder. Lösningen blev att definiera start och slutpunkt till LiDAR:n så att sökfältet blev smalt nog för att tillåta AGV:n att färdas in under bordet utan att bordsbenen bryter lasern. Beräkningarna baserades på det faktum att hela AGV:ns bredd måste inkluderas i skanningsfältet för att kollisionsundvikningssystemet ska fungera på ett tillfredställande sätt, se appendix C och figur 5.5. När AGV:n har lastat bordet programmeras LiDAR:n att mäta ett något bredare fält för att då även få med den extra bredden som bordet medför.

Beräkningar gjordes för att bestämma respektive vinkel i de två fallen. I tabell 5.1 återfinns resultatet och för mer ingående beskrivning av hur beräkningarna görs se appendix C.



Figur 5.4: Illustration av hur LiDAR:n skannar beroende på färdriktning

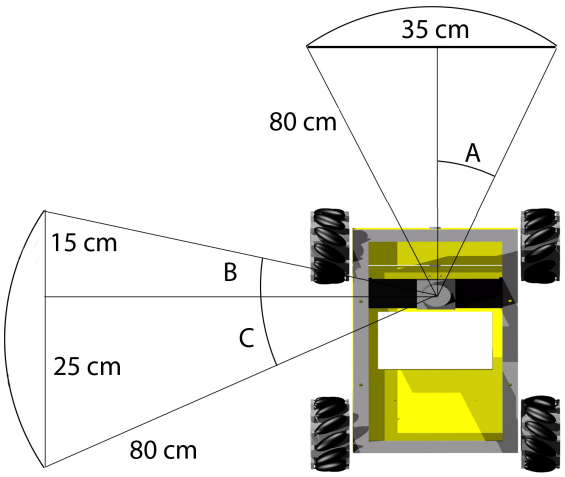
Tabell 5.1: Värdena för LiDAR:ns skanningsspann lastad och olastad, se 5.5 för vinkelförklaring

Vinkel	Lastad	Olastad
A	13°	20°
B	11°	16°
C	18°	24°

5.4.2 Agerande vid påträffande av hinder

Hinder som påträffas kategoriseras som rörliga eller fasta. För att avgöra vilken typ av hinder som påträffats så implementerades en sekvens som stannar AGV:n vid hindret. LiDAR:n fortsätter skanna kontinuerligt medan AGV:n avvaktar, försvinner hindret inom fem sekunder fortsätter AGV:n på sin bestämda rutt och inga ytterligare åtgärder tas. Om hindret fortfarande är kvar efter fem sekunder klassas det som ett fast hinder och AGV:n backar tillbaka till föregående QR-kod. MyRIO:n informerar samtidigt det överordnade systemet om det fasta hindrets position som då markerar den aktuella QR-koden som indisponibel Därefter beräknas en ny rutt, se mer angående hinder och ruttplanering under kapitel 4. I de fall som AGV:n påträffar ett fast hinder och backar tillbaka sker det i blindo på grund utav begränsningar i LiDAR:ns synfält, se kapitel 5.2. Detta är ett område med utvecklingspotential vilket diskuteras mer i kapitel 9.

Ett problem som påträffades under arbetet med LiDAR:n och implementeringen



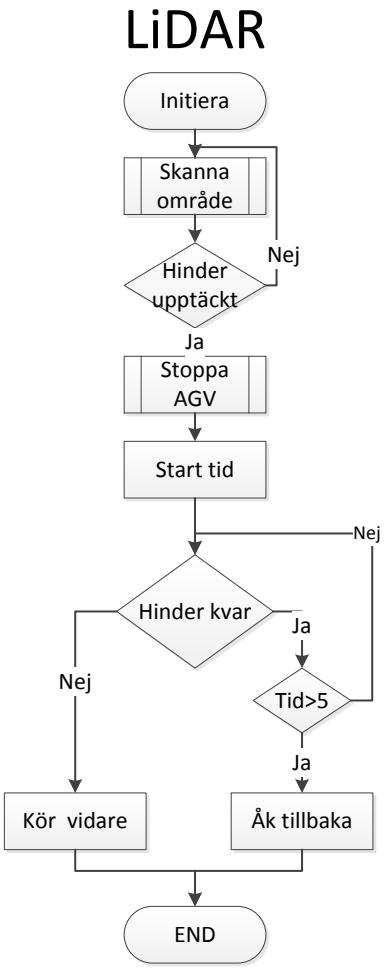
Figur 5.5: Dimensionerna på LiDAR:ns synfält och en definition av de ingående vinklarna

av systemet var att begränsningar i mjukvaran påverkade tiden det tog för LiDAR:n att upptäcka hindret tills dess att AGV:n stannat. Tester indikerade på att det tog ungefär 1,5 sekunder från det att hindret påträffats till att AGV:n stannade, se appendix F. Detta ansågs vara för lång tid. För att komma tillrätta med problemet kontaktades National Instruments för att få assistans med programmeringen av LiDAR:n i LabVIEW [1]. Vid genomgång tillsammans med en anställd vid National Instruments felsöktes funktionen och begränsningarna kunde kopplas till drivrutinerna för LiDAR:n till LabVIEW utgivna av Hokuyo. Detta är ett tydligt område där det finns potential för vidareutveckling, se kapitel 9.

5.4.3 Processbeskrivning

Ett flödesschema upprättades för att ge en överblick över hur det slutgiltiga systemet med LiDAR:n fungerar, se figur 5.6. LiDAR:n skannar av sin omgivning tio gånger i sekunden och om ett hinder påträffas inom bestämt avstånd så skickas en signal via myRIO:n till Arduino:n att stoppa motorerna och följaktligen stannar AGV:n. Sekvensen för hinderupptäckt initieras och beroende på vilken typ av hinder det är så åker antingen AGV:n vidare när hindret försvunnit eller så backar AGV:n tillbaka till föregående QR-kod.

[1] Payman Tehrani, Academic field sales engineer, National Instruments, möte den 11/5-15



Figur 5.6: Flödesschema för hur LiDAR:n arbetar

Kapitel 6

Anordning för transport av material

Som ett led i utvecklingen av ett autonomt lagerhanteringssystem krävs det en lösning för att AGV:n skall kunna förflytta lagerinredningen. Prototypen som framtas i projektet ska klara av att flytta bordet som finns att tillgå som lagerinredning. Följaktligen tillkommer då krav på utformningen och lastvikt som anordningen måste klara av, se appendix A.

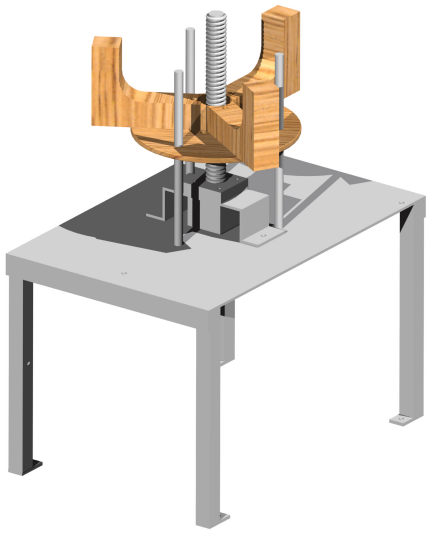
Tidigare projektgrupper har konstruerat och tillverkat en lyftanordning som kan monteras på AGV:n vilket gjorde att det fanns två vägar att gå; antingen utveckla och konstruera en ny design eller använda den befintliga lyftanordningen. En utvärdering kring befintligt prototyp gjordes och jämfördes med alternativa lösningar vilken resulterade i avvägningen att utnyttja och vidareutveckla den redan existerande lösningen.

6.1 Framtagande av koncept

För att ge en tydlig överblick av hur den redan existerande prototypen såg ut beskrivs den nedan mer ingående tillsammans med en diskussion av för- och nackdelar med prototypen.

6.1.1 Befintlig prototyp

Det befintliga konceptet bygger på att en trapetsgängad skruv roteras med hjälp av en stegmotor som i sin tur förflyttar ett lyftstycke, se figur 6.1. Lyftstycket är gjort i trä och består av en skiva med tre stycken armar. Det är dessa som är i direktkontakt med objektet som ska lyftas. Stegmotorn sammankopplas med Arduino:n och därifrån fås signal för att antingen hissa eller sänka lyftstycket. Stegmotorn är fäst i trapetsskruven med en låsskruv och vid signal börjar den rotera och därmed roterar även trapetsskruven. En flänsmutter fästs på lyftstycket och trapetsskruven roterar igenom muttern. Lyftstycket är förhindrat från att rotera på grund av de tre styrstängerna som löper igenom lyftstycket vilket gör att när stegmotorn körs kommer lyftstycke förflyttas uppåt eller nedåt längs trapetsskruven. Plattformen lyftanordningen står på gjordes för att inte störa de sensorer som utnyttjades för navigering och för att få plats med fler komponenter på chassits begränsade yta.



Figur 6.1: Modell av föregående års prototyp av anordningen för transport av material

Beräkningar gjordes under det föregående årets arbete vid framtagandet av prototypen, främst för val av motor och då undersöktes de vridmomentet som krävdes för att hissa lyftstycket, se ekvation 6.1 och 6.2. Som det går att se på ekvationerna så kommer den begränsande ekvationen vara 6.1, det vill säga momentet som krävs för upplyftningen av lyftstycket.

$$T_{raise} = \frac{Fd_m}{2}\left(\frac{l + \pi\mu d_m}{\pi d_m - \mu l}\right) \tag{6.1}$$

$$T_{lower} = \frac{Fd_m}{2}\left(\frac{\pi\mu d_m - l}{\pi d_m + \mu l}\right) \tag{6.2}$$

- T: moment som krävs för jämnvikt
- F: kraften som ligger på flänsmuttern
- d_m: medeldiametern på trapetsskruven
- l: stighöjden för ett varv på trapetsskruven
- μ: friktionskoefficienten mellan flänsmuttern och trapetsskruven

Alla de ingående variablerna i ekvationerna är relativt enkla att bestämma förutom friktionskoefficienten. Friktion uppkommer mellan trapetsmutter och trapetsskruven men även mellan lyftstycket och styrstängerna i viss mån. Detta medför en stor osäkerhet i beräkningarna vilket leder till att det bör tas till rejäl säkerhetsmarginal vid val av motor. Eftersom tidigare arbeten inte kom längre än vad som teoretiskt krävs betyder det att tester måste utföras för att fastställa om den befintliga stegmotorn klarar av belastningen i praktiken.

Om valet görs att utnyttja den befintliga anordningen erhålls en tids- och kostnadseffektiv lösning vilket är av stor vikt då tiden som finns tillgänglig för projektet är högst begränsad. Designen som konstruktionen bygger på är relativt enkelt men ändå lämpad för ändamålet. Tre kontaktpunkter med objektet som lyfts medför stabilitet och användningen av stegmotor möjliggör vetskapen om vilket avstånd lyftstycket flyttats samt att den bibehåller sin position. Att trapetsskruvar redan utnyttjas i liknande användningsområden, exempelvis skruvdomkrafter, visar på att metoden fungerar och är väl beprövad [37].

En jämförelse mellan den befintliga lösningen och alternativa metoder gjordes, där bland annat domkraft, saxlyft och pneumatiskt lyftbord undersöktes för att få en bild av andra lösningar som skulle generera samma resultat. Att utveckla och designa ett nytt koncept ansågs dock vara väldigt tidskrävande samtidigt som det var högst osäkert att ett bättre resultat skulle uppnås. Då det befintliga konceptet ansågs vara en robust konstruktion och väl lämpad för ändamålet med tidigare nämnda argument som grund för beslutet gjordes avvägningen att utnyttja den existerande prototypen. Några förbättringsområden identifierades dock och modifieringar av prototypen utfördes, se nästkommande kapitel.

6.2 Design av koncept

Vid utvärdering av det befintliga konceptet identifierades förbättringsmöjligheter på olika områden och modifieringar arbetades fram för att effektivisera och förbättra lyftanordningen.

6.2.1 Modifieringar

De delar som identifierades som förbättringsområden och modifierades var följande:

- *Lyftstycke* - För att förbättra lyftstabiliteten förlängdes armarna så att kontaktytorna med bordet förflyttades längre ut från centrum. Till följd av detta gjordes skivan som armarna sitter på större för att agera stöd och fördela krafterna som bordets tyngd ger upphov till på lyftarmarna.
- *Styrstänger* - På det befintliga konceptet gick styrstängerna endast igenom skivan vilken gjorde att lyftstycket blev instabilt. Genom att flytta styrstängerna till att gå igenom både skivan och armarna fås mer styrning och därmed en stabilare lyftanordning. För att få en tydlig bild av skillnaden hänvisas läsaren att jämföra figur 6.1 med figur 6.3.
- *Placering av lyftstycke på plattform* - Lyftstycket var ursprungligen centrerat på plattformen men då plattformen inte är centrerad på AGV:n så skedde inte lyftet mitt under bordet samt att tyngden inte fördelades jämt över hjulen på AGV:n. Därför förflyttades lyftstycket framåt på plattformen, jämför figur 6.1 med figur 6.3.
- *Sänkning av plattform* - Plattformen var ursprungligen gjord för att ha plats till lasern som användes för navigering av tidigare års arbeten, vilket gjorde att plattformen var onödig hög vid navigering med QR-koder. Höjden på plattformen ledde till att AGV:n blev instabil vid lyft och därför förkortades lyftplattformens ben vilket resulterade i en lägre tyngpunkt och en förbättrad stabilitet.

6.3 Färdigt koncept

En förklaring av de ingående komponenterna och dess användningsområde i det färdiga konceptet presenteras nedan.

Trapetsskruv
En trapetsgängad skruv används för att omsätta den roterande rörelsen från stegmotorn till lyftstyckets linjära rörelse. Skruven är gjord i stål och fästs i motorn med hjälp av en låsskruv.

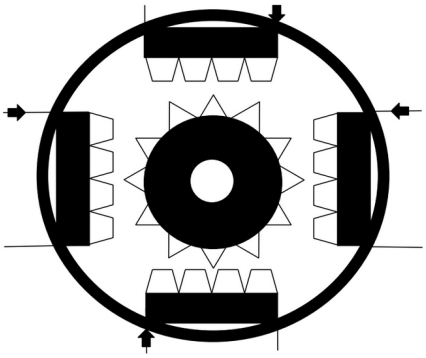
Lyftstycke
Lyftstycket är den del som är i direkt kontakt med bordet som ska lyftas och utgörs av tre armar som fästs med jämna mellanrum på en skiva. Tre kontaktpunkter med bordet medför ett stabilt lyft och alla ingående delar är tillverkade i trä för att det är enkelt att bearbeta samt för att hålla nere vikten.

Styrstänger
Styrstänger går igenom både skivan och armarna på lyftstycket för att minimera instabilitet samt ta upp momentet som skapas mellan flänsmutter och trapetsskruv. Styrstängerna är gjorda i solitt stål som är gängade i änden för fästning med skruv.

Motorhållare
En motorhållare gjord i bockad plåt används för att undvika att stegmotorn utsätts för kraft- och momentbelastning eftersom den inte är konstruerad för att hantera detta.

Stegmotor
Vid användning av stegmotor fås en hög precision på rotationen vilket möjliggör att ingen återkoppling på hur högt lyftstycket har höjts eller sänkts, utan den kan förutbestämmas. När en av de fyra elektromagneterna strömsätts flyttas den kuggformade rotorn ett steg vilket ger upphov till rotation och antalet steg som krävs för ett helt varv är beroende på motortyp. Stegmotorn drivs av en separat drivkrets som sköter strömfördelningen mellan faserna från en extern spänningskälla. Drivkretsen styrs i sin tur av Arduino:n som sätter antalet steg och riktning som skall köras. För en principskiss över en stegmotor, se figur 6.2.

Plattform
Plattformen är utformad som ett bord; en platta gjord i två millimeter tjock plåt med fyra stycken kvadratiska ben i stål. Benen är hopsvetsade med plattan och fästes på ramen på AGV:n.



Figur 6.2: Principsskiss av en stegmotor [12]

6.4 Implementering

Eftersom stora delar av den befintliga lyftanordning ansågs robust och lämpad för ändamålet förenklades tillverkningen och fokus kunde läggas på de områden där förbättringsmöjligheter identifierats. I CATIA konstruerades modeller och ritningar upprättades som grund för tillverkningsprocessen, se appendix E. Tillverkningen bestod av bearbetning av stål, trä och plåt och flera tillverkningsmetoder utnyttjades, bland annat fräsning, bockning och borrning.

Eftersom modifieringar gjorts på lyftanordningen genomfördes tester för att undersöka hur väl den modifierade lösningen klarar av att utföra den givna uppgiften. Stegmotorn, med hållkraften 0.098 Nm, var inte tillräckligt stark för att lyfta den av kravspecifikationen definierade lasten utan klarade en vikt på ungefär ett kg, se appendix A. För att åtgärda problemet testades först att minska friktionen genom att smörja trapetsskruven och styrstängerna, detta var dock inte tillräckligt effektivt.

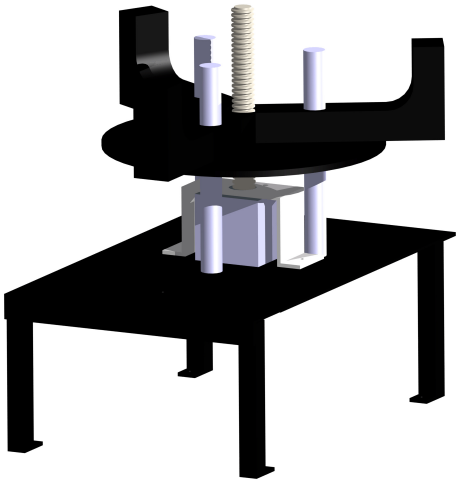
Med alla andra ingående variabler kända i ekvation 6.1, kunde friktionskoefficienten beräknas, se appendix D. Utifrån beräkningen togs ett teoretiskt momentkrav på en ny motor fram. En ny stegmotor införskaffades med hållkraften 1.1 Nm vilket innebar att den var cirka tio gånger starkare än den föregående stegmotorn.

Den nya stegmotorn var dock betydligt större, sett till både axelns samt motorhusets dimensioner, vilket ledde till att ytterligare modifieringar var tvungna att göras. Trapetsskruven gick inte att få på stegmotorns axel och därför gjordes hålet på trapetsskruven större. De var samma problem med motorhållaren som håller motorn på plats på lyftplattformen, den var för liten och en ny tillverkades. I helhet medförde modifieringen små förändringar och därmed att lyftanordningens utformning i stort sett blev som planerat.

Modifieringarna som förklaras under kapitel 6.2.1 genomfördes under labbtimmar i verkstaden och för att höja den estetiska upplevelsen lackades allt förutom styrstängerna svart.

När väl nämnda modifieringar var gjorda monterades lyftanordningen ihop och sattes på plats på AGV:n för att integreras med resterande delsystem. Tester utfördes för att undersöka hur väl lösningen uppfyllde de satta kraven och önskemålen i kravspecifikationen, resultatet redovisas under kapitel 7.

Slutligen erhölls en lyftanordning som efter modifieringar tillåter ett stabilare lyft i jämförelse med den ursprungliga prototypen. I figur 6.3 ses en bild på det slutgiltiga konceptet.



Figur 6.3: Det slutgiltiga konceptet

Kapitel 7

Integrering av delsystem

Som en avslutande del i utvecklingen av ett autonomt lagerhanteringssystem behöver alla delprocesser som beskrivits tidigare integreras. För att de olika delsystemen (navigering, överordnade systemet, kollisionsundvikning samt lyftanordning) på ett effektivt sätt ska fungera ihop måste kommunikationen mellan de olika delarna utformas. Vad som skickas emellan de olika komponenterna och hur det går till beskrivs mer ingående och även de justeringarna som gjordes för att möjliggöra integreringen. Nedan följer en genomgång av integreringsarbetet samt lite mer ingående förklaringar och motiveringar till val som gjorts.

7.1 Förflyttning

För att förflytta AGV:n mellan koordinatsystemets kända positioner behövde hänsyn tas till de olika delsystemen som skall kunna verka samtidigt och tillsammans. Flera kompromisser krävdes för att möjliggöra integreringen av dessa delsystem.

7.1.1 Död räkning

För att förflytta sig mellan koderna behöver AGV:n information om antingen avstånd eller tid och hastighet. Arduino:n klarar att flytta AGV:n en given sträcka via feedback från sina encoders. Ett alternativ till detta är att skicka körkommandot till Arduino:n och efter en viss tid skicka ett stoppkommando.

Eftersom förflyttningen via en angiven sträcka innebar att Arduino:n var tvungen att tolka motorenas encoders under hela förflyttningen kunde inte eventuel-

la stoppkommandon från serieporten upptäckas. För att kollisionsundvikningen skulle kunna implementeras krävdes att serieporten kunde läsa under förflyttning. Detta gjorde att död räkning implementerades hos myRIO:n för att bestämma förflyttningens avstånd. Se kap 7.1.2 för uträkningar gällande tider.

För att öka systemets effektivitet genom att minska transporttiderna utvärderades möjligheten för AGV:n att förflytta sig dubbla avstånd innan den positionsåterkopplar. För att lösa detta programmerades servern att vid längre sträckor skicka en siffra efter förflyttningskommandot. Denna siffra används som en multipel för den tid som AGV:n sedan färdas. Hänsyn var tvungen att tas till den förlorade sträckan som AGV:n färdas under acceleration och retardation varvid detta tidsbidrag adderades till den totala tiden att färdas.

7.1.2 Matematisk modell vid förflyttning

Då funktionerna som finns implementerade i Arduino:ns programbibliotek kräver en tid för att accelerera och retardera när en förflyttning utförs krävdes en matematisk modell. Modellen användes för att beräkna tiden som rörelsen behövde utföras för att få roboten att förflytta sig en viss sträcka i sidled, längdled och vinkel vid rotation.

Modellen bygger på ett antagande om linjär acceleration och retardation samt konstant fart där emellan, se figur 7.1. För att beräkna den tid som krävs för att förflytta sig en viss sträcka ställdes ekvation 7.1 upp.

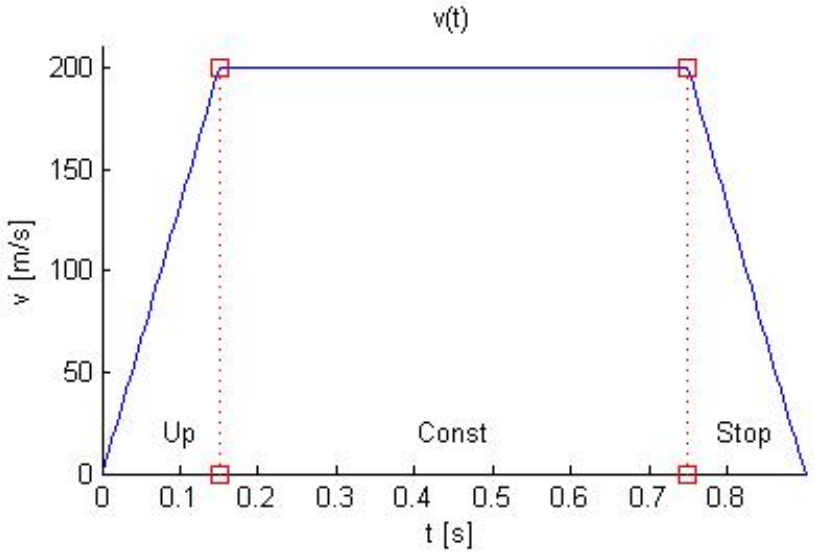
$$s(t) = \int_{t_{up}} \frac{v}{t_{up}} t \, \mathrm{d}t + \int_{t_{konst}} v \, \mathrm{d}t + \int_{t_{stop}} v - \frac{v}{t_{stop}} t \, \mathrm{d}t \tag{7.1}$$

Datan från figur 7.1 visas i tabell 7.1

Tabell 7.1: Data från figur 7.1

Parametrar	**variabel**	**Värde**	**Enhet**
Hastighet	v	200	mm/s
Up	t_{up}	0.15	s
Const	t_{const}	0.6	s
Stop	t_{stop}	0.15	s
Distans	s	150	mm
Minst dist.	–	30	mm

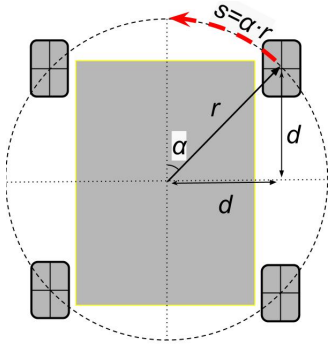
Då $t_{up} = t_{stop}$ kan t_{const} lösas ut och då fås ekvation 7.2.



Figur 7.1: Graf över AGV:ns generella rörelse

$$t_{const} = \frac{s}{v} - t_{up/stop} \tag{7.2}$$

För att kunna rotera en given vinkel togs även modell 7.2 fram, som bygger på samma modell som tidigare, men istället justeras robotens vinkelfel i förhållande till avläst QR-kod. Då $s = \alpha\, r$ kan ekvation 7.2 användas för att bestämma den tid som krävs för att utföra en rotation av angiven vinkel. En grafisk illustration återfinns i figur 7.2.



Figur 7.2: Grafisk beskrivning av rotationsmodell för AGV:n, $d = 150mm$

Då det alltid krävs en acceleration och retardation vid förflyttning kommer det uppstå en teoretiskt minsta sträcka som roboten kan förflytta sig. Den sträcka

kan beräknas med ekvation 7.2 då $t_{const} = 0$. Denna minsta sträcka är relevant då toleranserna behöver bestämmas för när justeringar skall genomföras. Med ekvation 7.3 beräknas det minsta förflyttningsavståndet.

$$s = v\, t_{up/stop} \tag{7.3}$$

Teoretiskt skulle det minsta förflyttningsavståndet vara noll och ett optimalt system för justering vid QR-koder skulle kunna uppnås. Fysiska begränsningar i hårvaran begränsar dock detta. Under utvecklingsfasen testades dessa parametrar för att ge ett optimalt förhållande mellan accelerationstid och fart. Då robotens förflyttning är som minst vid justeringsfaserna minskas v och $t_{up/stop}$ jämfört när förflyttning sker under längre sträckor. Resultatet presenteras i tabell 7.2 och tabell 7.3.

Feljusteringens krav beräknades för att med säkerhet kunna hålla största möjliga toleranser utan att resultatet av förflyttningen blir lidande. Beräkningarna baseras på ”worst case” scenario för en förflyttning i längdled vilket förutsätter följande:

- Den avlästa QR-koden befinner sig precis innanför toleransgränsen vilket gör att ingen justering genomförs.
- Förflyttningen sker över två QR-koder, det vill säga det dubbla avståndet.

Detta fall illustreras i figur 7.3. Om QR-kodens centrumpunkt befinner sig inom det gröna området behöver ingen positionskorrigering göras, vilket krävs om den är inom det vita området. Befinner sig centrumpunkten i det röda området måste felsökningssekvensen köras.

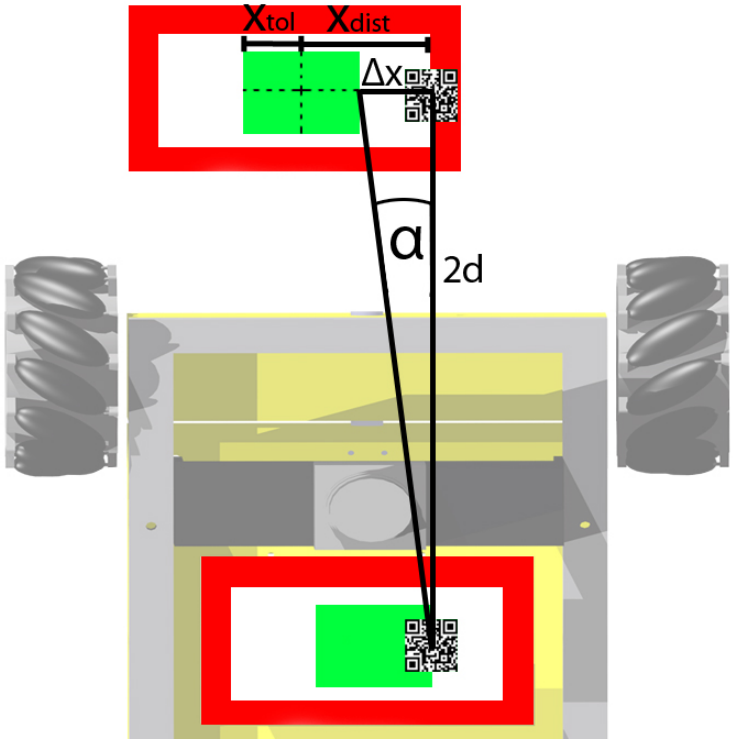
Toleransförhållandet beräknas enligt följande ekvation:

$$\alpha = \arctan \frac{\Delta x}{2d} \tag{7.4}$$

Ur denna ekvation fås ett linjärt samband vilket ses i figur 7.4. I grafen representerar linjen alla de kombinationer av toleranser som utgör gränsvärden för att säkert träffa nästa QR-kod.

Vid implementationen visade sig denna teoretiska modell något felaktig, antagligen på grund av oriktighet gällande avstånden för kameran. Därför valdes en toleranskombination som låg något innanför gränsvärdet. Punkten i grafen visar valda värden för projektet.

För att få ett bra flöde för AGV:n genom systemet är kravet direkt kopplat till hur många sträckor som körs innan felet blir så stort att QR-avläsningen kommer

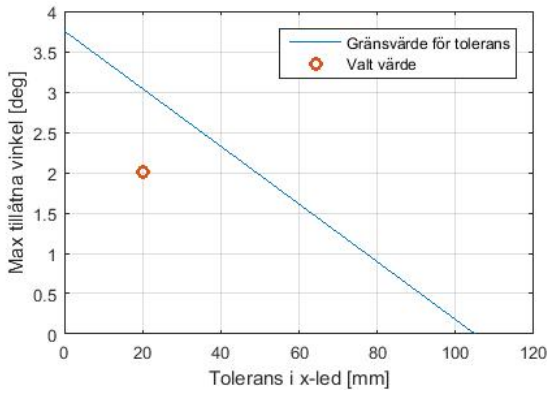


Figur 7.3: Illustration av ”worst case” scenario vid toleransberäkningar

utanför synfältet för kameran och felsökningssekvensen behöver genomföras för att hitta tillbaka till koden. Därför är det av stor vikt att förbättra toleranserna för att på sikt kunna låta AGV:n röra sig så långa sträckor som möjligt.

Tabell 7.2: Data vid justering av rotation vid QR-kod

Parametrar	**Värde**	**Enhet**
Hastighet	75	mm/s
Up	0.1	s
Const	0.6	s
Stop	0.15	s
Vinkel	10	deg
Minst vinkel	2.02	deg



Figur 7.4: Gräns för tolerans

Tabell 7.3: Data vid justering sidleds och längdleds vid QR-kod

Parametrar	**Värde**	**Enhet**
Hastighet	75	mm/s
Up	0.1	s
Const	1.9	s
Stop	0.1	s
Distans	150	mm
Minst dist.	7.5	mm

7.1.3 Reglering av motorer

För att möjliggöra en stabil förflyttning där AGV:ns hjul varken riskerar att slira eller hålla olika hastigheter krävdes en aktiv reglering av motorerna. Då motorerna är försedda med encoders användes dessa för att ge feedback till Arduino:ns PID-regulatorer. Parametrarna för regulatorn återanvändes från förra årets arbete, se ekvation 7.5 [12].

$$K_p = 0.31,\ K_i = 0.01,\ K_d = 0.1 \tag{7.5}$$

En begränsning upptäcktes hos Arduino:n i och med att dess serieport både behövde användas till att kommunicera med myRIO:n och till motorregleringen. Detta betydde att Arduino:n inte kunde läsa kommandon från myRIO:n och samtidigt reglera sina motorer. För att lösa problemet med befintlig setup justerades mjukvaran. Arduino:n programmerades så att den reglerar motorerna i 100 ms och där efter läser serieporten efter nya kommandon, för att sedan iterera vidare. Under tiden läsningen av serieporten drivs motorerna utan aktiv reglering. Detta var dock

inget som påverkade resultatet av förflyttningen utan samma stabila gång som vid full reglering erhölls.

7.2 Kommunikation

Då de olika enheterna programmerats avskilt från varandra under utvecklingen behövdes protokoll för kommunikationen införas. För att inte belasta WiFi i onödan beslutades det att minimera kommunikationen och endast innefatta det mest nödvändiga.

7.2.1 Protokoll mellan myRIO och överordnat system

För att upprätthålla en trådlös kommunikation mellan det överordnade systemet och AGV:n valdes Transmission Control Protocol (TCP). Valet i detta fall var enkelt då TCP till skillnad från andra nätverksbaserade protokoll garanterar att informationen kommer fram och att informationen kommer fram i rätt ordning.

Protokollet enligt tabell 7.4 utformades för att definiera de kommandon som krävdes för att det överordnade systemet skulle ha möjlighet att koppla upp och styra myRIO:n. Alla kommandon som skickas mellan myRIO:n och servern avslutas med ”\r\n” vilket är en sekvens av CR (Carriage Return) och LF (Line Feed) för att på så sätt definiera slutet av kommandot.

Tabell 7.4: Protokoll mellan server och myRIO

Nr	**Server**	**myRIO**
1	-	LOGIN <name>
2	PING	PONG
3	STOP	OK
4	NORTH <step>	READY <code>
5	SOUTH <step>	READY <code>
6	EAST <step>	READY <code>
7	WEST <step>	READY <code>
8	LIFT	OK
9	LOWER	OK

Systemet är konstruerat så att myRIO:n kontinuerligt läser TCP porten och när ett kommando registreras från servern exekverar myRIO:n den matchande funktionen och meddelar servern dess status när funktionen är slutförd.

För att verifiera uppkopplingen mellan servern och myRIO:n, används kommandona PING och PONG. Servern skickar ett PING var 25:e sekund där myRIO:n svarar med PONG. Skulle myRIO:n inte svara, stänger servern TCP anslutningen och myRIO:ns avslutningssekvens inleds, se kapitel 7.5.3.

7.2.2 Protokoll mellan myRIO och Arduino

Då den medföljande Arduino:ns kommunikationsmöjlighet med externa enheter är begränsad till den enda seriella porten, användes den för att utföra all dataöverföring som krävs mellan myRIO:n och Arduino:n. Ett protokoll mellan myRIO:n och Arduino:n utformades enligt tabell 7.5 för att definiera de funktioner som krävs för styrning av Arduino:n. Kommandona avslutas med ett ; för att definiera slutet av varje kommando.

Tabell 7.5: Protokoll mellan myRIO och Arduino

Nr	**myRIO**	**Arduino**
1	SETSPEED <mm/s>	D
2	SETUPTIME <ms>	D
3	SETREGULATE <ms>	D
4	HIGHT <mm>	D
5	ROTATE <rad>	D
6	ADJROT <rad>	D
7	ADJFB <mm>	D
8	ADJRL <mm>	D
9	LIFT	D
10	LOWER	D
11	STOP	D
12	FORWARD	D
13	BACKWARD	D
14	LEFT	D
15	RIGHT	D

När kommando 1-10 skickas från myRIO:n väntar den på att Arduino:n skall slutföra kommandot och svara med D (en förkortning på ”Done”) innan den fortsätter i programmet. Anledningen till att Arduino:n svarar med D är att myRIO:n behöver respons för att veta när programmet skall fortsätta exekvera. I 12-15 skickar myRIO:n kommando, utför dödräkning och skickar STOP (kommando 11) när dödräkningen slutförts.

7.2.3 myRIO

Flödeschemat i figur 7.5, ger en övergripande bild över hur myRIO:n fungerar för att förtydliga dess process. Efter att ha initierats och loggat in i systemet avvaktar den centrala styrenheten, myRIO:n, tills något av de sex möjliga kommandon mottas från det överordnade systemet.

Om en riktning skickas från det överordnade systemet, konverterar myRIO:n först den mottagna globala riktningen till en lokal riktning, exempelvis från ”North” till ”backward” om AGV:n är vänd mot ”South”, och därefter skickas den lokala riktningen till Arduino:n. Om inte LiDAR:n finner ett hinder, skickas ett ”STOP”-kommando först när tiden för död räkningen är slut. Då försöker myRIO:n läsa av eventuell QR-kod för att sedan delge den avlästa informationen till det överordnade systemet. När informationen har skickats avvaktar myRIO:n återigen tills en nytt kommando fås.

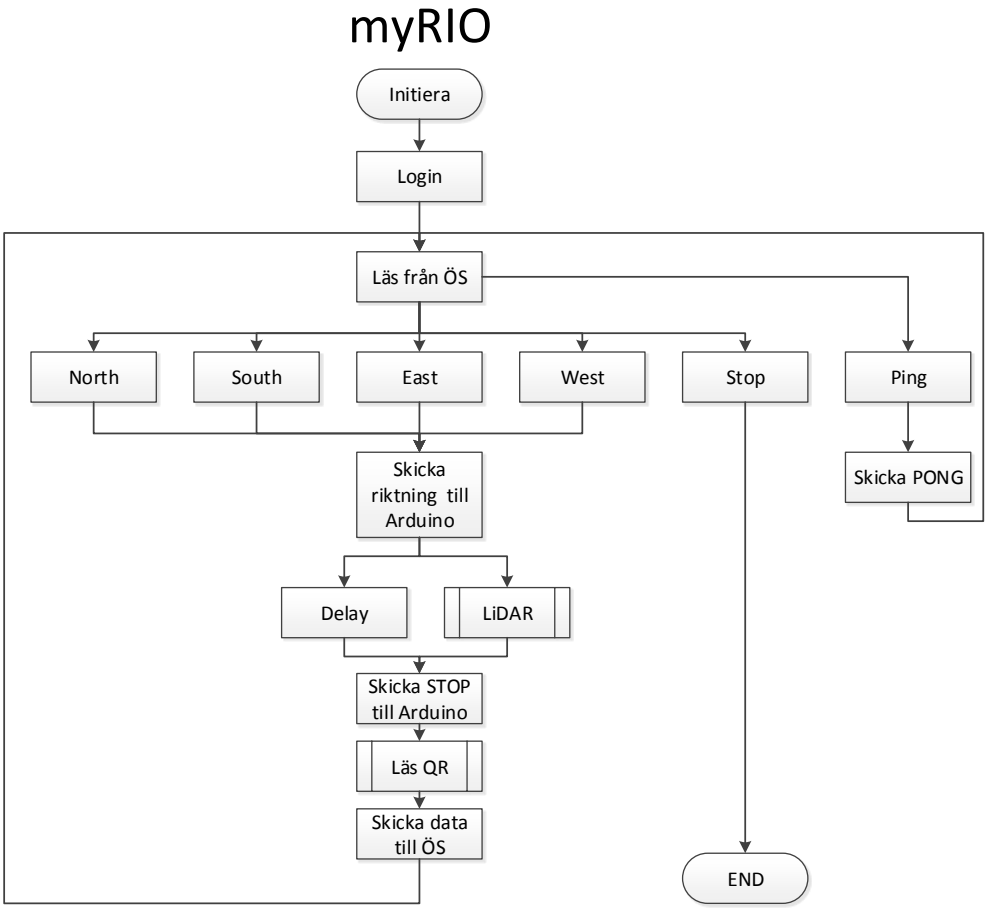
Utöver de fyra riktningarna, finns två kommandon till som myRIO:n kan behandla. Som en typ av nödstopp kan det överordnade systemet skicka ”Stop” till myRIO:n vilket innebär att AGV:n stannar och kopplas ur systemet. Detta är inte reversibelt utan när det görs måste AGV:n startas om för att den skall kopplas tillbaka in i systemet. Det sista kommandot är ”Ping” vilket existerar för att säkerställa att kontakt finns, vidare förklaring hittas under avsnitt 7.2.1.

7.3 Initieringsprocesser

För att AGV:n skall fungera vid start krävs att två krav ska vara uppfyllda. Det första är att servern i systemet måste vara aktiv och vänta på att AGV:er skall koppla upp sig. Det andra är att roboten skall vara placerad över valfri QR-kod inom avläsningsområde för kameran så den kan meddela sin nuvarande position.

När dessa krav är uppfyllda kan AGV:n startas och initieringsprocesserna kan påbörjas enligt följande:

1. myRIO:n sätter Arduino:ns parametrar så som hastighet, accelerationstid, retardationstid, regleringstid och lyfthöjd för lyftanordning. Dessa parametrar var viktiga att kunna justera för att smidigt ändra robotens beteende utan att koppla upp datorn till Arduino:n.
2. myRIO:n söker efter servern på nätverket och öppnar TCP anslutningen till det överordnade systemet genom att skicka LOGIN <namn>.
3. LiDAR:n och webbkameran initieras.



Figur 7.5: Flödesschema över hur myRIO:n arbetar

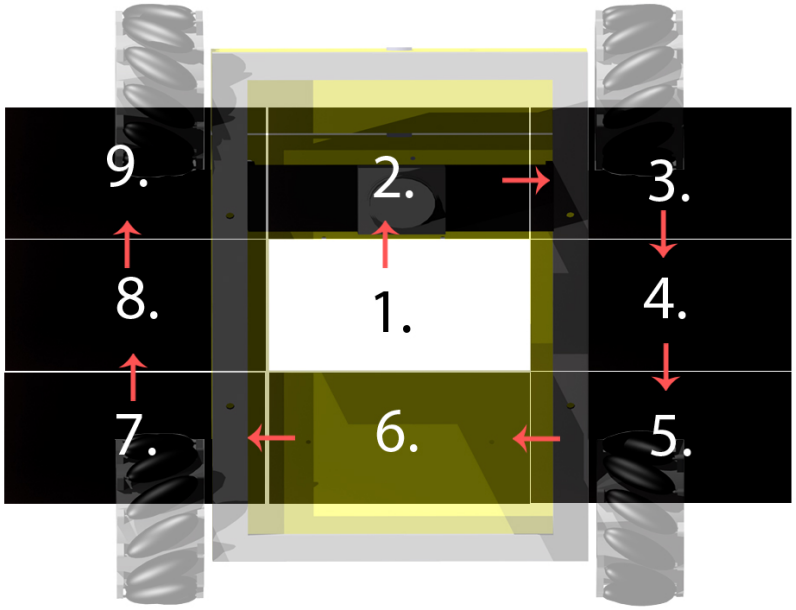
4. QR-koden läses och roboten justerar sin position och meddelar servern sin nuvarande position.

AGV:n är nu i viloläge och inväntar nu order från det överordnade systemet.

7.4 Felsökningssekvens

För att systemet autonomt ska kunna lösa eventuella positioneringsfel skapades en felsökningssekvens för att lätt hitta tillbaka till en känd punkt i koordinatsystemet. Sekvensen baserades på tester som visade att QR-koderna som inte upptäcktes låg i 100 % av fallen maximalt inom ett synfälts avstånd. I samband med det faktum att QR-koderna avlästes stillastående valdes en sekvens där AGV:n förflyttar sig för att sedan ta en bild som analyseras. Återfinns en QR-kod så justerar AGV:n

och fortsätter som vanligt. Om en QR-kod saknas så sker en ny förflyttning.



Figur 7.6: Illustration av felsökningssekvens

Det rörelsemönster som valdes sker i en ökande rektangel klockvis, se figur 7.6. Detta mönster ansågs som ett mer systematiskt sätt att öka synfältet än exempelvis ett cirkulärt rörelsemönster. För att säkerställa att inga halva QR-koder uppstår vid bildkanterna så valdes en förflyttningsdistans som gav en överlappning av synfältet motsvarande en QR-kod.

Denna sekvens genomförs efter varje förflyttning om ingen QR-kod hittats i den första bilden och pågår tills en QR-kod funnits eller det totala sökfältet om nio rutor genomsökts. Valet av nio rutor baseras på det tidigare nämnda testet som visade på att i 100 % av fallen som felsökningssekvensen utnyttjades befann sig QR-koden inom ett synfälts avstånd. Sökfältet är också tillräckligt litet för att AGV:n skall röra sin inom den ruta som det överordnade systemet binder upp till AGV:n, se kapitel 4.5. Detta för att AGV:n inte ska inteferera med övriga AGV:er som rör sig runtom. Om ingen QR-kod, mot förmodan, hittas inom de nio rutorna skickas ett felmeddelande till operatören att AGV:n är ur kurs och manuell justering krävs. När felsökningssekvensen körs utnyttjas inte kollisionsundvikningssystemet eftersom sträckorna är väldigt korta samt att LiDAR:n inte kan skanna av bakom AGV:n.

7.5 Systemöverblick

För att beskriva den helhet som integreringen av alla delsystem utgör beskrivs nedan hur implementeringen skedde. En cykel består av uppstart och en hämtning och återlämning av ett bord.

7.5.1 Uppstart

Vid uppstart av systemet behöver servern och AGV:n startas och placeras ovanför en QR-kod. AGV:n kommer genomföra de initieringsprocesser som beskrivs i kapitel 7.3. När dessa processer är genomförda är AGV:n uppkopplad mot servern och dess position relativt det globala koordinatsystemet meddelas.

7.5.2 Arbetscykel

Under en arbetscykel kommer AGV:n att hämta och lämna ett bord. Cykeln kommer att börja med att operatören lägger sin order i gränssnittet genom att meddela vilken artikel som behövs, om fler artiklar behövs kan även de meddelas samtidigt. Servern kommer sedan, för en tillgänglig AGV, att beräkna den rutt i det globala koordinatsystemet som ger den kortaste förflyttningen enligt Manhattanmetoden.

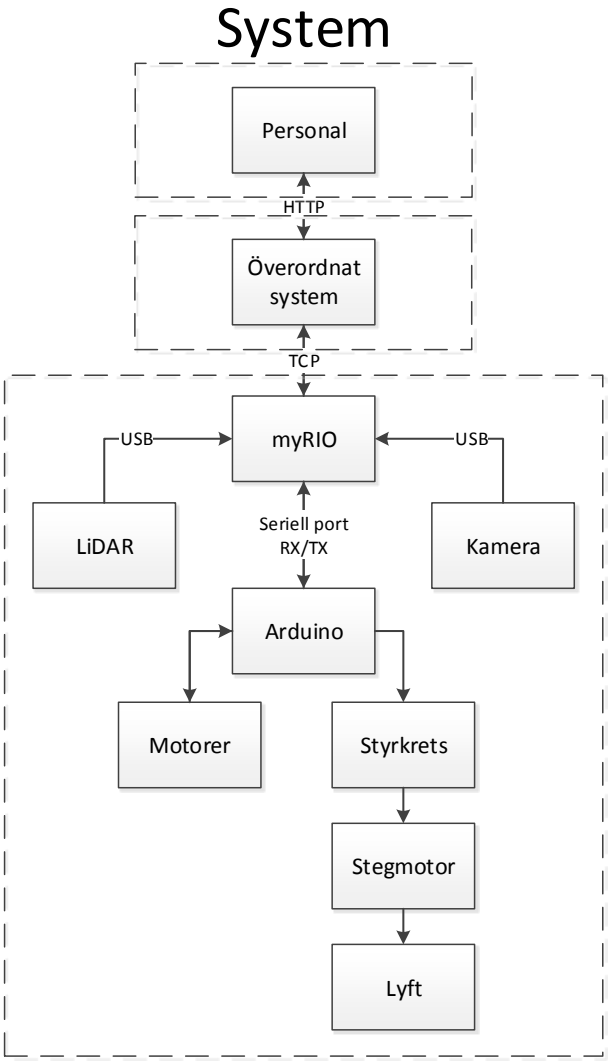
När rutten är planerad kan AGV:ns position i förhållande till nästa punkt i rutten utläsas och utifrån detta kan ett kommando om förflyttning skickas, exempelvis ”North” om nästa punkt skulle befinna sig norr om AGV:n.

AGV:n mottar kommandot och exekverar sitt program. Först kontrollerar AGV:n att den står i rätt väderstreck, om inte så korrigerar den detta. Sedan skickas ett kommando om rörelse framåt. AGV:n börjar då åka framåt och gör detta tills den mottar ett stoppkommando från antingen död räkningen eller LiDAR:n. Förutsatt att inget hinder finns bör AGV:n vara placerad över en QR-kod och kan sedan justera sig för att hålla en centrerad position över QR-koden. Samtidig skickas information om den nya positionen till servern.

Servern kan nu uppdatera sin karta och den planerade rutten. Om AGV:n ännu inte nått sin destination så kommer servern beräkna nästa förflyttning och ovanstående delar upprepas. Om AGV:n befinner sig på den sista positionen betyder det att AGV:n står under en hylla. Då skickas ett lyftkommando och hyllan lyfts.

Därefter beräknas en ny rutt för att föra AGV:n med hyllan till plockstationen. Förflyttningarna som följer genomförs på samma sätt som ovan. Vid plockstationen

kan operatören plocka det antal som önskas och sedan kvittera ordern. AGV:n kommer då att föra hyllan till sin ursprungliga plats där den sedan placeras. Efter hyllan placerats kommer AGV:n att stå still och invänta nästa order.



Figur 7.7: Schematisk bild över styrhierakin i hela systemet

7.5.3 Avslutning

Då servern antingen skickar stopp eller stänger TCP anslutningen, kör myRIO:n avslutningsprocessen. Då stänger myRIO:n anslutningarna till LiDAR:n, kameran och servern. Efter det skickas även STOP till Arduino:n för att säkerställa att

AGV:n står still. I figur 7.7 återfinns en schematisk bild över styrhierarkin i det skapade systemet. Det överordnade systemet fungerar något som en brygga mellan användare och system, själva AGV:n är alltså helt automatiserad.

Kapitel 8

Resultat

Överlag uppnåddes målen för projektet och en fungerande prototyp har skapats. För att tydliggöra projektresultatet redovisas i detta kapitel resultatet i form av mål- och kravåterkoppling. Här redovisas även de olika tester som genomförts för att utvärdera de olika delsystemens funktionalitet. För att få se en bild av den slutgiltiga prototypen hänvisas läsaren till framsidan av rapporten.

8.1 Måluppfyllnad

Utöver att det skapade systemet skulle kunna hantera flera AGV:er var de primära målen att en AGV skulle skapas med förmågan att:

- Autonomt navigera mellan kända positioner.
- Ta order om att utföra leveranser från ett överordnat datorsystem.
- Undvika att kollidera med människor och annan rörlig utrustning inklusive andra AGV:er.
- Utrustas med ett system för att automatiskt hämta och lämna lagerinredning.

Det framtagna systemet baseras på användning av QR-koder för att få feedback att reglera mot. Eftersom koderna måste placeras i ett rutnät begränsas flexibiliteten i systemet. Navigationen blir dock mycket precis tack vare den kontinuerliga återkoppling och som en följd av detta minskas risken att tappa rutten.

Ett överordnat system håller ordning på positionen hos AGV:erna, QR-koderna samt lagerinredningen. Genom att använda sig av ruttplaneringsalgoritmen A* kan

det överordnade systemet delge information till AGV:n om hur den ska köra och därmed kan AGV:n, med det överordnade systemets hjälp, navigera autonomt.

För att kunna ta emot leveransordrar tillhandahåller det överordnade systemet ett webbaserat gränssnitt som tillåter operatören att lägga ordrar på vilka varor som skall hämtas. Det överordnade systemet kommunicerar i sin tur rutten till en tillgänglig AGV. När artikeln har plockats från bordet kan ordern kvitteras och AGV:n returnerar bordet till dess ursprungliga plats och väntar sedan på nästa order.

För att fysiskt hämta och lämna materialet har en lyftanordning, som ursprungligen konstruerades av föregående års kandidatprojekt, modifierats och vidareutvecklats till en fullt fungerande lyftanordning som är integrerad med AGV:n.

Enligt det sista huvudsakliga målet skall AGV:n med hjälp av sensorer förhindra kollision med andra AGV:er, människor eller andra objekt som den stöter på. I enlighet med målet finns en lasersensor (LiDAR) som skannar området i AGV:ns rörelseriktning, vilket gör det möjligt att upptäcka oförutsedda hinder såsom människor och därigenom också stanna innan kollision. Problemet med AGV:er som stöter på varandra förebyggs även genom att det överordnade systemet håller ordning på alla AGV:ers positioner och inte tillåter en annan att åka till varken den position som en AGV senast befann sig på eller till en position som en annan AGV redan försöker åtkomma.

De huvudsakliga målen enligt ovan har alltså uppnåtts men det bör nämnas att inte alla deluppgifter som definierades, se kapitel 1.4, har lösts. För att förbättra ergonomi, lastningsmöjlighet, lastsäkerhet samt funktion var tanken att analysera och skapa en optimal lagerinredning vilket på grund av tidsbrist inte har gjorts. Inte heller har själva interaktionen med plockpersonal, som AGV:n hämtar varor till, hunnit analyseras. Den sista deluppgiften som inte lösts är att prototypen enligt den ursprungliga idén skulle kontrollera att rätt bord hämtas men av praktiska skäl gjordes antagandet att lagerinredningen inte förflyttas manuellt och därför inte behöver kontrolleras.

Utifrån målen som sattes för projektet skapades också en kravspecifikation för prototypen innehållandes olika krav som måste uppfyllas av en AGV för att den skall vara användbar men också önskemål för det som är önskvärt att AGV:n uppnår. Kravspecifikationen samt tester som utförts för att utvärdera funktionaliteten av prototypen nämns kortfattat senare i detta kapitel men för mer utförlig information hänvisas läsaren till appendix A och F.

8.2 Navigation

Som tidigare nämnts baseras navigationen på QR-koder där de placeras ut i ett rutnätsmönster med 800 mm mellanrum mellan de parallella koderna. Detta tillåter AGV:n att reglera sin position och orientering vid kodavläsningen. QR-koderna avläses med hjälp av en kamera placerad i centrum robotens samt ett program skapat i LabVIEW. På grund av att bilderna som kameran tar blir suddiga kan inte AGV:n köras i mer än 0,1 m/s vilket är lägre än önskemålet där endast AGV:ns maxhastighet skall vara den begränsande faktorn, se tabell 8.1. Det bör nämnas att istället för att försöka läsa koder i rörelse ändrades metodiken så att AGV:n står still över koden när bilden tas, se kapitel 3.4.3. Utöver hastighetsbegränsningen uppfylldes alla krav och önskemål ställda på delsystemet.

Tabell 8.1: Kravspecifikation för navigationssystemet

3	**Navigationssystem**	**Krav**	**Önskemål**	**Uppnått**
3.1	Lokaliseringsprecision	<50 mm felmarginal	<10 mm felmarginal	Önskemål
3.2	Hårdvarukostnad	<30000 kr	<15000 kr	Önskemål
3.3	Storlek	Skall rymmas på AGV		Krav
3.4	Vikt	<10 kg	<4 kg	Önskemål
3.5	Möjlig hastighet på robot	>0,05 m/s	>0,5 m/s	Krav

Som ett led i optimeringsarbetet kan AGV:n förflytta sig över flera QR-koder för att snabba upp processen. Om inte en kod hittas, där AGV:n stannar för att kontrollera sin position, startas autonomt en felsökningssekvens där den avsöker det angränsande området. Om fortfarande ingen kod hittas skickas information till det överordnade systemet om att kontakt har tappats som i sin tur uppmärksammar operatören om detta via gränssnittet.

För att utvärdera navigeringsförmågan gjordes tester på bland annat rörelseprecisionen och hur ofta felsökningssekvens behöver köras. Resultatet visar att i ungefär 6% av fallen var AGV:n tvungen att påbörja felsökningssekvensen för att hitta koden. Detta skedde främst när AGV:n körde dubbelt avstånd efter rotation. AGV:n tappade aldrig under testet sin rutt helt utan kunde utföra alla givna ordrar korrekt. Med mer tid kan förmodligen navigeringen optimeras ytterligare och därigenom få bättre flyt i systemet när felsökningssekvens inte behöver köras lika ofta. I tabellen 8.1 redovisas kraven på navigationssystemet och hur väl de är uppfyllda.

8.3 Kollisionsundvikningssystem

För att undvika kollisioner i systemen används en lasersensor, en LiDAR, som vid påträffande av ett hinder skickar information till myRIO:n som i sin tur stannar AGV:n. Programmeringen resulterade i att AGV:n agerar olika beroende på om hindret som påträffats flyttas eller är fast. Vid fasta hinder informeras det överordnade systemet som gör aktuell QR-kod indisponibel vid framtida ruttplanering. Hinderupptäckning fungerar som planerat men dock tar det lång tid från att hindret upptäcks tills dess att AGV:n stoppar. Detta beror på begränsningar i mjukvaran och är ett område med stor utvecklingspotential. Detta diskuteras mer under kapitel 9.

Tabell 8.2: Kravspecifikation för kollisionsundvikningssystemet

1	**Kollsionssystem**	**Krav**	**Önskemål**	**Uppnått**
1.1	Vikt på ingående komponenter	<5 kg	<1kg	Önskemål
1.2	Skall ej krocka	Skall stanna vid kollisionsrisk	Skall flytta undan från objekt i rörelse	Krav
1.3	Hårdvarukostnad	<30000 kr	<15000 kr	Önskemål
1.4	Storlek	Skall rymmas på AGV		Krav
1.5	Avskanning av omgivning	AGV-bredd i färdriktningen	360°	Krav
1.6	Skanningsavstånd	Längre än stoppsträckan		Krav
1.7	Skanningsfrekvens	5 ggr/sekund	10 ggr/sekund	Önskemål

Mer utförliga tester gjordes på AGV:ns stopptid för att fastställa funktionaliteten på systemet vilket gav ett medelvärde på under 1,5 sekund eller ungefär 300mm med AGV:ns hastighet på 200mm per sekund, se appendix F. Förhoppningen är dock att stoppsträckan minskar avsevärt med annan mjukvara eller uppgradering av hårdvaran. Eftersom skanningsavståndet är ungefär 800mm uppnås ändå det kanske viktigaste kravet enligt tabell 8.2, att skanningsavståndet är längre än stoppsträckan. Denna tidsfördröjning utgör den primära hastighetsbegränsningen för systemet.

8.4 Överordnat system

Det överordnade systemet är programmerat i språket Clojure och tillhandahåller ruttplanering, gränssnitt till användare, orderhantering och information om lagermiljön såsom var lagerinredning, hinder eller AGV:er befinner sig. I gränssnittet kan ordrar läggas på de artikelnummer som önskas och när de är hämtade till plockstationen kan användaren kvittera ordern för att resterande varor skall återlämnas till sin plats. Systemet har stöd för flera AGV:er och notifierar även användaren om något oväntat händer under körning i enlighet med kravspecifikationen, se tabell 8.3.

Tabell 8.3: Kravspecifikation för det överordnade systemet

5	**Överordnat system**	**Krav**	**Önskemål**	**Uppnått**
5.1	Stöd för hantering av flera AGV:er	>1		Krav
5.2	Notifiera användare vid problem	Notiser ges vid oväntade händelser		Krav
5.3	Karta i gränssnitt över systemet		Karta som uppdateras i realtid	Önskemål

För att uppfylla kravspecifikationens önskemål finns en realtidskarta som ger möjlighet att se orderlistan, var eventuella hinder har stötts på samt var AGV:erna, plockstationen och lagerinredningen befinner sig.

8.5 Anordning för transport av material

En vidareutveckling av den redan befintliga prototypen gjordes och resultatet blev en lyftanordning som baseras på att ett lyftstycke höjs och sänks med hjälp av en trapetsskruv och en stegmotor. Lyftanordningen placerades på en upphöjning för att möjliggöra användandet av kameran och LiDAR:n.

Med den nuvarande lyftanordningen finns nuvarande inget sätt att få feedback på vilken höjd lyftanordningen är vilket kan vara ett problem om AGV:n skall köra under ett bord och lyftanordningen är upphöjd. För att veta hur mycket lyftanordningen förflyttas i höjdled över ett antal ordrar gjordes en simulering med tio höjningar och sänkningar som resulterade i att lyftanordningen sänktes ungefär

Tabell 8.4: Kravspecifikation för anordningen för transport av material

2	**Anordning för transport av material**	**Krav**	**Önskemål**	**Uppnått**
2,1	Förflyttning ska ske stabilt	AGV:n skall inte tappa lagerinredningen	Ingen last skall tappas under färd	Krav
2.2	Hårdvarukostnad	<3000 kr	<1500 kr	Önskemål
2.3	Tid att lasta varor för transport	<10 sekunder	<5 sekunder	Krav
2.4	Storlek	Få plats på AGV med resterande utrustning		Krav
2.5	Möjlig lastvikt	>4 kg	>10 kg	Krav
2.6	Vikt på ingående komponenter	<8 kg	<4kg	Önskemål

1 mm. Eftersom det var så liten skillnad anses resultatet vara tillfredställande. Själva tiden för att höja upp ett bord, vilket påverkar hur snabb AGV:n är på att hämta material, var mindre än 8 sekunder enligt mätningar från samma test. För en prototyp är resultaten fullt acceptabla men för en kommersiell produkt bör lyfttiden minskas i största möjliga mån och någon form av positionsfeedback måste fås på lyftanordningens höjd eftersom den då skall fungera över ett mycket stort antal ordrar.

Angående vikten som lyftanordningen på prototypen kan lyfta så sattes ett önskemål på 10 kg i kravspecifikationen och krav på 4 kg, se tabell 8.4. Tester visar dock att den inte orkar mer än 10 kg men däremot utan problem klarar två bord vilket motsvarar ungefär 7 kg. För prototypen anses den möjliga lastvikten vara rimlig men bör höjas i kommersiell användning för att öka antalet varor som kan förvaras på varje lagerhylla. En betraktelse som gjordes var att viktfördelningen var en betydelsefull faktor eftersom en snedfördelning av vikten skapar större friktionskrafter och därmed tillåter en lägre last.

Det bör nämnas att det inte gjorts tillräckligt många tester för att säkerställa att varken lagerinredning eller last kan åka av från AGV:n, särskilt då den vibrerar under färd och lyftanordningen kan wobbla något. Dock har lagerinredningens utformning troligen stor betydelse i frågan, något som var tänkt att analyseras i projektet men som inte gjordes på grund av tidsbrist. Utifrån de testkörningar som gjorts bör dock inte lagerinredningen åka av medan det finns risk för att varor

vibrerar av om ett plant bord används som lagerinredning såsom det har gjorts i projektet.

8.6 Test av system

För att utvärdera hela systemets funktionalitet genomfördes ett test där bord hämtades, kvitterades och lämnades tio gånger. Genom att göra testet gjordes en bedömning av hur väl de olika delsystemen fungerade tillsammans. Fyra gånger under testet simulerades hinder och AGV:n agerade som önskat vid dessa tillfällen. Alla de tio ordrarna genomfördes, vilket innebär att AGV:n aldrig tappade rutten helt. Genomsnittstiden det tog för AGV:n att genomföra en arbetscykel, utan några hinder, var 104 sekunder.

Kapitel 9

Diskussion

Överlag är vi i gruppen nöjda med vad vi kommit fram till och de huvudsakliga målen som sattes upp har uppnåtts på ett tillfredställande sätt. Något som hade underlättat arbetet och sparat tid under projektets gång är om fler av gruppens medlemmar haft större förkunskaper inom programmering. Detta då stor vikt av programmering låg i det överordnade systemet och myRIO:n.

Vad gäller handledning för LabVIEW-programmeringen finns en mycket hjälpsam studentkontakt från National Instruments som går att rådfråga. I efterhand kan det konstateras att det hade varit fördelaktigt att ta kontakt med denna person långt tidigare än vad som gjordes i projektet vilket också därför ges som tips till kommande kandidatarbeten inom området.

9.1 Problemområden

Arbetet under projektets gång har varit en iterativ process med kontinuerlig återkoppling och då problem har uppstått har olika lösningar framarbetas och utvärderats. AGV:ns förmåga att effektivt navigera mellan QR-koderna är starkt beroende på hur väl kameran är centrerad över AGV:n, hur noggrant QR-koderna är utplacerade samt avståndet mellan koderna. Ett kontinuerligt optimeringsarbete med toleranserna för de olika delsystemens processer har krävts under arbetets gång för att uppnå tillfredställande funktionalitet på systemet. Om kameran fästs med ett vinkelfel eller en förskjutning från centrum så innebär detta att AGV:n kommer felställas med samma fel eftersom AGV:n korrigeras mot QR-koden utefter kamerans centrumpunkt. Om då inte QR-koderna är fästa parallellt mot varandra, med samma avstånd och vinkel, kan felet ökas ytterligare. Avståndet mellan koderna

påverkar därefter hur stort utslaget av korrigeringsfelen hinner bli innan AGV:n hittar en ny kod att reglera sin position mot. Det som avgör hur noggrant allting måste vara är storleken på kamerans synfält eftersom koden måste befinna sig i detta fält för att möjliggöra avläsning. Av denna anledning bör kamerans synfält analyseras vidare om projektet skall vidareutvecklas.

Eftersom projektet till stor del baserats på tillgänglig teknik från tidigare års arbeten är den slutliga konceptlösningen sannolikt inte optimal. Det finns exempelvis inget som styrker att Nexusplattformen skulle vara optimal för detta ändamål. Även för komponenter som myRIO:n och webbkameran kan det finnas lämpligare substitut.

Grundtanken vad gäller avläsningen av QR-koder var att det skulle ske kontinuerligt utan att stanna över koderna. Vidare skulle AGV:n också reglera över koden under rörelse. Detta var för att skapa ett mer flytande system med få stopp. Vid tester visade det sig att bilderna blev för dåliga för att systemet ska kunna få ut information från koderna. Därför fick grundtanken omvärderas och AGV:n tilldelas nu ett avstånd som den kör fritt innan den stannar och läser av QR-koden.

AGV:ns rörelsemönster är inte helt perfekt utan där förekommer variationer. Om AGV:n belastas tungt så finns risker för sämre rörelseprecision. Den extra belastningen samt osäkerheten vad gäller viktfördelningen över AGV:ns hjulpar gör att greppet varierar vilket kan medföra en oprecis gång.

Även om effektiviteten hos systemet kan förbättras i och med en minskad justeringsfrekvens så kan en slutsats kring dess driftssäkerhet dras. Trots att rutnätet inte är perfekt och kameraoptimering återstår så löser systemet uppgiften via sina justeringar och söksekvenser. Viktigt att lyfta fram är även betydelsen av flera interagerande AGV:er. Ett test med endast en AGV kommer lätt att framstå som ineffektivt då väntetiden för operatören upplevs som lång. Syftet är att AGV:erna i stort sett skall avlösa varandra vid plockstationen vilket leder till en minskad väntetid.

Ett bekymmer som inträffade med LiDAR:n var att när systemet startades så tog en Linuxmodul över kontrollen över LiDAR:n vilket gjorde att den inte kunde användas. Förgående års projekt löste det problemet genom att manuellt gå in och ta bort den modulen efter varje uppstart, något som var tidskrävande. Istället för att göra på detta sätt så har det nu ändrats i uppstartsfiler för att se till att modulen inte tar över LiDAR:n.

Ett annat kvarvarande problem är att det tar lång tid innan informationen om ett hinder har mottagits och bearbetats. Detta påverkar sträckan som AGV:n färdas innan den stannar och på grund av tidsbrist är detta något som det inte framar-

betats en lösning till. Efter samarbete med kontaktpersonen på National instruments framgick det att LiDAR:ns drivrutiner till LabVIEW är ineffektiva. Denna begränsning utgör systemets primära hastighetsbegränsning. För att AGV:n skall kunna stanna för hinder utan att riskera att köra in i dem kan AGV:n maximalt transporteras i 200 mm/s.

Vad gäller implementeringen av LiDAR:n har vissa avgränsningar påtvingats på grund av hårdvarubegränsningar. LiDAR:n har ett sökfält på 240° vilket gjort att systemet programmerats så att AGV:n aldrig, med två undantag, kör bakåt utan att den först roterar innan den kör tillbaka. Undantagen finns när felsökningssekvensen sker och när LiDAR:n detekterat ett hinder som inte flyttar sig. Det senare fallet gäller när AGV:n påträffat ett fast hinder och då ska backa tillbaka till föregående QR-kod. Backningen sker då i blindo och risken för kollision finns.

Rotationerna som systemet nu utför är en klar felkälla då det vinkelfel som uppkommer. Detta vinkelfel tillsammans med de dubbla avstånden som AGV:n ibland färdas leder till att QR-koden missas och felsökningssekvensen går igång. Eftersom rotationen är en relativt tidskrävande rörelse kan cykeltiden förkortas om rotationen undviks.

Ett problem för det överordnade systemet är att det inte får någon återkoppling mellan QR-koderna. AGV:n tillåts också gå en eller två koder innan den stannar och reglerar vilket medför att AGV:n går fritt över QR-koderna utan att det överordnade systemet vet exakt var den befinner sig. Vad som kan hända då är att det överordnade systemets inbyggda funktion som skall se till att två AGV:er inte krockar med varandra kort slås ut. Om två AGV:er går över två koder finns alltså risken att det överordnade systemet inte kan undvika att olika AGV:ers rutter korsars. Det överordnade systemet har heller ingen kontroll i de fall som AGV:n börja söka efter QR-koder och får först återkoppling då AGV:n antingen hittar QR-koden eller när felsökningssekvensen är klar. Detta kan leda till att AGV:er interfererar på varandras rutter vilket bör undvikas. Eftersom felsökningssekvensen valts att genomföras på en så liten yta som möjligt i förhållande till ett bra felsökningsresultat så minskar risken att en AGV som söker kommer att söka sig in på andras AGV:ers rutt, se kapitel 7.4.

Två problem som uppkommer med det befintliga lyftanordningskonceptet är dels att anordningen wobblar när den rör sig upp och ner samt att det inte existerar någon återkoppling vad gäller om lyftanordningen är uppe eller nere. Det kan då skapa problem vid uppstart av systemet om lyftanordningen är uppe men systemet tror att den är nere. Ytterligare skulle AGV:n kunna känna av om en hylla finns placerad på anordningen, alternativt att någon form av hyllidentifiering införs.

Vad gäller lagerhyllorna som i nuläget används för att transportera material i

lagermiljön så är det inga som är ämnade för uppgiften. De används helt enkelt för att simulera en lagerhylla. Eftersom de är utformade som helt vanliga, platta bord, finns risken att artiklarna som ligger på bordet glider av. Här finns stor utvecklingspotential, dock bör det material som skall transporteras specificeras närmre. Vid utveckling föreslås att en standardiserad stomme anpassad till AGV:n utvecklas. Denna bör vara kompatibel med anordningar för såväl stora som små artiklar.

9.2 Utvecklingsområden

Från början var tanken att låta AGV:n justera riktning och offset vid varje QR-kod utan att stanna, för att skapa ett flytande och precist system. Efter tester visade det sig att bilderna var för dåliga för att systemet ska få ut information från koderna och istället programmerades AGV:n till att stanna vid varje kod. Att stanna vid varje kod är dock tidskrävande och en optimering som då genomfördes för att snabba på processen var att AGV:n skulle hoppa över varannan kod vid färd (framåt). Gruppen tror att det här fortfarande finns en stor potential för förbättring. Det har diskuterats två möjligheter vilket är att antingen hitta en optimal längd mellan koderna för att AGV:n skall stanna på så få ställen som möjligt, utan att äventyra systemets precision, eller lösa så att AGV:n kan skanna QR-koder och reglera i rörelse.

Om metoden att stanna vid QR-kod utnyttjas kan en förenklad justering användas. Till exempel om AGV:n går 10 mm för långt så skulle, förutsatt att AGV:n skall fortsätta i samma riktning, en subtraktion kunna genomföras på nästkommande sträcka med 10 mm. Då undgås justeringen helt. Alternativt skulle AGV:n kunna kompensera för offseten i sin vinkeljustering, så att den bara behöver genomföra en av tre justeringar. Detta kräver dock information om nästa förflyttning eftersom distansen är variabel.

Konkurrenter som Kiva systems använder sig frekvent av större transportvägar, så kallade motorvägar, i sina system för att uppnå effektiva flöden av AGV:er. Detta har inte behandlats i rapporten, främst på grund av det begränsade rutnät som projektet utvecklats vid. Vid vidareutveckling tros detta vara intressant område för optimering.

För att effektivisera navigeringen finns det flera utvecklingsmöjligheter. I nuläget går AGV:n enbart i de fyra kardinalstrecken north, south, west och east. För att öka snabbheten i systemet kan dessa väderstreck utökas till att involvera även interkardinalstrecken northeast, northwest, southeast och southwest. Detta skulle

betyda att AGV:n tillåts röra sig diagonalt över rutnätet. Dock måste hänsyns tas till begränsningen av LiDAR:ns synfält i de riktningarna. En kombination av flera LiDAR-komponenter eller implementering av ett nytt kollisionsundvikningssystem är möjliga lösningar på problemet.

Till problemet som rör AGV:ns kinematik när den är belastad med en tyngre vikt så skulle en möjlighet vara att, i LabVIEW, sätta ett villkor som gör att den till exempel kör på ett annat sätt när den är belastad. Här krävs optimeringsarbete och tester med AGV:n belastad för att se hur den uppför sig.

En utvärdering kring placering av kollisionsundvikningskomponenterna för att minimera dödzonerna är något som vi tror skulle vara av stor vikt vid en eventuell vidareutveckling. För att erhålla kollisionsundvikning i alla riktningar och därmed ha möjlighet att utnyttja AGV:ns alla rörelseriktningar så krävs det antingen en laser som scannar 360° eller en implementering av två LiDAR där de delar lika med 180° var. Alternativt skulle AGV:ns ultraljudsensorer kunna användas för att erhålla kollisionsundvikning i alla riktningar, men då måste problemet med antalet anslutningsportar på Arduino:n lösas. Vinster i minskning av tid, tack vare att rotation blir överflödig, samt möjligheten att röra sig i alla riktningar är argument nog för att rättfärdiga vidareutveckling. Angående begränsningen i systemet med att informationen från LiDAR:n och bearbetningen i LabVIEW är för långsam så kan det här behöva tittas vidare på andra lösningar.

För att se till att det överordnade systemet får mer frekvent återkoppling vad gäller AGV:ns position finns möjligheten att komplettera QR-navigeringen med någon trilaterationsmetod som till exempel WiFi. QR-koderna skulle då enbart fungera som kalibreringspunkter som kompletterar WiFi-navigeringen. Att integrera detta i en QR-navigering med bättre flöde som beskrivs ovan skulle göra systemet väldigt flexibelt och det överordnade systemet skulle hela tiden kunna hålla koll på AGV:erna för att undvika att de kolliderar med varandra.

Vad gäller lyftanordningen fungerar den som planerat men om användningsområdet kräver att AGV:n ska lyfta tyngre saker kan det vara värt att utvärdera alternativa metoder att driva lyftanordningen på. Det finns kraftfullare stegmotorer att införskaffa än vad som används i projektet men vid tyngre vikter kan det vara mer lämpligt att utnyttja någon alternativ metod. Exempel på det kan vara någon sorts saxlyftteknik eller få inspiration av en domkraft då det är två metoder som är väldigt kraftfulla. Möjlighet finns också att utnyttja kugghjul tillsammans med nuvarande stegmotor för att växla ner rotationshastigheten men höja momentet som motorn levererar. Då erhålls en långsammare lyftning, men den blir desto kraftfullare.

Att integrera systemet med en standardiserad lagerhylla som används ute i in-

dustrin vore ett viktigt steg mot en fullskalig implementering av systemet kommersiellt. Här behövs en utredning om vilka lagerhyllor som används och vilka lagerhyllor som skulle kunna användas i samband med vårt system.

För att lösa problemet med hinderavläsning i det överordnade systemet skulle en funktion kunna utvecklas som tillåter operatören, via gränssnittet, att ta bort eventuella hinder. Exempelvis en knapp som tillåter operatören skriva in positionen och därefter ta bort hindret. Genom att införa detta skulle systemet bli ännu smidigare och kravet på omstart skulle elimineras.

Interaktionen mellan människor och AGV vid plockstationen, det vill säga när AGV:n hämtat en artikel och den ska lastas av, har inte hunnit analyseras och där anser vi att det finns en hel del att titta närmare på. Förutom framtagandet av en mer lämplig lagerinredning att lagerhålla artiklarna på så bör en plockstation utformas. Då bör fokus läggas på att möjliggöra en enkel och ergonomisk avlastning från AGV:n. AGV:n är relativt låg och därför bör någon typ av upphöjningsmekanism till AGV:n skapas vid plockstationen som tillåter operatören att arbeta i en bekväm höjd. Vidare kan ett fokus på kognitiv automation vara en god idé, exempelvis en lampa eller laserpekare som indikerar på vilken som är den korrekta artikeln att plocka. Genom att implementera detta kan arbetet underlättas och eventuella felplock minimeras.

Referenser

[1] *4WD Mecanum Wheel Mobile Robot Kit*. 2010. URL: http://www.nexusrobot.com/product.php?id_product=67 (hämtad 2015-03-03).

[2] *Amazon's new Kiva Robots use QR Codes to sense their Location*. 5 april 2012. URL: http://www.scandit.com/2012/04/05/amazon%C3%A2%C2%80%C2%99s-new-kiva-robots-use-qr-codes-to-sense-their-location (hämtad 2015-04-14).

[3] *Amazon's new Kiva Robots use QR codes to sense their location*. 5 april 2012. URL: http://www.scandit.com/2012/04/05/amazon%C3%A2%C2%80%C2%99s-new-kiva-robots-use-qr-codes-to-sense-their-location/ (hämtad 2015-02-09).

[4] *Arduino Ultrasonic Distance Sensor*. 2010. URL: http://www.nexusrobot.com/product.php?id_product=44 (hämtad 2015-04-14).

[5] H. Bing m. fl. *A Route Planning Method Based on Improved Artificial Potential Field Algorithm*. Rapport. Xi'an Hongqing Research Institute of Hi-Tech, 2011. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6014330.

[6] Sakmongkon Chumkamon, Peranitti Tuvaphanthaphiphat och Phongsak Keeratiwintakorn. *A Blind Navigation System Using RFID for Indoor Environments*. Rapport. IEEE, 2008. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4600543&tag=1.

[7] *Clojure Programming Language*. 2015. URL: http://clojure.org/ (hämtad 2015-05-11).

[8] *ClojureScript*. 2015. URL: http://clojure.org/clojurescript (hämtad 2015-05-11).

[9] Dave Coleman. *Lee's O(n2 log n) Visibility Graph Algorithm Implementation and Analysis*. Rapport. Department of Computer Science, University of Colorado at Boulder, 2 maj 2012. URL: http://dav.ee/papers/Visibility_Graph_Algorithm.pdf.

[10] *Det sker många arbetsolyckor med truckar*. 2014. URL: http://www.av.se/teman/truckar/?AspxAutoDetectCookieSupport=1 (hämtad 2015-03-27).

[11] Ioan Doroftei, Victor Grosu och Veaceslav Spinu. *Omnidirectional Mobile Robot – Design and Implementation*. Rapport. Bioinspiration och Robotics, 1 sept. 2007. URL: http://www.intechopen.com/books/bioinspiration_and_robotics_walking_and_climbing_robots/omnidirectional_mobile_robot_-__design_and_implementation.

[12] Anton Ekman m. fl. *Warehouse Solution using Automated Guided Vehicles*. Kandidatarbete. Signaler och system Chalmers tekniska högskola, 2014.

[13] *Extensible Data Notation*. 2014. URL: https://github.com/edn-format/edn (hämtad 2015-05-13).

[14] The Boston Consulting Group. *Takeoff in Robotics Will Power the Next Productivity Surge in Manufacturing*. 10 febr. 2015. URL: http://www.bcg.com/media/PressReleaseDetails.aspx?id=tcm:12-181684 (hämtad 2015-02-12).

[15] Yanying Gu, Anthony Lo och Ignas Niemegeers. *A Survey of Indoor Positioning Systems for Wireless Personal Networks*. Rapport. IEEE, 2009. URL: http://www.csd.uoc.gr/~hy439/papers/gu2009pdf.pdf.

[16] Erik Henriksson m. fl. *Materialtransport med AGV*. Kandidatarbete. Signaler och system Chalmers tekniska högskola, 2014.

[17] *Heuristics*. 2012. URL: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html (hämtad 2015-04-09).

[18] *How Much Time Is Required to Read an RFID Tag*. 24 sept. 2013. URL: http://www.rfidjournal.com/blogs/experts/entry?10736 (hämtad 2015-02-26).

[19] Rafia Inam. *A* Algorithm for Multicore Graphics Processors*. Rapport. CHALMERS UNIVERSITY OF TECHNOLOGY, Department of Computer Science och Engineering, 2009. URL: http://publications.lib.chalmers.se/records/fulltext/129175.pdf.

[20] Patrik Jonsson och Stig-Arne Mattson. *Logistik — Läran om effektiva materialflöden*. 2011.

[21] *LACK*. 2015. URL: http://www.ikea.com/se/sv/catalog/products/90302060/#/10279822 (hämtad 2015-05-13).

[22] *Laser Guidance*. 13 aug. 2013. URL: http://www.agve.se/?s=laser (hämtad 2015-02-26).

[23] *Lidar - Light Detection and Ranging*. 2014. URL: http://www.accessscience.com.proxy.lib.chalmers.se/content/lidar/380750 (hämtad 2015-04-14).

[24] *Line Follower Robots - Controlling, Working Principle and Application*. 2013. URL: https://www.elprocus.com/line-follower-robot-basics-controlling/ (hämtad 2015-02-26).

[25] *Logitech Webcam C930*. 2013. URL: http://www.logitech.com/en-us/product/webcam-c930e-business (hämtad 2015-05-13).

[26] Wong Yuen Loong, Liew Zhen Long och Lim Chot Hun. *A STAR PATH FOLLOWING MOBILE ROBOT*. Rapport. Faculty of Engineering och Technologies, Multimedia University Malacca Campus, 2011. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5937169.

[27] *Magnet Navigation*. 2009. URL: https://www.amerden.com/AmerdenWeb/Information/Guiding/magnet.html (hämtad 2015-02-26).

[28] *MyRIO National Instruments - för studenter*. 2015. URL: http://www.ni.com/newsroom/release/students-can-now-design-sophisticated-systems-in-one-semester-with-ni-myrio/sv/ (hämtad 2015-05-12).

[29] *Navigation methods*. 17 juli 2013. URL: http://www.agve.se/?s=navigation (hämtad 2015-04-14).

[30] *NDC8 AGV Control System*. 2013. URL: http://www.kollmorgen.com/en-us/products/vehicle-controls/electrical-vehicle-controls/ndc8/ (hämtad 2015-04-14).

[31] *QR Code Essentials*. 2011. URL: http://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo=&tabid=1426&mid=4802 (hämtad 2015-02-20).

[32] *Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms*. 2011. URL: http://www.ijcte.org/papers/738-T012.pdf (hämtad 2015-03-25).

[33] *Robots Kit Manual*. 2013. URL: https://www.active-robots.com/fileuploader/download/download/?d=0&file=custom/upload/File-1342627200.pdf (hämtad 2015-04-14).

[34] Hani Safadi. *Local Path Planning Using Virtual Potential Field*. Rapport. McGill University, School of Computer Science, 18 april 2007. URL: http://www.cs.mcgill.ca/~hsafad/robotics/index.html.

[35] *Satellite Navigation Systems*. 2014. URL: http://www.accessscience.com.proxy.lib.chalmers.se/content/satellite-navigation-systems/602800 (hämtad 2015-02-24).

[36] *Scanning Laser Range Finder URG-04LX-UG01 Specifications*. 2009. URL: https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html (hämtad 2015-04-14).

[37] *Skruvdomkrafter*. 2013. URL: http://www.mekanex.se/produkter/linjar/se-skruvdomkrafter.shtml (hämtad 2015-04-16).

[38] *Sonar - Sound Navigation and Ranging*. 2014. URL: http://www.accessscience.com.proxy.lib.chalmers.se/content/sonar/636600 (hämtad 2015-04-14).

[39] Evan A. Sultanik, Ali Shokoufandeh och William C. Regli. *Dominating Sets of Agents in Visibility Graphs: Distributed Algorithms for Art Gallery Problems*. Rapport. Drexel University, 2011. URL: http://www.ifaamas.org/Proceedings/aamas2010/pdf/01%20Full%20Papers/16_06_FP_0444.pdf.

[40] *The art gallery problem*. 14 juni 2014. URL: https://plus.maths.org/content/art-gallery-probleml (hämtad 2015-04-09).

[41] *The WebSockets API*. 2012. URL: http://www.w3.org/TR/websockets/ (hämtad 2015-05-13).

[42] *User Guide and Specifications - MyRIO 1900*. 2013. URL: http://www.ni.com/pdf/manuals/376047a.pdf (hämtad 2015-04-14).

[43] Dmitry S. Yershov och Steven M. LaValle. *Simplicial Dijkstra and A* Algorithms: From Graphs to Continuous Spaces*. Rapport. Department of Computer Science, University of Illinois at Urbana-Champaign, 20 aug. 2012. URL: http://www.tandfonline.com.proxy.lib.chalmers.se/doi/pdf/10.1080/01691864.2012.729559.

Appendix A

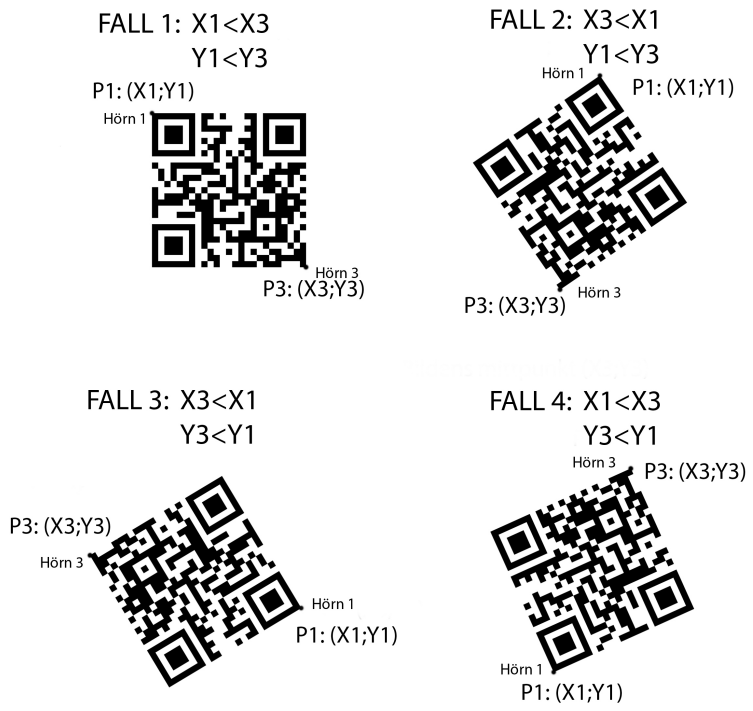
Kravspecifikation

Tabell A.1: Samlad kravspecifikation för projektet

Nr	Typ av kravområde	Krav	Önskemål	Uppnått
1	**Kollisionssystem**			
1.1	Vikt på ingående komponenter	<5 kg	<1kg	Önskemål
1.2	Skall ej krocka	Skall stanna vid kollisionsrisk	Skall flytta undan från objekt i rörelse	Krav
1.3	Hårdvarukostnad	<30000 kr	<15000 kr	Önskemål
1.4	Storlek	Skall rymmas på AGV		Krav
1.5	Avskanning av omgivning	AGV-bredd i färdriktningen	360°	Krav
1.6	Skanningsavstånd	Längre än stoppsträckan		Krav
1.7	Skanningsfrekvens	5 ggr/sekund	10 ggr/sekund	Önskemål
2	**Anordning för transport av material**			
2.1	Förflyttning ska ske stabilt	Lagerinredning skall ej tappas	Lagerinredningen ska ej tappas under färd	Krav
2.2	Hårdvarukostnad	<3000 kr	<1500 kr	Önskemål
2.3	Tid att lasta varor för transport	<10 sekunder	<5 sekunder	Krav
2.4	Storlek	Skall rymmas på AGV		Krav
2.5	Möjlig lastvikt	>4 kg	>10 kg	Krav
2.6	Vikt på ingående komponenter	<8 kg	<4 kg	Önskemål
3	**Navigationssystem**			
3.1	Lokaliseringsprecision	<50 mm felmarginal	<10 mm felmarginal	Önskemål
3.2	Hårdvarukostnad	<30000 kr	<15000 kr	Önskemål
3.3	Storlek	Skall rymmas på AGV		Krav
3.4	Vikt	<10 kg	<4 kg	Krav
3.5	Möjlig hastighet på robot	>0,05 m/s	>0,5 m/s	
4	**AGV**			
4.1	Bredd	<450 mm		Krav
4.2	Höjd	<450 mm		Krav
4.3	Längd	<450 mm		Krav
4.4	Vikt olastad	<20 kg	<10 kg	Krav
5	**Överordnat system**			
5.1	Stöd för hantering av flera AGV:er	>1		Krav
5.2	Notifiera användare vid problem	Notiser ges vid oväntade händelser		Krav
5.3	Karta i gränssnitt över systemet		Karta som uppdateras i realtid	Önskemål

Appendix B

Offset och vinkelberäkningar



Figur B.1: Visar de fyra olika fallen för offsetberäkning

Det finns fyra olika fall för offsetberäkningarna. Här nedan redovisar vi beräkningen för fall 4 och hänvisar till den vad gäller de andra fallen. Beräkningsgången är samma för dem.

För att ta fram mittpunkten för fall 4:

$$QRmittpunkt_{x-led} = \frac{X3 - X1}{2} + X1 \tag{B.1}$$

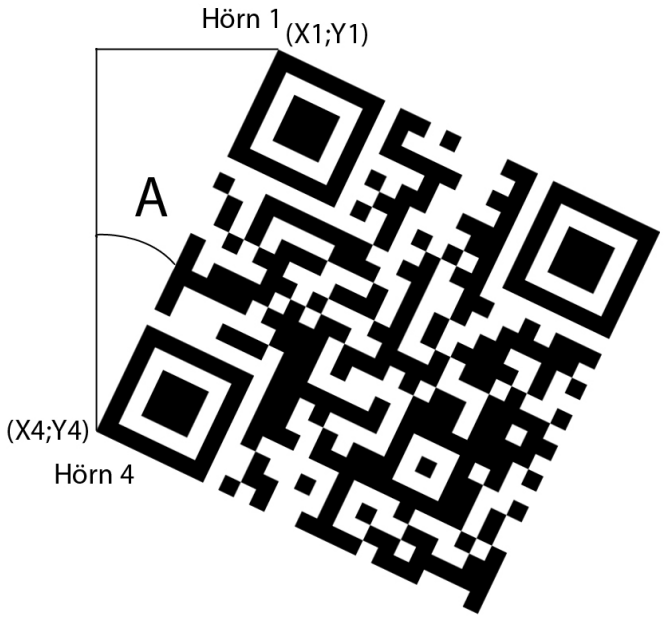
$$QRmittpunkt_{y-led} = \frac{Y1 - Y3}{2} + Y3 \tag{B.2}$$

För att ta fram offseten för fall 4:

$$Offset_{xled} = QRmittpunktxled - Kameransmittpunkt_{xled} \tag{B.3}$$

$$Offset_{yled} = QRmittpunkt_{yled} - Kameransmittpunkt_{yled} \tag{B.4}$$

För vinkeloffset: Ekvation B.5 och B.6 visar hur vinkelfelet beräknas.



Figur B.2: Visar hur vinkelfelet tas ut

$$tanA = \frac{\Delta y}{\Delta x} \tag{B.5}$$

$$A = arctan\left(\frac{\Delta y}{\Delta x}\right) \tag{B.6}$$

Appendix C

Beräkningar för LiDAR

För att möjliggöra att LiDAR:n skannar av AGV:ns dimensioner krävs en teoretisk beräkning för att veta vilken vinkel LiDAR:n skall skanna av. I figur C.1 visas en grafisk framställning hur vinklarna kan beräknas. LiDAR:n har ett skanningspann på 240°vilket betyder att det villkoret som måste uppfylla är:

$$2(C + 90^\circ) < 240^\circ \tag{C.1}$$

$$C < 120^\circ - 90^\circ \tag{C.2}$$

$$C < 30^\circ \tag{C.3}$$

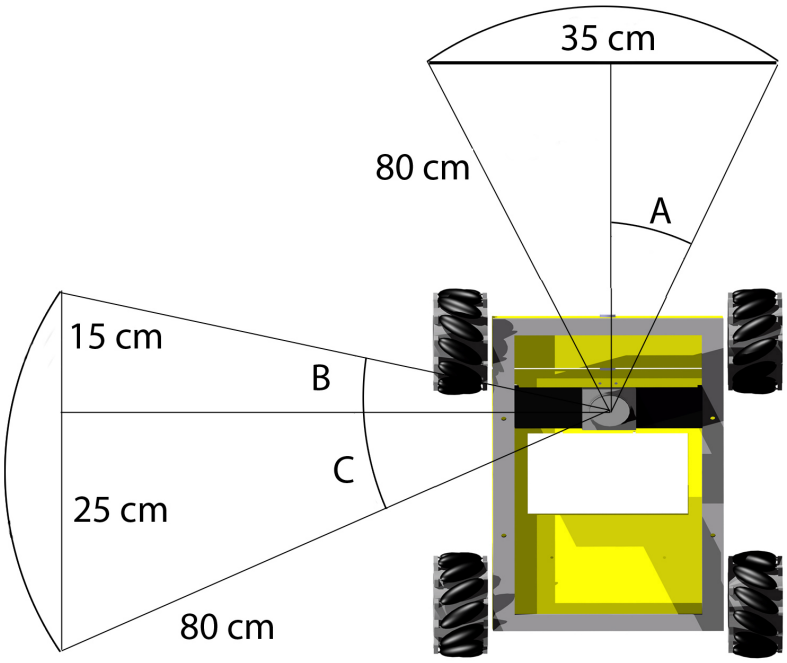
Från figuren C.1 nedan fås:

$$sinC = \frac{25}{80} \tag{C.4}$$

$$C = arcsin\left(\frac{25}{80}\right) \tag{C.5}$$

$$C = 18.21^\circ \tag{C.6}$$

Här syns det att C klarar av villkoret att det skall vara mindre än 30°för att skanningen skall rymmas inom de 240°som LiDAR:n erbjuder. Nedan följer beräkningarna av de andra vinklarna.



Figur C.1: Visar LiDAR:ns scanningsvidd vid körning

$$sinB = \frac{15}{80} \tag{C.7}$$

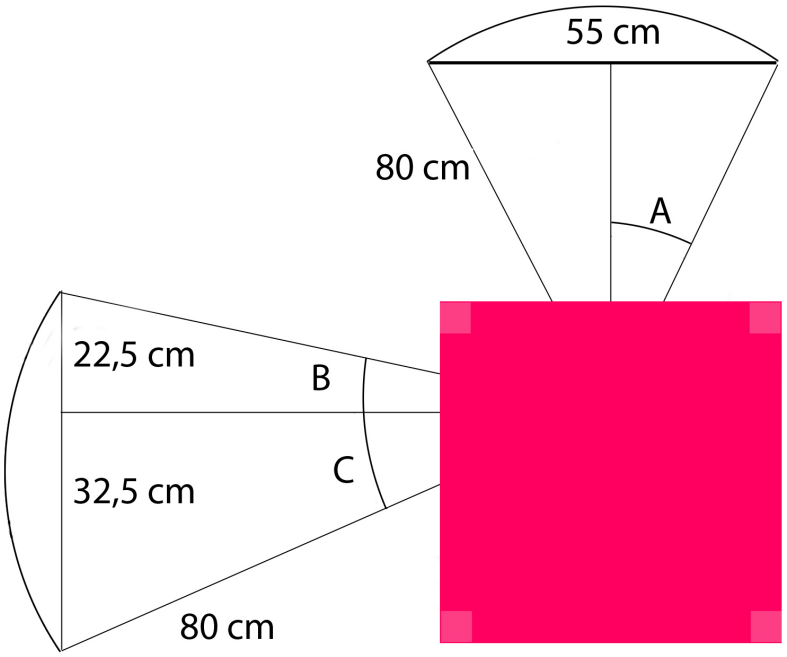
$$B = arcsin\left(\frac{15}{80}\right) \tag{C.8}$$

$$C = 18.81° \tag{C.9}$$

$$sinA = \frac{35}{160} \tag{C.10}$$

$$A = arcsin\left(\frac{35}{160}\right) \tag{C.11}$$

$$A = 12.64° \tag{C.12}$$



Figur C.2: Visar LiDAR:ns scanningsvidd vid körning med belastning

I fallet när AGV:n är belastad med bordet blir beräkningarna desamma som för fallet ovan men med skillnaden att breddvariablen ändras, se figur C.2. Breddvariablen ändras för att bordets dimensioner är nu det som sätter kraven på storleken hos skanningsområdet.

Vinklarna blir då på samma sätt som ovan:

$$A = 20.11^\circ \tag{C.13}$$

$$B = 16.33^\circ \tag{C.14}$$

$$A = 23.97^\circ \tag{C.15}$$

Vi ser att även i detta fallet så klarar C av villkoret:

$$C < 30^\circ \tag{C.16}$$

Vilket betyder att LiDAR:nS skanningsbredd på 240°räcker.

En sammanställning av vinklarna återfinns neda i tabell C.1

Tabell C.1: Värdena för LiDAR:ns skanningsspann lastad och olastad

Vinkel	**Lastad**	**Olastad**
A	13°	20°
B	11°	16°
C	18°	24°

Appendix D

Beräkning för lyftanordning

Vid beräkningarna av vilket moment som krävdes från stegmotorn för att lyfta lasten användes följande ekvationer (Ekvation D.1 och D.2):

$$T_{raise} = \frac{Fd_m}{2}\left(\frac{l + \pi\mu d_m}{\pi d_m - \mu l}\right) \tag{D.1}$$

$$T_{raise} = \frac{Fd_m}{2}\left(\frac{\pi\mu d_m - l}{\pi d_m + \mu l}\right) \tag{D.2}$$

De ingående variablerna i ekvation D.1 och D.2 är:

- T: moment som krävs för jämnvikt
- F: kraften som ligger på flänsmuttern
- d_m: medeldiametern på trapetsskruven
- l: stighöjden för ett varv på trapetsskruven
- μ: friktionskoefficienten mellan flänsmuttern och trapetsskruven

Momentet som krävs var störst vid upplyftning av lyftstycket vilket innebar att det var ekvation D.2 som utvärderades, då det var detta fall som krävde mest av motorn och därmed var dimensionerande.

Tester av den befintliga stegmotorn, med ett hållkraft på 0,098 Nm, gjordes och det indikerade på att den skulle klara av en last på cirka ett kilogram utöver lyftstycket, vilket efter vägning motsvarade en totalvikt på 1,73 kg. Genom att utnyttja denna information kunde en ungefärlig friktionskoefficient erhållas, där följande värden på de ingående variablerna utnyttjades:

$$T = 0,098Nm \tag{D.3}$$

$$F = 1,73 * 9,81 = 16,97N \tag{D.4}$$

$$d_m = 0,01575m \tag{D.5}$$

$$l = 0,004m \tag{D.6}$$

Friktionskoefficienten bröts ut och ekvationen blev således:

$$\mu = \left(\frac{2T_{raise}\pi d_m - FdmL}{2T_{raise}L + Fd_m^2\pi} \right) \tag{D.7}$$

Beräkningen gav en friktionskoefficient på cirka $\mu = 0.62$ vilket kunde utnyttjas vid beräkningen av ett nytt moment alternativt en ny last.

Beräkningen av friktionskoefficienten var ungefärlig vilket innebar en hög osäkerhetsfaktor och därför utvärderades flera olika möjliga stegmotorer som skulle kunna utnyttjas. Valet föll på en motor med hållkraft 1.1 Nm och den teoretiska maxlasten beräknades genom att F bröts ut ur ekvation D.1 vilket gav följande beräkning:

$$F = \frac{2T_{raise}}{d_m} \frac{\pi d_m - \mu L}{L + \pi \mu d_m} \tag{D.8}$$

Vilket ger en förväntad kraft på $F2 = 190,5N$.

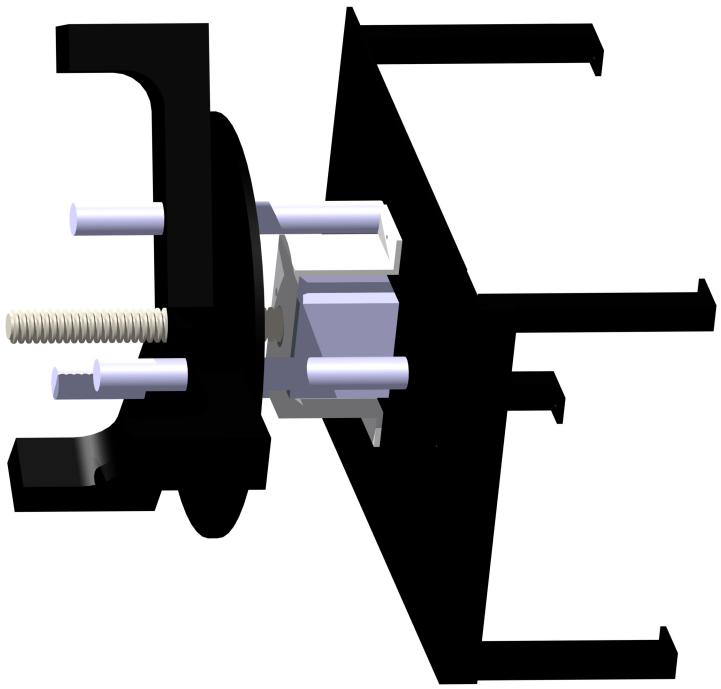
Det innebär att den nya motorn ska teoretiskt sett orka med en last på cirka 19 kg. Detta var mycket högre än den last på cirka 4 kg som det innebar att lyfta bordet men med tanke på den osäkerheten i vissa ingående variabler, främst friktionskoefficienten, ansågs det vettigt att ha en stor marginal.

Appendix E

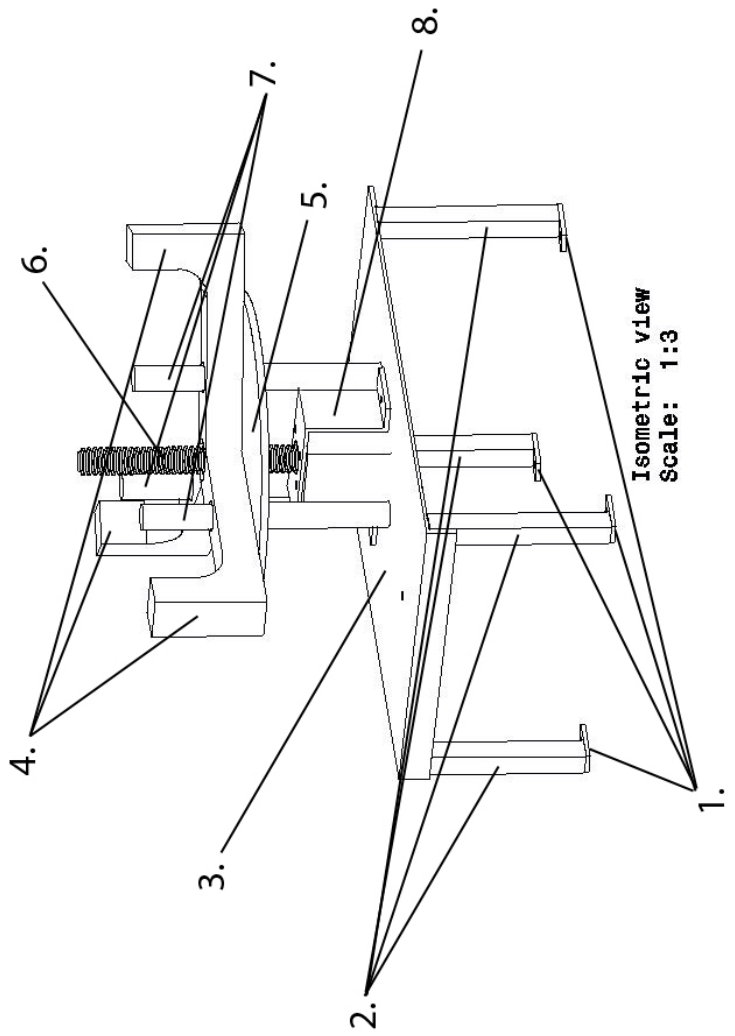
Ritningar och modeller av anordning för transport för material

Här redovisas ritningarna på anordningen för transport av material. Först följer en bild från CATIA hur det slutgiltliga konceptet ser ut, se figur E.1

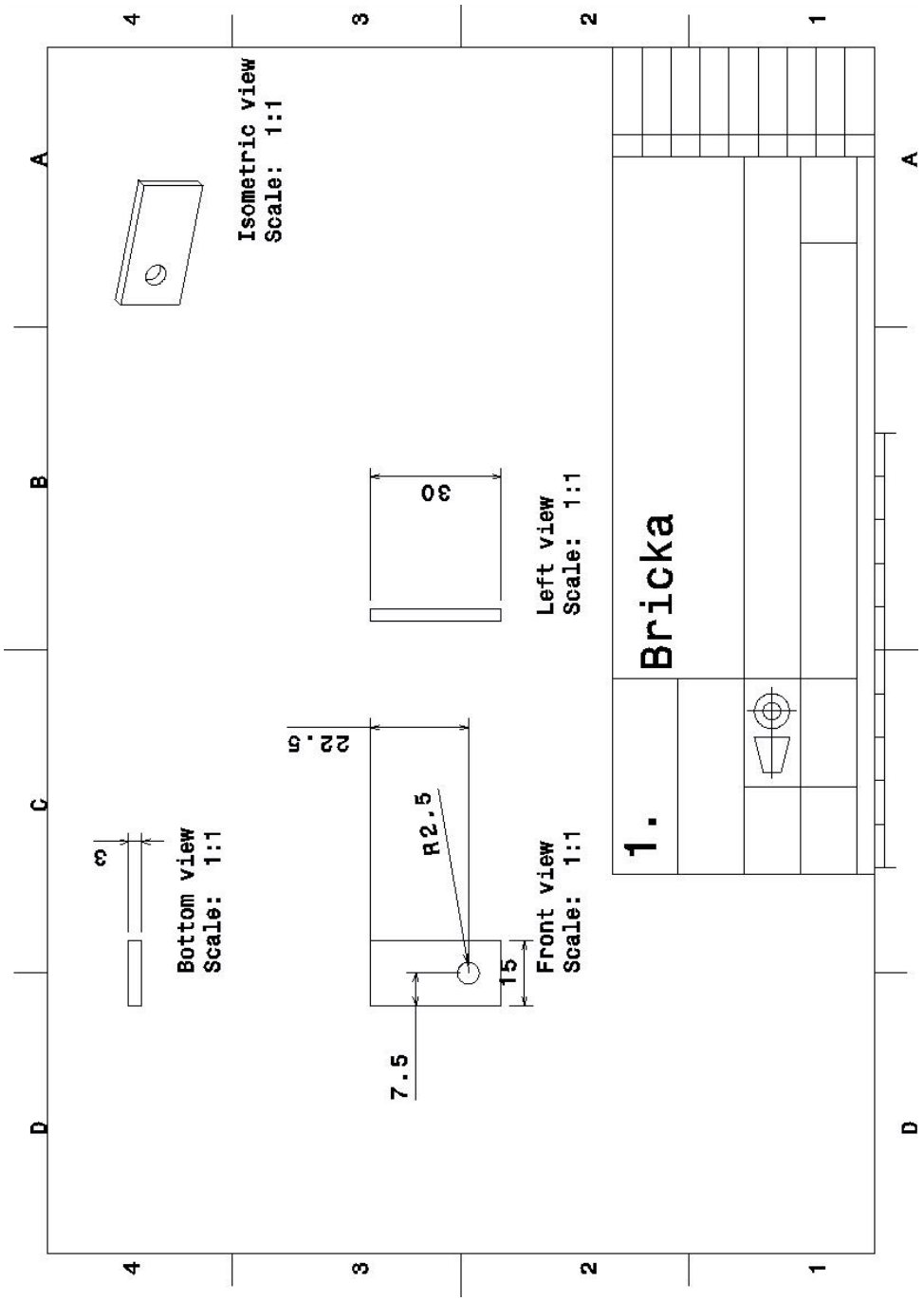
Efter det visas en ballongritning av hela konceptet med numrering av varje ingående del, se figur E.2. Därefter redovisas varje del som hittas i figur E.2.



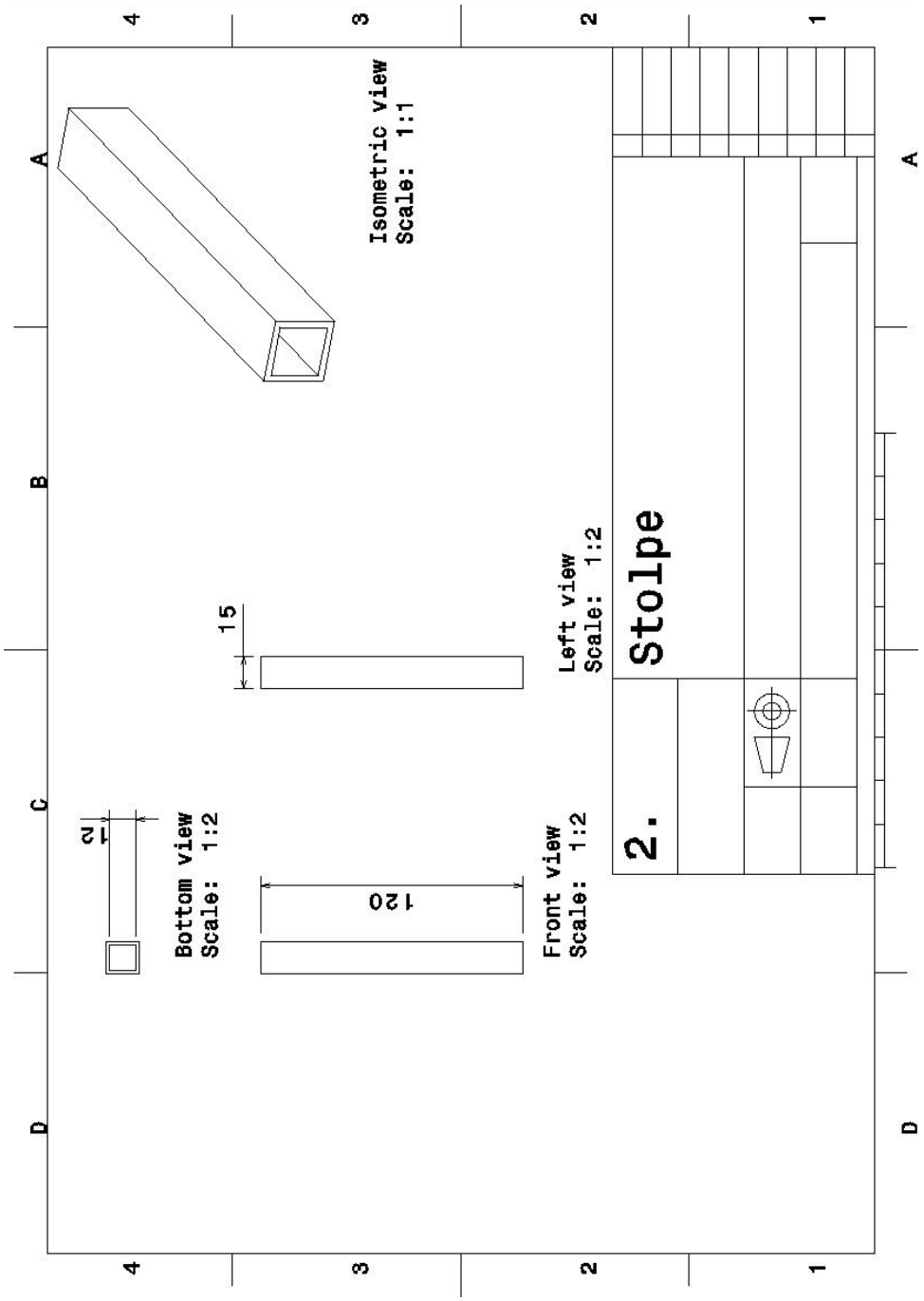
Figur E.1: Det slutgiltliga konceptet



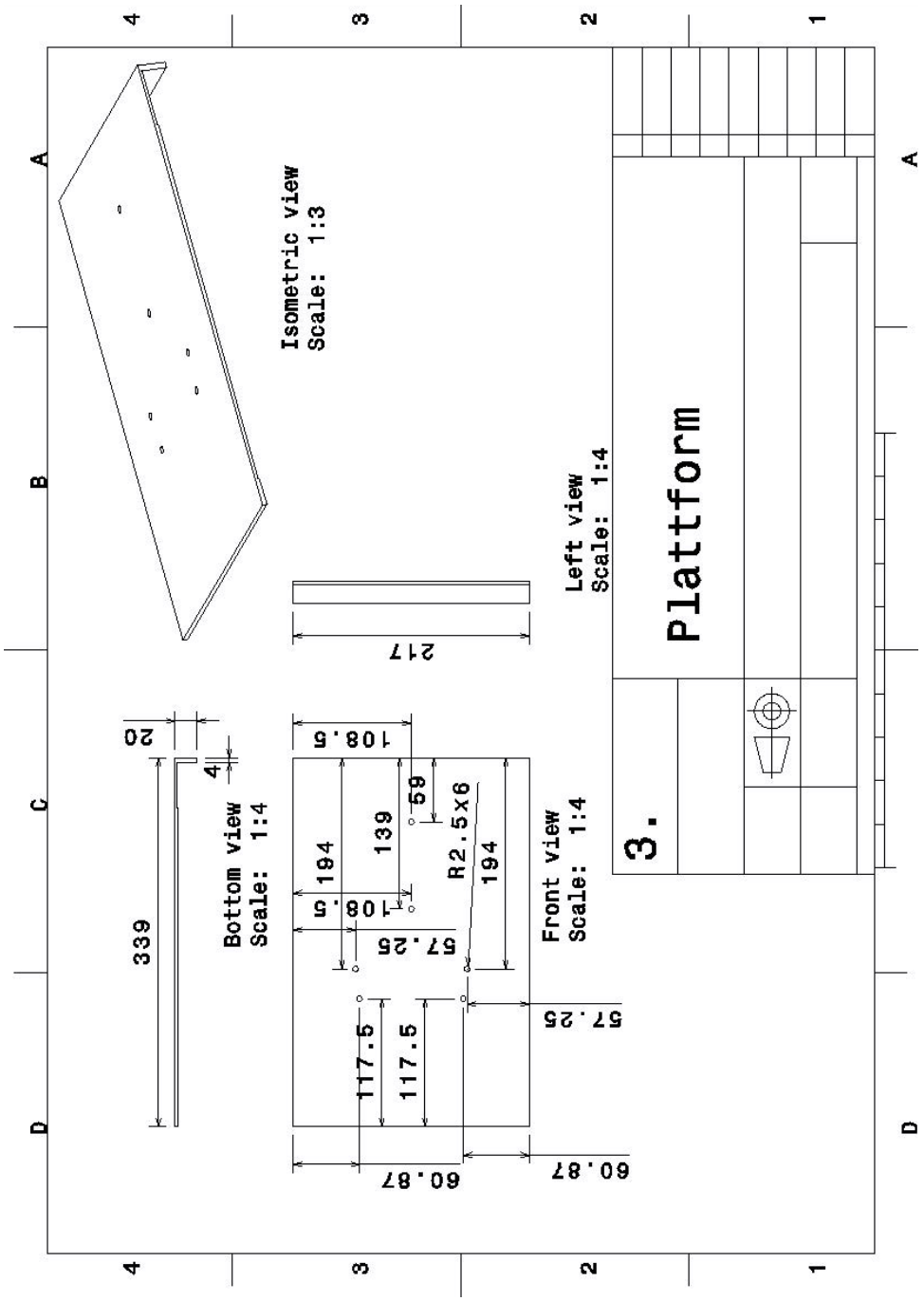
Figur E.2: Ballongskiss på det slutgiltliga konceptet



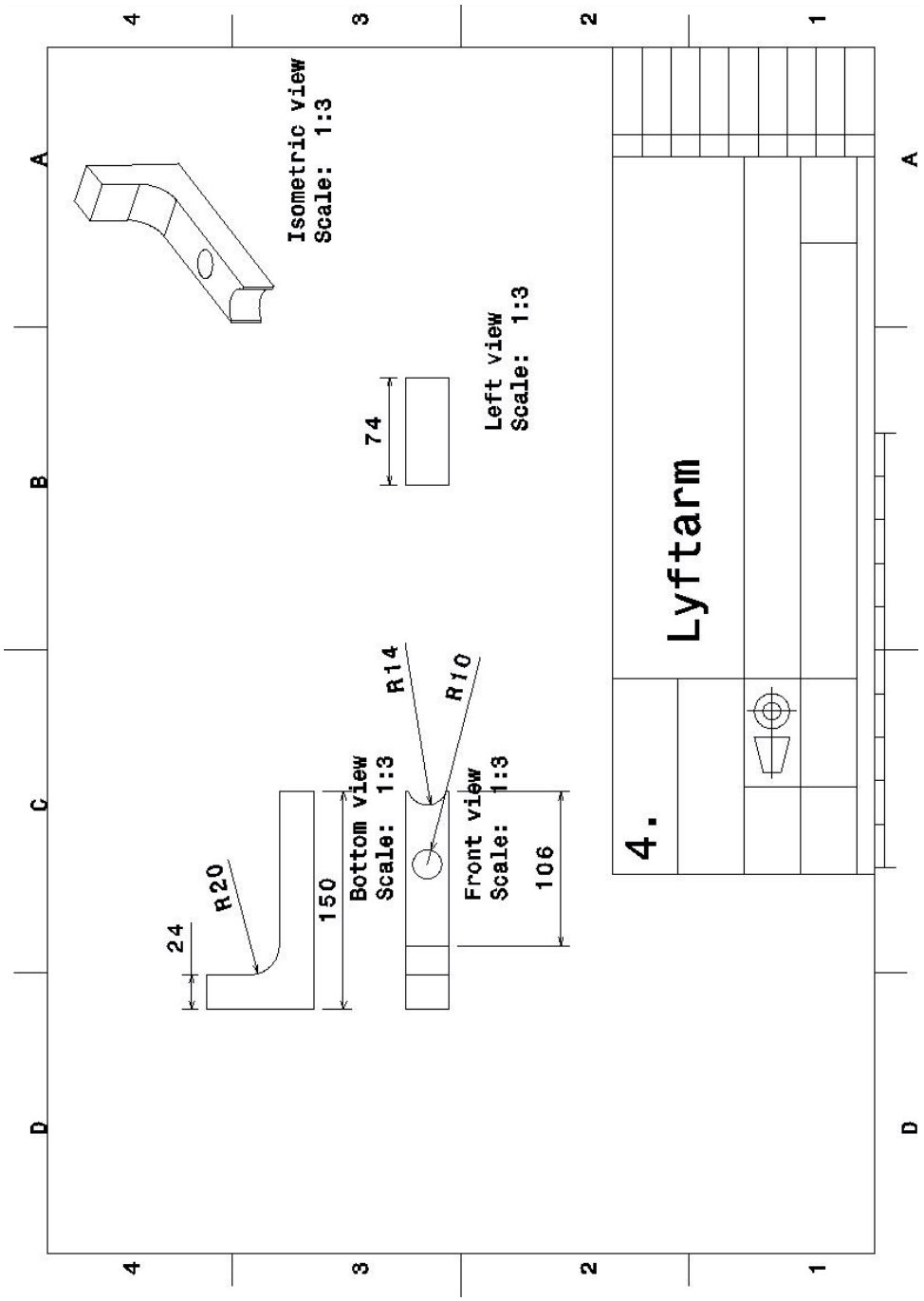
Figur E.3: Ritning på brickan



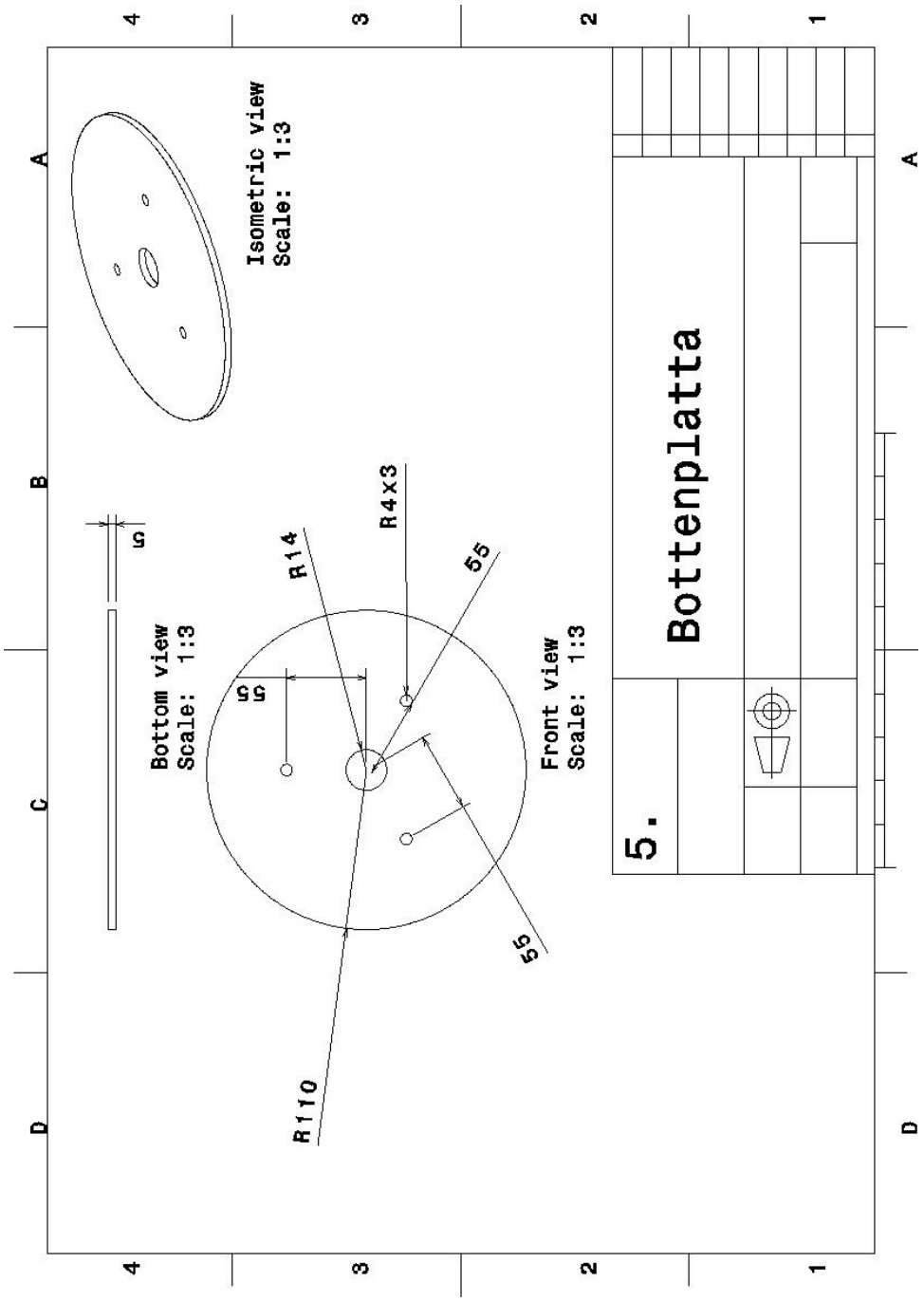
Figur E.4: Ritning på stolpen



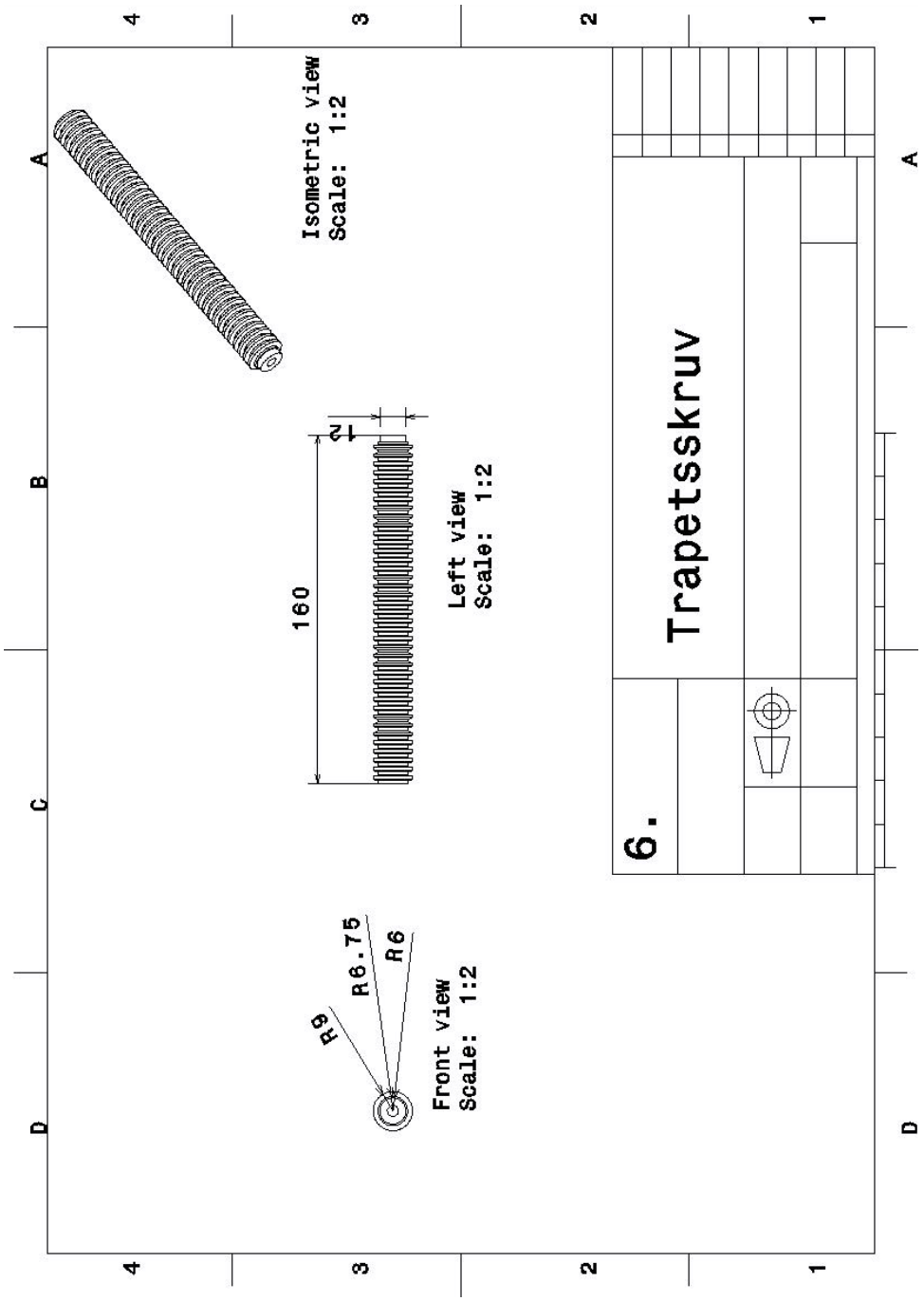
Figur E.5: Ritning på plattform



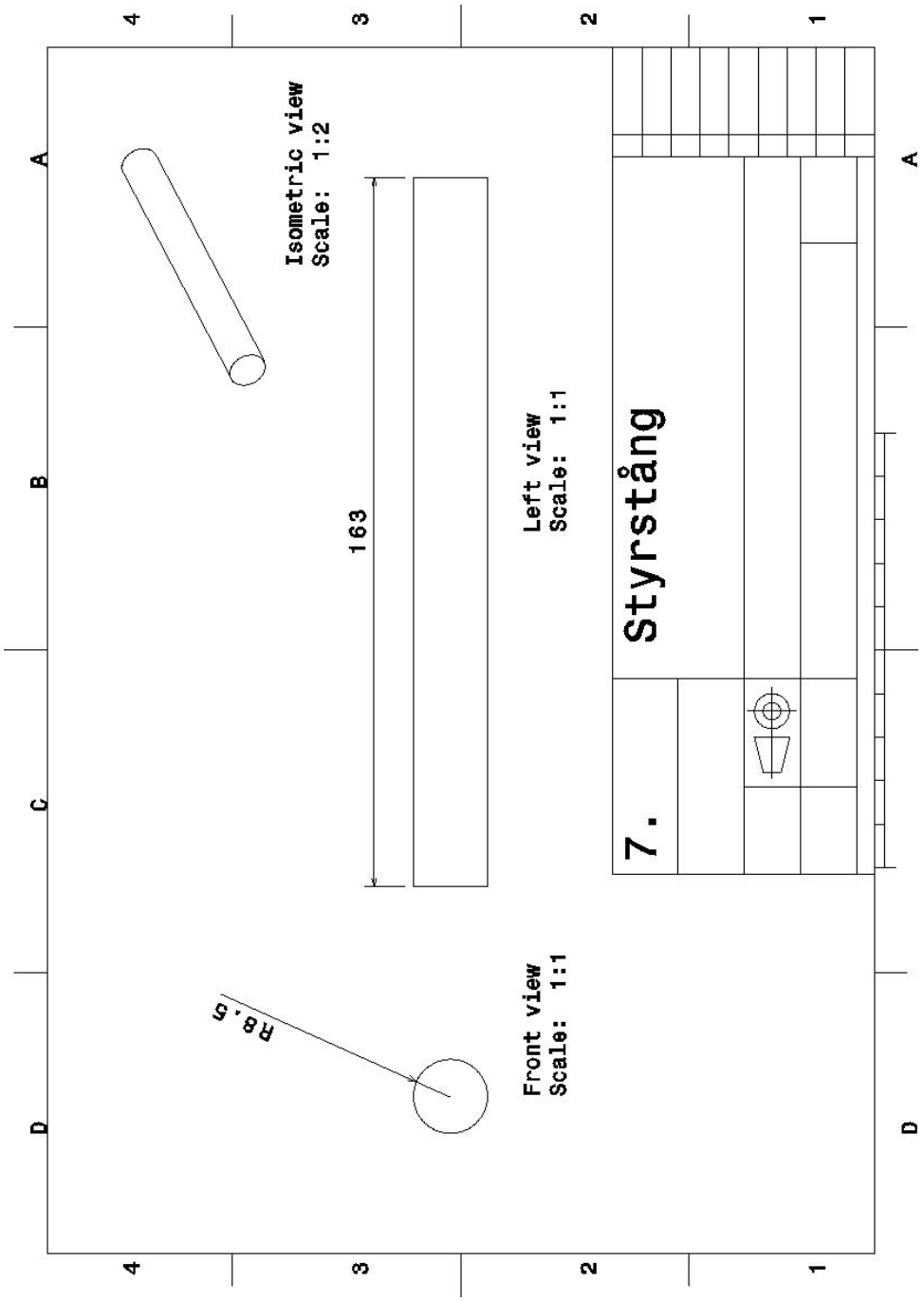
Figur E.6: Ritning på lyftarm



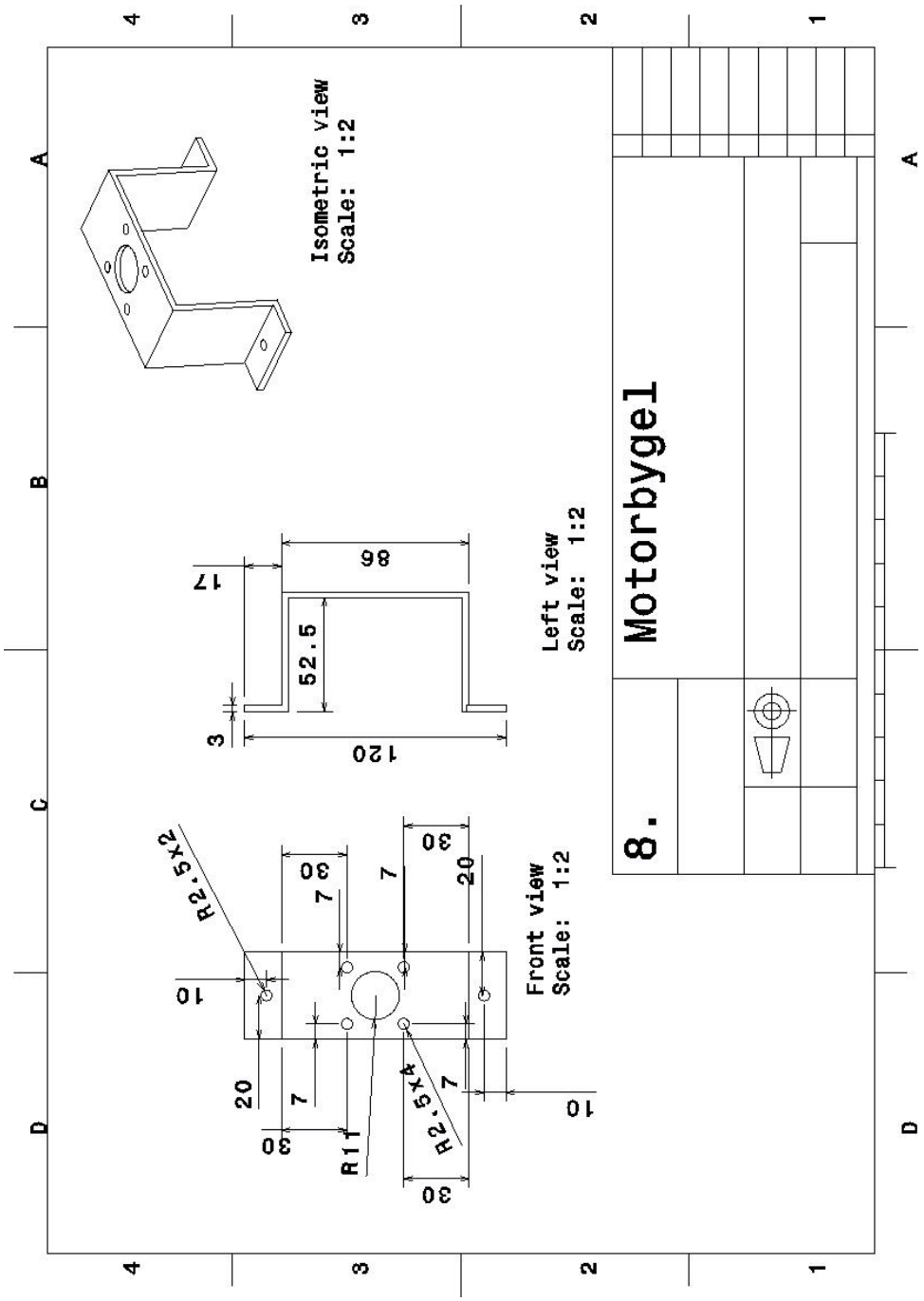
Figur E.7: Ritning på bottenplatta



Figur E.8: Ritning på trapetsskruv



Figur E.9: Ritning på styrstång



Figur E.10: Ritning på motorbygel

Appendix F

Tester

Här redovisas de tester som utförts på systemet.

F.1 Rörelseprecision

Test på hur väl AGV:n genomför en specificerad rörelse, resultatet visar på uppmätt fel.

Rotation ett helt varv.

Testfall nr.	**Fel**
1	$-1°$
2	$0°$
3	$+1°$
4	$+1°$
5	$-1°$

Rotation ett kvarts varv.

Testfall nr.	**Fel**
1	$-1°$
2	$-1°$
3	$-1°$
4	$-2°$
5	$-1°$

Test av minimal rotation (2°).

Testfall nr.	Fel
1	0°
2	0°
3	0°
4	0°
5	0°

Test av hur mycket AGV:n avviker vid 800mm körning frammåt.

Testfall nr.	Fel
1	-26 mm
2	-9 mm
3	21 mm
4	-20 mm
5	-29 mm

F.2 Navigation

AGV:n testkördes i en bana med 20 olika positioner, tabellen visar hur många gånger felsökningsalgoritmen startades för att QR-koden inte gick att läsa.

Testfall nr.	Antal
1	1
2	2
3	1
4	1
5	1

Tid det tar för AGV:n att läsa en QR-kod.

Testfall nr.	Tid
1	0.34 s
2	0.48 s
3	0.21 s
4	0.38 s
5	0.34 s

F.3 Kollisionsundvikning

Tid från att hinder är upptäckt tills att AGV:n står still.

Testfall nr.	Tid
1	1.54 s
2	1.29 s
3	1.68 s
4	1.56 s
5	1.45 s

F.4 Överordnat system

Uppgift att utföra 10 stycken ordrar initierade från gränssnittet. Förutsatt att minst en AGV är tillgänglig så utförs alla ordrar.

F.4.1 Flera AGV:er

Testkörning av systemet med flera AGV:er utfördes virtuellt då endast hårdvara för en AGV finns. Uppgiften var att utföra 10 stycken ordrar, vilket slutfördes.

F.5 Lyftanordning

Tid på sänkning/höjning av hylla.

Testfall nr.	Tid
1	7.56 s
2	7.79 s
3	7.75 s
4	7.67 s
5	7.64 s

F.6 Test av system

Test av tio stycken arbetscykler som innefattar hämtning, kvittering samt återlämning av bordet. Bordet placerades så att AGV:n behövde förflytta sig fyra QR-koder enkel väg. Kollisionssystemet utvärderades även i testet då hinder simulerades fyra gånger, två gånger fasta hinder och två gånger rörliga. AGV:n agerade då som planerat, när det rörliga hindret flyttade sig inom fem sekunder körde AGV:n vidare enligt planerad rutt. När det fasta hindret inte försvann inom

fem sekunder så backade AGV:n tillbaka till föregående QR-kod samt informerade det överordnade systemet som då blockerade den aktuella QR-koden. Vid de två blockeringarna fungerade även gränssnittet som planerat då en notifiering gavs till operatören om att ett hinder upptäckts. AGV:n lyckades genomföra alla tio ordrar och felsökningssekvensen gick in i snitt en gång per arbetscykel. Genom att göra testet utvärderades funktionaliteten på hela systemet och hur väl de olika delarna fungerade tillsammans. I tabellen nedan redovisas antalet gånger felsökningssekvensen gick in vid varje försök, om det var ett hinder på rutten samt tiden för varje arbetscykel.

Testfall nr.	**Antal**	**Hinder**	**Tid**
1	1		104 s
2	2	Rörligt	124 s
3	1		108 s
4	1	Fast	145 s
5	1		100 s
6	0	Rörligt	97 s
7	1		102 s
8	1	Fast	141 s
9	1		107 s
10	1		103 s

Genomsnittstiden för en arbetscykel utan hinder beräknades enligt ekvation F.1.

$$\frac{104 + 108 + 100 + 102 + 107 + 103}{6} = 104 \tag{F.1}$$