CHALMERS

Autonom materialtransport med hög precision

AGV1

Kandidatarbete inom automation, reglerteknik och mekatronik.

Karl Gustavsson 900124
Mikael Ahlstedt 900702
Adam Maaninka 900502
Johan Florbäck 880823
Daniel Svensson 881014
Anders Stenbeck 830108

Institutionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2012
Kandidatarbete SSYX02-12-09

Sammanfattning

Inom tillverkningsindustrin blir effektiva och flexibla materialleveranser allt viktigare. För att möta de utmaningarna är ett alternativ att använda en Automatic Guided Vehicle (AGV). Det är en förarlös truck som kan programmeras för att utföra olika uppgifter. Denna rapport behandlar arbetet med att färdigställa en AGV som ska leverera delar till en automatiserad monteringslina för modellbilar. Monteringslinan kräver hög precision vid materialleveransen och området som AGVn ska navigera genom är trångt. AGVn ska dessutom som sidouppdrag kunna identifiera och navigera till röda läskburkar. För att möta de krav som ställs på AGVn använder den ett Sick NAV200 laserpositioneringssystem som är kopplat till en CompactRIO styrdator. CompactRIOn är också kopplad till ultraljudssensorer för att AGVn ska undvika att kollidera med omgivningen. AGVn har dessutom en Microsoft Kinectkamera kopplad till en laptop för att identifiera läskburkar. Laptoppen används även för att beräkna optimal färdväg med hjälp av A*-algoritmen. För att säkerställa god precision vid materialleveransen används en mekanisk lösning med dockstation och fixtur.

Abstract

Efficient and flexible material handling is getting increasingly important in the manufacturing industry. One way to meet this challenge is to use an Automatic Guided Vehicle (AGV). An AGV is a driverless vehicle that can be programmed to perform different tasks. This report describes the process of designing an AGV that will deliver parts to an automated assembly line for model cars. High precision in the material delivery is required of the AGV and it also has to navigate through a narrow corridor. The AGV also has a side mission where it will find and navigate to red soda cans. To meet the requirements of the AGV it uses a Sick NAV200 laser tracking system that is linked to a CompactRIO controller computer. The CompactRIO is also connected to ultrasonic sensors which are used for collision avoidance. A Microsoft Kinect camera is mounted on the AGV and connected to a laptop to enable the AGV to find red soda cans. The laptop is also used to calculate the optimal route by the A*-algorithm. A mechanical solution consisting of a docking station and a fixture is used to ensure good precision in the material delivery.

INNEHÅLLSFÖRTECKNING

1 Inledning

Denna rapport beskriver kandidatarbetet SSYX02-12-09 ”Styrning av Servicerobot” som genomförts vid institutionen för signaler och system på Chalmers Tekniska Högskola (Chalmers) under vårterminen 2012.

Kandidatarbetet är ett av flera kandidatarbeten som samarbetar för att färdigställa en monteringslina för modellbilar i laboratoriet på institutionen produkt- och produktionsutvecklings (PPU-labbet) på Chalmers. Liknande kandidatarbeten har utförts tidigare och deras resultat har kunnat utnyttjas i detta projekt. Ett likadant kandidatarbete har utförts parallellt med detta (i fortsättningen benämnd som AGV2), visst samarbete har förekommit mellan grupperna.

1.1 Bakgrund

En AGV (Automatic guided vehicle) är en autonom robot som kan utföra ett flertal olika uppgifter. De utvecklades ursprungligen på 1950-talet för att ersätta manuellt styrda truckar i tillverkningsindustrin [1]. Ursprungligen kunde de inte navigera autonomt utan följde fördefinierade vägar genom att känna av magnetiska fält som bildades av strömförande kablar i golvet. Moderna AGVer har mer avancerade navigationssystem som medger betydligt högre flexibilitet och ökade användningsområden. De vanligaste anledningarna för att välja AGVer istället för manuell varuhantering är att de är effektivare och sänker lönekostnader. De är särskilt användbara i branscher som utnyttjar skiftarbete där arbetet är repetitivt och materialintensivt. Det typiska exemplet är bilindustrin som har höga krav på pålitliga ”Just In Time” materialleveranser [2].

1.2 Problemdefinition och syfte

Problemdefinition lyder enligt följande: *"Projektet syftar till att sätta samman mekanik, elektronik och datorsystem för att få en funktionell AGV som sedan kan användas för att utföra olika arbetsuppgifter i ett större tillverkningssystem. Förutom grundläggande mekanik och elektronik så ska AGVn ha ett visionssystem, ett lasernavigationssystem samt möjlighet till trådlös kommunikation."*

Problemdefinitionen kompletteras av AGVns specifika arbetsuppgifter vilka är att kunna:

- Autonomt lämna råmaterial och hämta färdig produkt från tillverkningslinan med en repeternoggrannhet på 1mm i x, y och z-led.
- Undvika att kollidera med både statiska och rörliga föremål i dess omgivning.
- Som sidouppdrag kunna identifiera och plocka upp skräp.

Kandidatarbetets syfte är att färdigställa en fungerande AGV som kan användas vid montering av modellbilar i tillverkningslinan på PPU-labbet. AGVn ska också kunna användas i utbildningssyfte på Chalmers.

1.3 Avgränsningar

Projektet avgränsas enligt följande punktlista:

- Endast leveransen av råmaterial, hämtningen av färdig produkt och identifiering av skräp ska skötas autonomt av AGVn. Annat arbete som t.ex. pålastning av nytt råmaterial och uppladdning av batteri utförs manuellt.
- Sidouppdraget begränsas till att enbart innefatta identifiering och navigering till röda 33cl läskburkar. Ingen upplockning kommer att utföras.
- Leverans av råmaterial kan enbart utföras på en specifik plats i PPU-labbet.

1.4 Metod

Eftersom projektet består av flera olika sorters delproblem krävs även olika lösningsmetoder. För relativt enkla problem med flera tänkbara lösningar används brainstorming, eftersökningar på internet, samtal med personal från PPU-labbet, testkörning av befintlig AGV samt studier av tidigare kandidatarbeten för att hitta lösningsalternativ. Viktningsmatriser av hur väl lösningarna uppfyller olika önskemål används därefter för att identifiera den bästa lösningen. Komplexa problem som till exempel val av algoritm för ruttplanering och programmering löses genom omfattande litteraturstudier och experiment.

2 Teori

I detta kapitel ges det teoretiska ramverk som krävs för att förstå hur en AGV fungerar och den teoretiska bakgrund som finns för de val som har gjorts under projektet. Eftersom AGVn består av flera olika delar har teorikapitlet delats upp efter dessa delar; mekanik, bildbehandling, navigering och styrning, kommunikation, ruttplanering och kollisionsundvikning.

2.1 Mekanik

De mekaniska komponenterna på en AGV är oftast enkla och merparten av dessa kommer därför inte behandlas i denna rapport. Den mekanik som är av störst betydelse för hur väl AGVn fungerar är drivlinan.

Drivlinan på en AGV består av motor, transmission och hjul eller larvfötter. Motorn är i princip alltid elektrisk. Transmissionen kan bestå av kedja, rem eller kugghjul. Hjulen kan vara konventionella hjul, OmniWheels (hjul med rullar på rotationsbanan som gör att de kan flyttas i sidled.) [3], svivelhjul eller larvfötter m.m. Det som avgör hur väl AGVn fungerar är framförallt vilken typ av hjul som används och hur det är tänkt att AGVn ska svänga.

För att svänga kan AGVn antingen styra några av hjulen likt en bil eller driva hjulen på olika sidor av AGVn i olika hastighet eller åt olika håll likt en bandvagn [4]. Bandvagnsprincipen kräver att några av hjulen glider mot marken för att AGVn ska kunna svänga [5], det krävs inte om den styrs likt en bil. Bilprincipen innebär dock fler komponenter som ska styras. Ett

tredje alternativ är att använda OmniWheels med en motor på varje hjul vilket gör att AGVn kan styras i alla riktningar vilket kallas arkadprincipen [4].

Konventionella hjul och larvfötter kan enbart rotera åt ett håll vilket gör dem svåra att flytta i sidled. Det gör AGVn stabil men kräver att hjulen kan glida relativt obehindrat om bandvagnsprincipen ska kunna användas för att svänga. OmniWheels och svivelhjul medger förflyttning av AGVn i samtliga riktningar utan att behöva glida mot underlaget. Svivelhjul används inte för att driva AGVn framåt utan agerar enbart stödhjul. OmniWheels kan både driva och stödja AGVn.

2.2 Bildbehandling

För att kunna identifiera skräp, som i detta fall valts till en röd aluminiumburk, behövs en kamera vars bilder analyseras med någon lämplig algoritm. En av de mer kända algoritmerna i bildanalys är så kallad *eigenface* [6] som oftast används för ansiktsigenkänning. Den skulle dock kunna modifieras för att kunna användas för skräpigenkänning. En stor fördel med denna algoritm är att den testats i många olika sammanhang och visat sig fungera relativt väl [7]. Det är också enkelt att lägga till nya objekt i efterhand. Nackdelar med algoritmen är att den på förhand måste ha bilder av burkar som analyserats för att kunna jämföra de nya bilderna med de gamla. Det skulle också krävas en programmerare som kan implementera algoritmen effektivt.

En annan algoritm är att bilden genomsöks efter någon pixel som ser ut som en pixel från det eftersökta objektet. Efter att en korrekt pixel hittats undersöks omgivningen till denna pixel för att undersöka om omgivningen också ser ut som objektet. Om tillräckligt stor del av bilden överensstämmer med det sökta objektet är det sannolikt att det upphittats. En fördel med denna algoritm är att den är förhållandevis lätt att implementera för en ovan programmerare och enkel att förstå. En stor nackdel är att den är svår att anpassa för att hitta olika sorters objekt då det inte finns något enkelt sätt att få den att hitta nya sorters skräp. Den är möjligen också ineffektiv då den har mycket data som måste analyseras.

2.3 Navigering och styrning

För att AGVn ska kunna förflytta sig från en punkt till en annan krävs det att den vet sin nuvarande position. För att ta reda på sin position kan en rad metoder användas, exempelvis triangulering, död räkning eller krysspejling. Död räkning bygger på att startposition och hastighet är känd. För att detta ska fungera krävs att information om rotationshastigheten och friktionen för hjulen är exakt vid varje tillfälle och att ingen yttre faktor påverkar AGVns position. Triangulering bygger på att vinkeln eller avståndet till minst tre kända objekt kan beräknas och med hjälp av den informationen samt de tre objektens position kan den egna positionen beräknas. Vinkeln eller avståndet kan bestämmas visuellt eller med hjälp av till exempel radiovågor. Ett känt exempel är GPS där avståndet till olika satelliter fås och positionen beräknas utifrån det. Tyvärr fungerar inte GPS så bra inomhus eftersom signalerna blockeras. Ett vanligt sätt att få sin position inomhus är att skicka ut laserstrålar som träffar reflektorer och genom att mäta de vinklar

där ljuset reflekteras så kan positionen beräknas. Fördelen med denna metod är att man när som helst kan beräkna sin egen position så länge tre reflektorer är synliga.

När AGVns position och riktning är känd kan informationen användas för att styra AGVn till rätt plats. Det finns flera tänkbara metoder för att lösa detta, t.ex. kan en linje mellan de två punkterna skapas där sedan regleringen försöker minimera avståndet till linjen. En annan metod är att AGVn alltid ska sikta på målpunkten, denna metod är lättare eftersom den kräver mindre avancerade beräkningar. För att avgöra lämplig hastighet rakt fram och hur mycket den bör svänga kan en PID-regulator användas. Det är också möjligt att köra med konstant hastighet eller välja olika fasta hastigheter i olika avståndsintervall.

2.4 Kommunikation

För att AGVn ska fungera som en del av en större produktionscell krävs det att den på något sätt kan kommunicera med övriga delar av cellen för att koordinera arbetet. Kommunikationen behöver ske trådlöst då AGVn måste kunna röra sig fritt i cellen. Det kan lösas med till exempel en OPC-server [8]. OPC är ett flertal standarder, den vanligaste är OPC Data Access som behandlar hur data ska överföras mellan olika maskiner. En OPC-server fungerar genom en lista med delade variabler som kan användas för att enskilda maskiner ska kommunicera med huvuddatorn. Kommunikationen kan även skötas med en TCP/IP-socket där information skickas i strängar mellan huvuddatorn och olika maskiner.

Koordinerationen av cellen kan skötas av en PLC (Programmable Logic Controller), vilket är en dator som är specialgjord för att kunna användas i en industrimiljö och är lämpad för att hantera många in- och utsignaler.

2.5 Ruttplanering

A*-algoritmen är en algoritm för att finna den bästa rutten mellan två punkter. Den beskrevs första gången av Peter Hart, Nils Nilsson och Bertram Raphael 1968. Rätt implementerad hittar A* alltid en rutt om någon existerar, dessutom hittas alltid den bästa rutten [9]. A* används flitigt inom framförallt datorspelsutveckling men den är tillämpbar överallt där problemet är att hitta den kortaste vägen baserad på en känd karta. A* nyttjar sig av positioner på kartan, så kallade noder, som kan vara sammankopplade på olika sätt, t.ex. som ett rutnät eller med polygoner.

A* beräknar kostnaden f(x) för varje nod som passeras för att ta sig mellan en startnod och en målnod. f(x) är summan av två delar:

- g(x) som är kostnaden för att ta sig till den aktuella noden.
- h(x) som kallas den heuristiska delen, h(x) är en uppskattning av kostnaden som återstår för att nå målnoden.

A* nyttjar två listor, ofta benämnda *openlist* och *closedlist*. Implementeringen ser ut som följer:

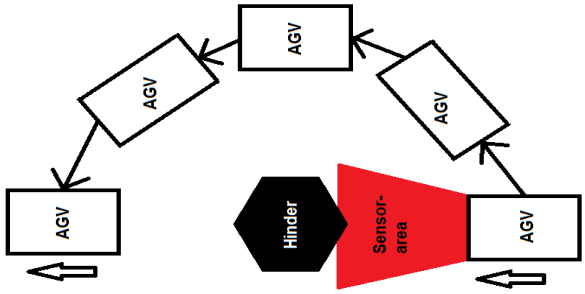
1. Börja i startnoden, lägg till den till *openlist*.
2. Repetera följande steg tills målnoden är tillagd på *closedlist* eller tills *openlist* är tom (i detta fall finns det ingen tillgänglig väg mellan start och mål).
    a. Hitta den nod på *openlist* med lägst f-kostnad, denna kallas den aktuella noden.
    b. Lägg till den aktuella noden till *closedlist* och ta bort den från *openlist*.
    c. Kontrollera alla angränsande noder, ignorera de noder som är på *closedlist* och de noder som inte kan beträdas, t.ex. väggar.
        - Beräkna g-, h- och f-kostnaden för de angränsade noderna.
        - Om de inte finns på *openlist*, lägg till de nåbara noderna till *openlist* och lägg till den aktuella noden som deras förälder.
        - Om noden finns på *openlist*, jämför g-kostnaderna för att hitta den bästa vägen. Om g-kostnaden som redan är på *openlist* är lägre, gör ingenting. Annars ändra förälder för noden på *openlist* till den aktuella noden och beräkna om f-kostnaden för den.
3. För att generera rutten gå baklänges från målnoden och se vilken nod som är förälder tills startnoden nås.

Värt att observera är att A*-algoritmen endast genererar den bästa rutten efter de förutsättningar som är kända vid beräkningstillfället. Eventuella dynamiska och plötsligt uppdykande hinder kan behöva andra system för upptäckt och undvikande.

2.6 Kollisionsundvikning

Som tidigare nämnts är en viktig funktion för att AGVn ska kunna förflytta sig i rummet att den ska kunna undvika eventuella kollisioner med plötsligt uppdykande hinder. Kinect-kameran som används till skräpidentifiering kan detektera avstånd till föremål och skulle därför kunna användas. Den kan upptäcka och avgöra avståndet till föremål i intervallet 0,8 till 4,6 meter och är därmed mycket begränsad i hur nära den kan se föremål vilket gör att den inte är ett särskilt bra val för mobila robotar utan att använda ytterligare stödjande sensorer [10]. Alternativ till Kinect-kameran är t.ex. laser-, IR- och ultraljudssensorer. Laser- och IR-sensorer mäter oftast avståndet endast till en punkt, men det finns också de lasersensorer som har en roterande axel och kan på så vis mäta avstånd till objekt i en cirkel kring lasern. Ultraljudssensorer mäter avståndet till det närmaste objektet i en cirkelsektor.

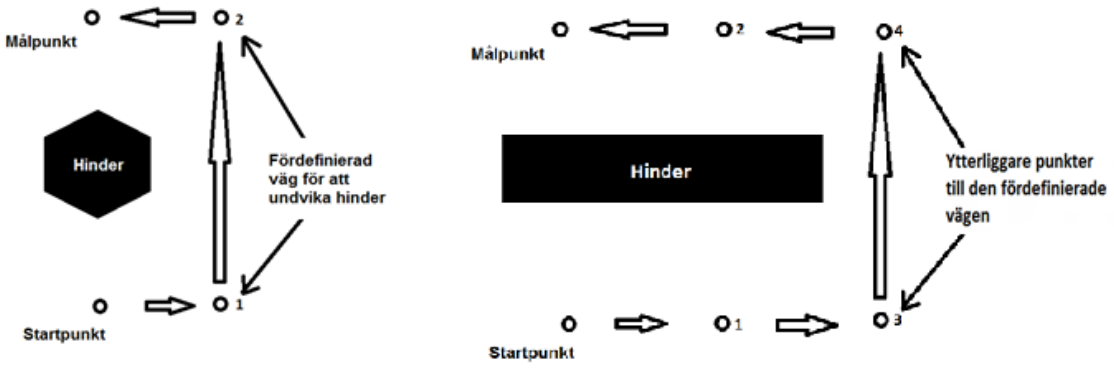
Tre vanliga metoder för att undvika objekt som detekterats av sensorer är *bug*-algoritmen, *potential field*-metoden och *vector field histogram*. *Bug*-algoritmen [11] är den enklaste av metoderna, den fungerar genom att roboten förflyttar sig runt objekt som identifierats genom att hålla ett konstant avstånd till objektet till dess att objektet passerats (se figur 1).



Figur 1: Illustration av Bug-algoritmen. När AGVn upptäcker ett hinder håller den sig på ett konstant avstånd från hindret till det har passerats.

Potential field-metoden innebär att varje hinder ger upphov till en imaginär repellerande kraft och målet ger upphov till en imaginär attraherande kraft. Resultanten av dessa två krafter blir den riktning som roboten kommer att röra sig mot [12]. *Vector Field Histogram* innebär att i roboten finns en karta uppbyggd av kvadrater. Varje kvadrat har ett värde och varje gång en sensor finner ett objekt i en av dessa kvadrater så ökar värdet i den kvadraten. Ett högt värde i en kvadrat innebär att det är stor sannolikhet att det finns något där och roboten undviker då dessa rutor [13].

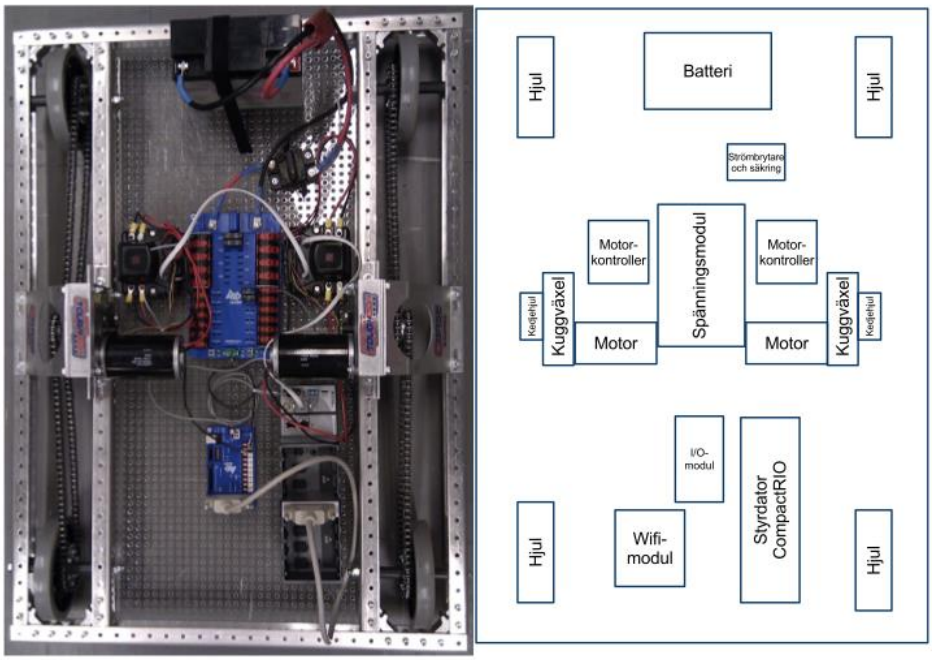
En förenklad version av *Bug*-algoritmen utvecklades. Den fungerar genom att AGVn testar en fördefinierad väg för att ta sig runt ett standardiserat hinder som kan förekomma i lokalen (människa eller AGV). När ett hinder upptäcks läggs två extra punkter till som AGVn ska passera på väg till sitt mål. AGVn använder sig sedan av de främre sensorerna för att bekräfta att denna fördefinierade väg saknar hinder. Om vägen är tillgänglig tar sig AGVn runt objektet. Om den fördefinierade vägen är blockerad lägger AGVn till ytterligare punkter i den fördefinierade vägen och proceduren upprepas tills AGVn fullständigt passerat hindret (se figur 2).



Figur 2: Till vänster syns den förenklade bug-algoritmen med en iteration där punkterna 1 och 2 har adderats till AGVns färdväg. Till höger används två iterationer, eftersom sträckan mellan punkt 1 och 2 inte var fri från hindret lades även punkt 3 och 4 till i färdvägen.

3 Material

Materialet att utgå från bestod av föregående års AGV, PPU-labbet och en modellbil i aluminiumplåt. En översikt av AGVn ses i figur 3 och en noggrannare beskrivning av de olika komponenterna ges nedan.

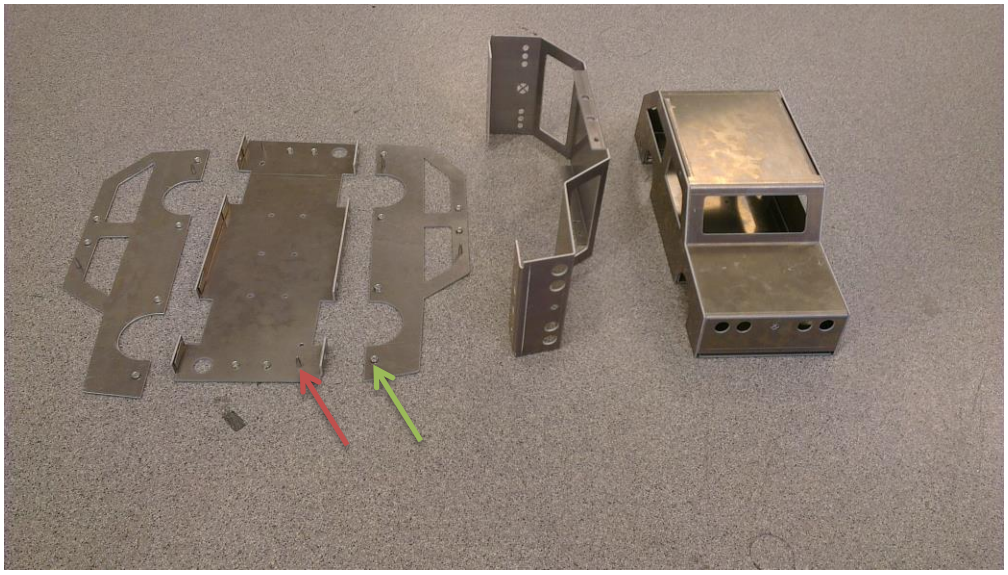


Figur 3: Översikt av de ursprungliga komponenterna på AGVn. Hämtad ur föregående års rapport [14].

PPU-labbet var inte färdigt i projektets början men färdigställdes under projektets gång. En bild på det nästan färdiga PPU-labbet visas i figur 4.

Figur 4: Översiktsbild av PPU-labbet. Gången till höger är där AGVn ska åka in i cellen. Den gröna pilen visar dockstationen som konstruerades.

Modellbilen konstruerades av en annan kandidatgrupp och resultatet visas i figur 5. Den hålls ihop av magneter och styrpinnar som trycks in i små hål i plåten.



Figur 5: Modellbilen som ska monteras i tillverkningscellen. Grön pil visar en av magneterna och röd pil visar en av styrpinnarna.

3.1 Motorkontroller

Motorkontrollerna är från Texas Instruments och har modellbeteckning MDL-BDC24 [15]. De tar emot digitala signaler och ger lämplig spänning till respektive motor. Motorkontrollerna kan seriekopplas och kopplas till en encoder monterad på växellådan som ger information om hastighet och sträcka.

3.2 Motorer

Motorerna tillverkas av AndyMark och har modellbeteckning AM802-001A, de är 12 volts likströmsmotorer med en maximal effekt av 337 watt [16].

3.3 Styrenhet och I/O modul

AGVn har två styrenheter. Den ena är en National Instruments cRIO-9074 (CompactRIO) som programmeras i LabVIEW. Det är en dator med 400MHz realtidsprocessor och 256MB RAM-minne. Den har två Ethernet-portar, en RS232-port och 8 expansionsplatser för olika expansionsmoduler [17]. CompactRIOn är kopplad via RS232-porten direkt till de båda motorkontrollerna och en av de två Ethernet-portarna är kopplad till en trådlös adapter. I en expansionsplats sitter en modul med 4st RS232-portar, NI9870, som fungerar som en hubb. I en annan expansionsplats sitter en I/O-modul, NI9403, som kan skicka och ta emot upp till 32 olika logiska signaler.

Den andra styrenheten är en laptop från Asus med modellbeteckningen 1011PX. Den har en två-kärnig processor med klockfrekvensen 1,67 GHz och 1GB RAM-minne samt är försedd med ett trådlöst nätverkskort [18]. Laptoppen används för bildbehandling och ruttplanering med program skrivna i C#.

3.4 Positioneringssystem

Positioneringssystemet består av en Sick NAV200 laserskanner. Den fungerar genom att laserpulser skickas ut från ett roterande torn. Laserpulserna reflekteras mot utplacerade reflektorer i rummet och om minst tre reflektorer är synliga kan Sick NAV200 med hjälp av triangulering bestämma AGVns position med en noggrannhet på 4mm och 0,1 grader [19]

3.5 Kamera

Kameran är en Microsoft Kinect som kan mäta både djup och färg. Den kopplas med USB-sladd till laptoppen där bildbehandling utförs.

3.6 Trådlöst nätverk

AGVn kommunicerar med det trådlösa nätverket i PPU-labbet med en Linksys WGA600N [20] som är kopplad till CompactRIO.

4 Metod

För att säkerställa att projektet skulle resultera i en väl fungerande AGV upprättades en övergripande kravspecifikation (se bilaga 1). Eftersom problemdefinitionen var mycket omfattande gjordes även en funktionsanalys för att bryta ned huvudproblemet till delproblem. Funktionsanalysen ses som ett träddiagram i bilaga 2. Både kravspecifikationen och funktionsanalysen utfördes genom brainstorming, samtal med ansvariga för PPU-labbet och studier av tidigare års kandidatarbeten. De olika delproblem som identifierades i funktionsanalysen var av olika karaktär och lösningsmetoderna skiljde sig därför åt. Nedan beskrivs lösningsmetoderna för de olika delområdena av AGVn.

4.1 Mekanik

De mekaniska delproblemen var framdrivning och repeternoggrannhet vid materialleverans. Arbetet med framdrivning utfördes i samarbete med AGV2, arbetet med repeternoggrannheten utfördes utan något samarbete.

4.1.1 Framdrivning

Framdrivningsproblemet löstes genom en omfattande generering av lösningsförslag och upprättande av en specifik kravspecifikation. Lösningsförslagen grundades på tester av befintlig AGV, brainstorming, informationssökning på Internet, samtal med teknisk expertis och studier av föregående års rapporter.

För att hitta den bästa lösningen sållades lösningsförslagen som inte uppfyllde kraven i kravspecifikationen bort. I kravspecifikationen ingick även önskemål. För att bestämma vilka önskemål som var viktigast användes viktningsmatriser och för att hitta den bästa lösningen jämfördes lösningsförslagens förmåga att uppfylla önskemålen.

4.1.2 Repeternoggrannhet

Eftersom industrirobotarna i PPU-labbet inte hade möjlighet att mäta in var AGVn befann sig behövdes mycket god precision vid leveransen av råmaterial. Därför ansågs det nödvändigt att säkerställa repeternoggrannheten med en mekanisk lösning.

För att hitta lösningsalternativ användes samma metod som för framdrivningen. Metoden för att hitta den bästa lösningen begränsades till att enbart identifiera de lösningsförslag som uppfyllde kraven i den specifika kravspecifikation som skapades och därefter utvärdera vilken lösning som ansågs lättast att tillverka. Tillverkningen av det vinnande lösningskonceptet utfördes i prototyplaboratoriet på Chalmers och bestod av plåtarbete, fräsning och svetsning.

4.2 Bildbehandling

För att kunna hitta skräp användes en Kinect-kamera som är utrustad med både färg och djupseende. För att programmera denna kamera användes Microsoft egna utvecklingspaket (SDK) till Kinect. All programmering utfördes sedan i C#. Då det fanns många funktioner tillgängliga i denna SDK var det viktigt att undersöka vilka funktioner som verkligen behövdes för att bildanalysen skulle fungera. De funktioner som inte behövdes stängdes av för att spara processorkraft. Laptoppen som utförde analysen hade begränsad beräkningskapacitet vilket gjorde det viktigt att säkerställa att bildanalysen kunde utföras i den takt som behövdes.

Den egengjorda algoritm som beskrivits i avsnitt 2.2 valdes på grund av dess enkelhet. För att avgöra avståndet och vinkeln till den funna burken användes djupkameran. Med hjälp av geometri och vetskapen om hur stor vinkel kameran ser kan sedan vinkeln till burken avgöras.

4.3 Navigering och styrning

För att kommunicera med Sick NAVen, PLCn, bildbehandlingsdatorn, ultraljuden och motorkontrollerna, som i sin tur styr motorerna, användes CompactRIOn. Den programmerades i National Instruments eget programmeringsspråk LabVIEW med tilläggen Real Time och FPGA. Real Time gjorde det möjligt att kommunicera samt lägga in program på CompactRIOn och FPGA-tillägget gjorde det möjligt att programmera de båda insticksmodulerna. LabVIEW är ett nästan helt grafiskt programmeringsspråk med många inbyggda subrutiner (subVIs) och dessutom finns det gott om subVIs som exempelvis styr motorkontrollerna att ladda ner från National Instruments hemsida.

Programmeringen utfördes på en laptop som skickade över det färdiga programmet till CompactRIOn. Navigeringsprogrammet byggdes ut steg för steg under hela projektets gång och kontinuerliga testkörningar identifierade eventuella svagheter som kunde åtgärdas.

4.4 Kommunikation

Den lösning för kommunikation som valdes var att låta informationen till och från AGVn gå via en TCP/IP-socket, detta trots att det ursprungligen skulle ske via en OPC-server med en delad variabellista. All kommunikation utfördes med en *integer* (ett heltal) vars olika värden symboliserar olika beställningar, bekräftelser eller förfrågningar.

Programmeringen av PLCn gjordes i så kallade ladder-diagram, att likna med reläkopplingar. Programmeringen delades upp i olika operationer för AGVn, exempelvis "åk till laddstation" eller "åk till robotcell". PLCns uppgift är att undersöka om vissa krav är uppfyllda och skicka en signal till AGVn om vilken operation som ska utföras. Ett exempel är operationen "åk in i robotcell", då kontrollerar PLCn att AGVn har lastats med råmaterial, att råmaterialet inte är färdigmonterat, att dörren till robotcellen är öppen, att den andra AGVn inte är i cellen etc. Är dessa krav uppfyllda så skickas en siffra till AGVn som symboliserar att den kan åka in. PLC programmeringen utfördes i samarbete med andra kandidatgrupper som också arbetade i PPU-labbet.

4.5 Ruttplanering

För att säkerställa att AGVn kan fungera autonomt krävs ett system för att planera den bästa rutten. I det aktuella fallet har detta system utvecklats i det objektorienterade programmeringsspråket C#. Även ett grafiskt användargränssnitt och program för tester och verifiering har utvecklats. Som stöd och referenslitteratur har boken ”Skarp programmering med C#” av Jan Skansholm använts [21]. Kontinuerliga tester utfördes för att säkerställa systemets funktion och tillräknelighet (se avsnitt 5.5).

4.6 Kollisionsundvikning

För att välja kollisionsundvikningssystem gjordes grundliga litteraturstudier där de olika sensorerna som presenteras i avsnitt 2.5 undersöktes. Olika sensortypers egenskaper utvärderades och det undersöktes om de använts tidigare vid liknande robotprojekt. Dessutom undersöktes hur sensorerna kan fungera med de olika algoritmer som beskrivs i avsnitt 2.5. Det lades även stor vikt på att jämföra kostnaden för sensorerna och deras komplexitet eftersom projektet hade en begränsad tid och budget.

När ett passande kollisionsundvikningssystem valts beställdes ett mindre antal sensorer för att testa systemet och därefter bestämdes om sensorerna skulle användas på AGVn.

5 Resultat

Kandidatarbetet har resulterat i en AGV som kan leverera råmaterial och hämta en färdig bil med en repeternoggrannhet på en halv millimeter i x-, y- och z-led. AGVn tar autonomt emot beställningar om när den behövs i tillverkningslinan och den undviker att kollidera med både rörliga och statiska objekt i sin omgivning. AGVn kan identifiera röda läskburkar och navigera till dem för att visa för människor var burkarna ligger. Nedan ges en utförligare beskrivning av arbetet med delproblemen.

5.1 Mekanik

Arbetet med lösningssökning och utvärdering av lösningsförslag resulterade i en AGV med OmniWheels istället för konventionella hjul och en dockstation för att säkerställa precisionen vid leverans av råmaterial.

5.1.1 Resultat av framdrivning

Vid samtal med teknisk expertis som deltagit vid tidigare arbeten med AGVer i PPU-labbet framkom att ett av de största problemen med AGVerna var att de fastnade i ojämnheter i golvet. En undersökning av problemet visade att ojämnheterna bestod av kabelrännor som täckts över med betongblock. Nivåskillnaden uppmättes till maximalt en centimeter. Testkörning av AGVn visade att den problemfritt kunde ta sig över kanter på en centimeter så länge den körde ungefär vinkelrätt mot kanten. Det största problemet tycktes vara när AGVn skulle svänga. Konstruktionen med 4 hjul gör att om AGVn ska rotera runt en punkt utan att köra framåt måste några av hjulen slira. Friktionen mellan mark och hjul var dock så hög att hjulen inte slirade fören motorerna kördes med högre effekt än 15 %. Dessutom tenderade den att göra små hopp runt istället för att svänga kontrollerat. Problemet var alltså en kombination av AGVns oförmåga att rotera och att hjulen vid rotation fastnade mot kanterna i golvet.

Arbetet med lösningssökning och utvärdering av lösningsförslag resulterade i två lösningar som visade sig bättre än de övriga. Den ena lösningen bestod av att montera OmniWheels på samtliga fyra hjul. Den andra lösningen var att montera ett svivelhjul i varje hörn på AGVn och ett fjäderupphängt drivande hjulpar i mitten. Eftersom arbetet utfördes tillsammans med AGV2 bestämdes det att båda lösningar skulle testas. Detta kandidatarbete valde att testa OmniWheels medan AGV2 testade svivelhjul. För komplett beskrivning av lösningsförslag och elimineringsprocessen se bilaga 3.

Testkörningar av lösningarna visade att båda kunde svänga med lägsta möjliga effekt på motorerna och båda kunde ta sig fram över de ojämnheter som fanns i golvet på PPU-labbet. Dessutom kunde båda köra in i dockstationen utan att fastna.

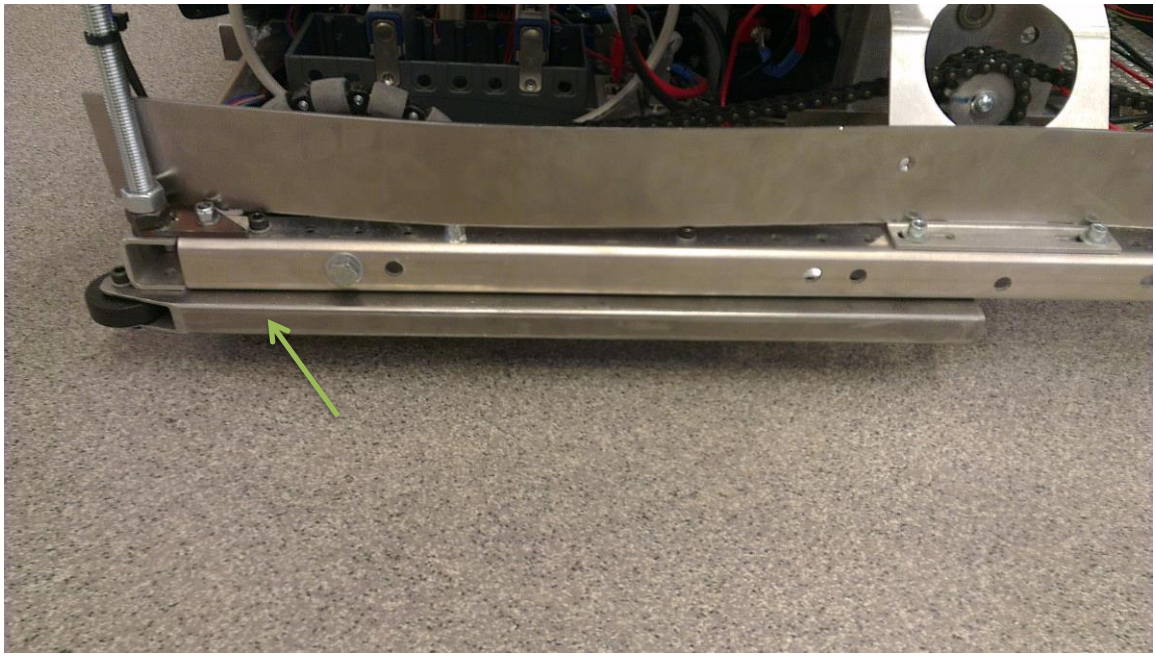
5.1.2 Repeternoggrannhet

Samtal med teknisk expertis, brainstorming och testkörning av befintlig AGV resulterade i ett flertal lösningsförslag och en kravspecifikation. De lösningar som inte uppfyllde samtliga krav sållades bort och slutligen valdes det lösningsförslag som ansågs lättast att tillverka. Se bilaga 4 för en komplett beskrivning av lösningssökning, eliminering och tillverkning.

Den vinnande lösningen blev en dockstation i stålplåt. Dockstationen konstruerades i CATIA V5R19 och tillverkades därefter i prototyplaboratoriet på Chalmers. Den färdiga dockstationen ses i figur 6.

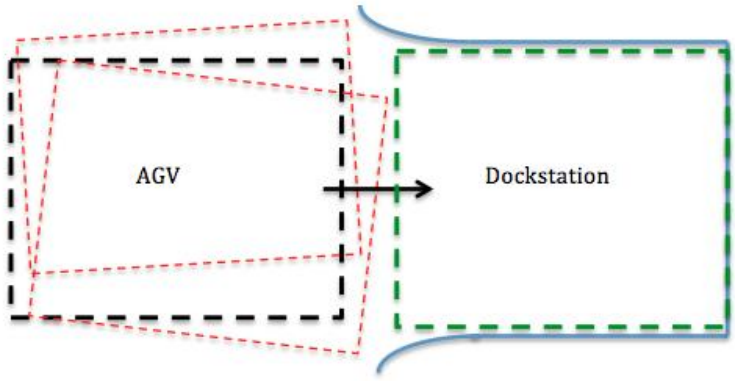
Figur 6: Närbild på dockstationen.

Dockstationen kompletterades med glidskenor på AGVn (se figur 7). Både dockstationen och glidskenorna gjordes justerbara så att dockstationen kunde kalibreras för båda AGVerna.



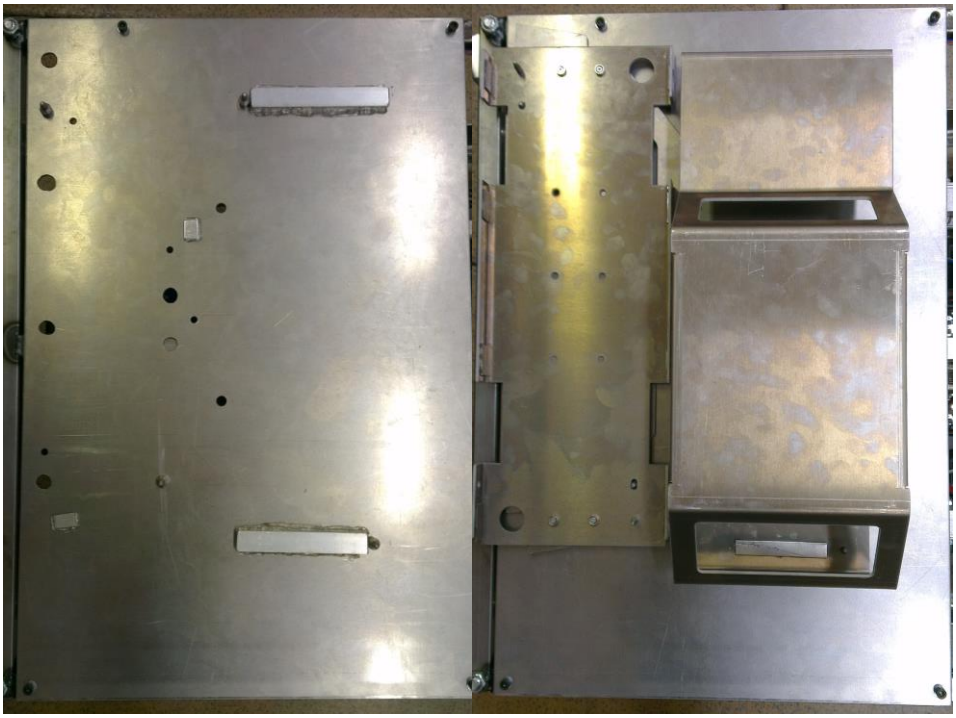
Figur 7: Glidskenan som glider mot dockstationen markerad med grön pil.

Testkörningar av dockstationen utfördes genom 20 försök där AGVns ursprungliga position förskjutits en till fem centimeter och riktats 0 till tio grader fel från dess optimala position (se figur 8). Resultatet visade att AGVn i samtliga fall lyckades köra in i dockstationen och nå sin korrekta slutposition med en maximal avvikelse på ±0.5mm i x-, y- och z-led.



Figur 8: Illustration av funktionstesterna på dockstationen.

För att hålla råmaterialet på plats konstruerades en fixtur av stålplåt. I plåten borrades hål med en tolerans på avståndet mellan hålen på ±0.005 mm som modellbilens styrpinnar (se figur 5) kunde placeras i. Fixturen försågs även med liknande styrpinnar som modellbilen för att hålla fast de delar som inte hade egna styrpinnar som kunde användas. En bild på fixturen med och utan monterad modellbil ses i figur 9.



Figur 9: Fixturen utan och med modellbilen monterad.

5.2 Bildanalys

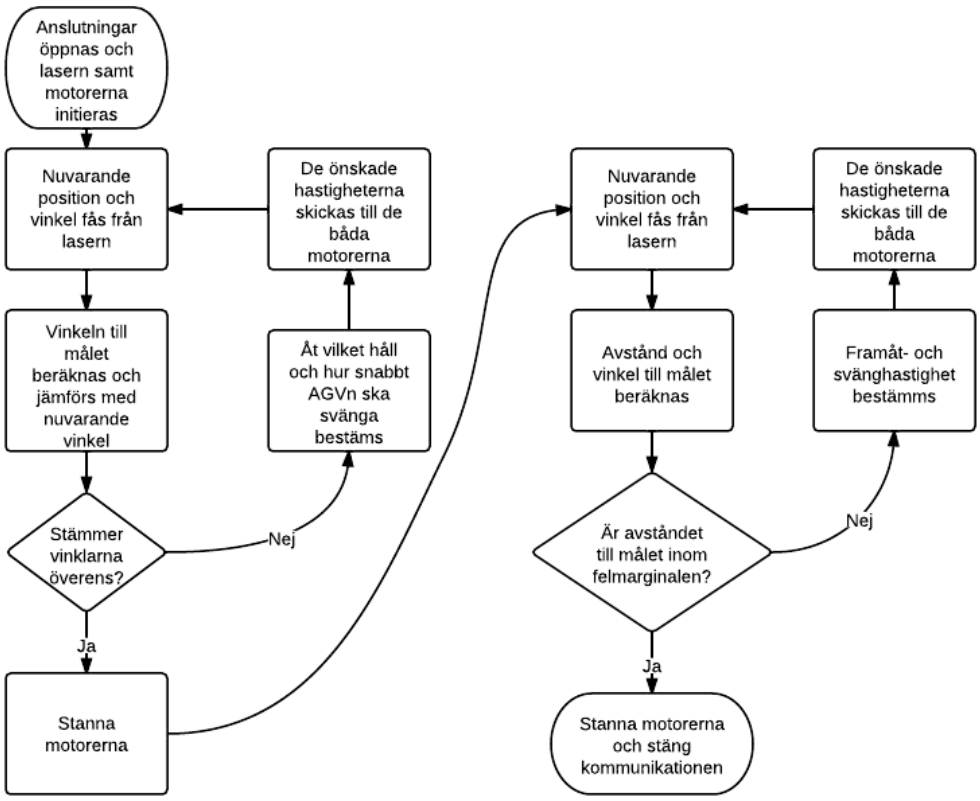
Då det visade sig mycket svårt att avgöra hur bra bildanalysen skulle fungera i praktiken genomfördes fortlöpande tester och optimering efterhand som programmet skrevs. Slutresultatet blev att laptoppen utan problem kunde ta in bilder från Kinecten i 30 fps i upplösningen 640*480 pixlar, vilket är maximalt för Kinecten.

Analysen av bilderna skedde parallellt med inspelningen och klarade vid optimala förhållanden av att hitta en burk flera gånger i sekunden och avgöra dess avstånd från AGVn. Då det är ett mindre område av bilden som täcks av en burk när den flyttas bort från Kinecten blev det svårare för algoritmen att finna en burk. Bortanför ca 2 meter kunde det ta flera sekunder innan burken hittades, om den över huvud taget hittades. Då Kinecten inte kan avgöra avståndet till föremål som är närmare än ca 8 decimeter kunde inte avstånden till de burkar som placerades för nära bestämmas. Vinkeln till burken beräknades baserat på rätvinkliga trianglar. Denna approximation medförde vissa beräkningsfel. Då burken står mindre än fem grader i förhållande till Kinectens centrum kunde inte avrundningsfelen urskiljas. Vid 20 grader beräknade programmet vinkeln till 18 grader.

5.3 Navigering och styrning

Eftersom CompactRIOn är den centrala delen när det gäller kommunikationen ombord på AGVn var många av de egenhändigt konstruerade programmen tvungna att behandla just kommunikation mellan olika enheter på AGVn. Den viktigaste kommunikationen var mellan Sick NAVen som skickar nuvarande position till CompactRIOn som sedan kommunicerar med motorkontrollerna. Vidare behövdes även program som reglerar styrningen, analyserar svar från ultraljuden och kommunicerar med PLCn och laptoppen via nätverket. Även ett program som kan koppla ur det autonoma systemet och i nödfall gå över till manuell styrning med hjälp av en joystick har konstruerats. Se bilaga 5 för en komplett lista över de LabVIEWprogram som användes.

Ett grundläggande SubVI som ändå ligger högt upp i hierarkin är det som får AGVn att ta sig från punkt A till punkt B. Detta program fungerar genom att få AGVn att först stå still och svänga mot målet. Vinkeln till målet beräknas och jämförs med den nuvarande vinkeln som fås från Sick NAVen. Den riktning som ger minsta vinkelskillnaden väljs och sedan används den vinkelskillnaden för att reglera hastigheten. Hastigheten regleras genom att vinkelskillnaderna delas in i tre intervall, där varje intervall ger en bestämd hastighet. Detta visade sig vara en mycket snabbare reglering än en PID-reglering eftersom det uppstår en hel del fördröjning när positionen ska beräknas och skickas från lasern till CompactRIOn. När AGVn sedan siktar rakt mot målet regleras hastigheten framåt och hur mycket AGVn ska svänga, på liknande sätt som ovan, så att AGVn alltid siktar rakt mot målet. Processen pågår tills den nuvarande positionen är inom en bestämd felmarginal, hela förfarandet illustreras i ett flödesschema i figur 10.

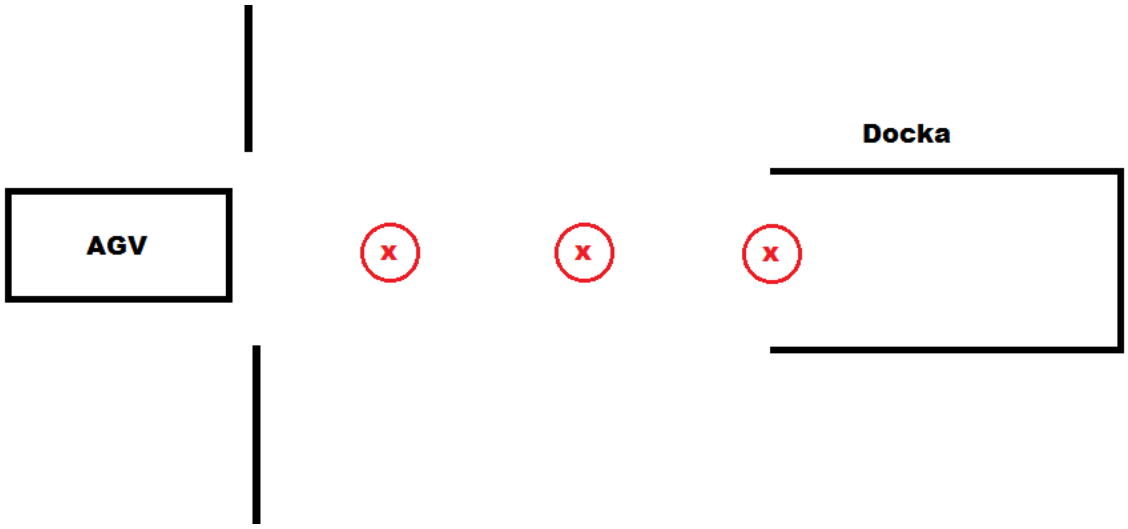


Figur 10: I figuren visas ett flödesschema över hur programmet som kör AGVn i en rak linje mellan två punkter är uppbyggt.

När detta program väl var tillförlitligt så kunde det byggas vidare genom att implementera A*-algoritmen för att bestämma bästa vägen runt fasta hinder samt kollisionsundvikning som kontrollerar att det inte är några rörliga hinder i vägen för AGVn. Både A* och kollisionsundvikningen kan lägga till extra punkter mellan AGVns nuvarande position och dess målkoordinat. Dessa punkter förflyttas till på samma sätt som ovan och om det skulle uppkomma nya rörliga hinder så läggs ytterligare nya positioner in i listan. Eftersom AGVn inte ska stanna på de punkterna så behövs inte lika hög precision som för andra punkter. När väl AGVn har nått sin målkoordinat så avslutas denna del och huvudprogrammet kan gå vidare i sin logik.

När AGVn ska leverera material gäller det att den på ett säkert sätt kan köra in i dockstationen. Eftersom det är en trång korridor som leder till dockstationen så behövde dockningssekvensen hårdkodas. Metoden som AGVn använder för att köra in i dockstationen bygger på att det finns fördefinierade punkter (checkpoints) som den ska köra förbi på väg mot dockan. I korridoren som leder fram till dockan finns det tre sådana punkter definerade. Den gör detta genom att sikta rakt mot punkten, men inom ett givet intervall så kör den endast rakt fram tills den har kommit förbi punkten och då siktar den istället på nästa punkt. När AGVn har kommit förbi den sista checkpointen kör den rakt

fram tills den är helt inne i dockstationen, lyckligtvis är dockningsstationen så pass förlåtande att även om AGVn kommer in lite snett så rätar den upp sig och kommer hela vägen in. När AGVn är inne i dockstationen verifierar den sin position med hjälp av Sick NAVen och om dockningen visar sig vara lyckad så skickas en signal till PLCn. För att köra ut ur tillverkningscellen används ett liknande program. En bild på dockningsprogrammet visas i figur 11.



Figur 11: Dockningsprogrammet fungerar genom att AGVn siktar på ett antal punkter (kryssen i figuren). När AGVn befinner sig inom ett visst område från punkten (innanför cirklarna) anser sig AGVn ha nått punkten och fortsätter därefter till nästa. När sista punkten har passerats kör AGVn rakt fram och pressar sig in i dockstationen.

5.4 Kommunikation

Kommunikationen mellan AGVn och andra maskiner i tillverkningscellen utförs med en PLC som skickar information trådlöst till AGVn. AGVn kan dessutom skicka information till PLCn, till exempel för att berätta att den vill in i tillverkningscellen. Då AGVn inte har fått någon order av PLCn åker den runt och letar efter läskburkar. De olika beställningarna ses i tabell 1.

	Från PLC	Från AGV
10	Bil beställd, åk till laddstation	I laddstation, karossdelar lastade
20	Åk till robotcell	Utanför robotcell
30	Åk in i robotcell och docka	I robotcell och docka
40	Kör ur docka, till dörr	Vid dörr
50	Åk ut genom grind	Ute ur grind
60	Lämna färdigmonterad bil	Färdigmonterad bil lämnad
70	Uppdrag färdigt, AGV ledig	AGV ledig
99	Nödstopp	Nödstopp

Tabell 1: De olika beställningarna till- och från AGVn. Kolumnen längst till vänster visas de olika beställningsnumrena. Mittenkolumnen visar vad beställningen betyder då den går från PLCn till AGVn. Den högra kolumnen visar vad beställningen betyder då den går från AGVn till PLCn.

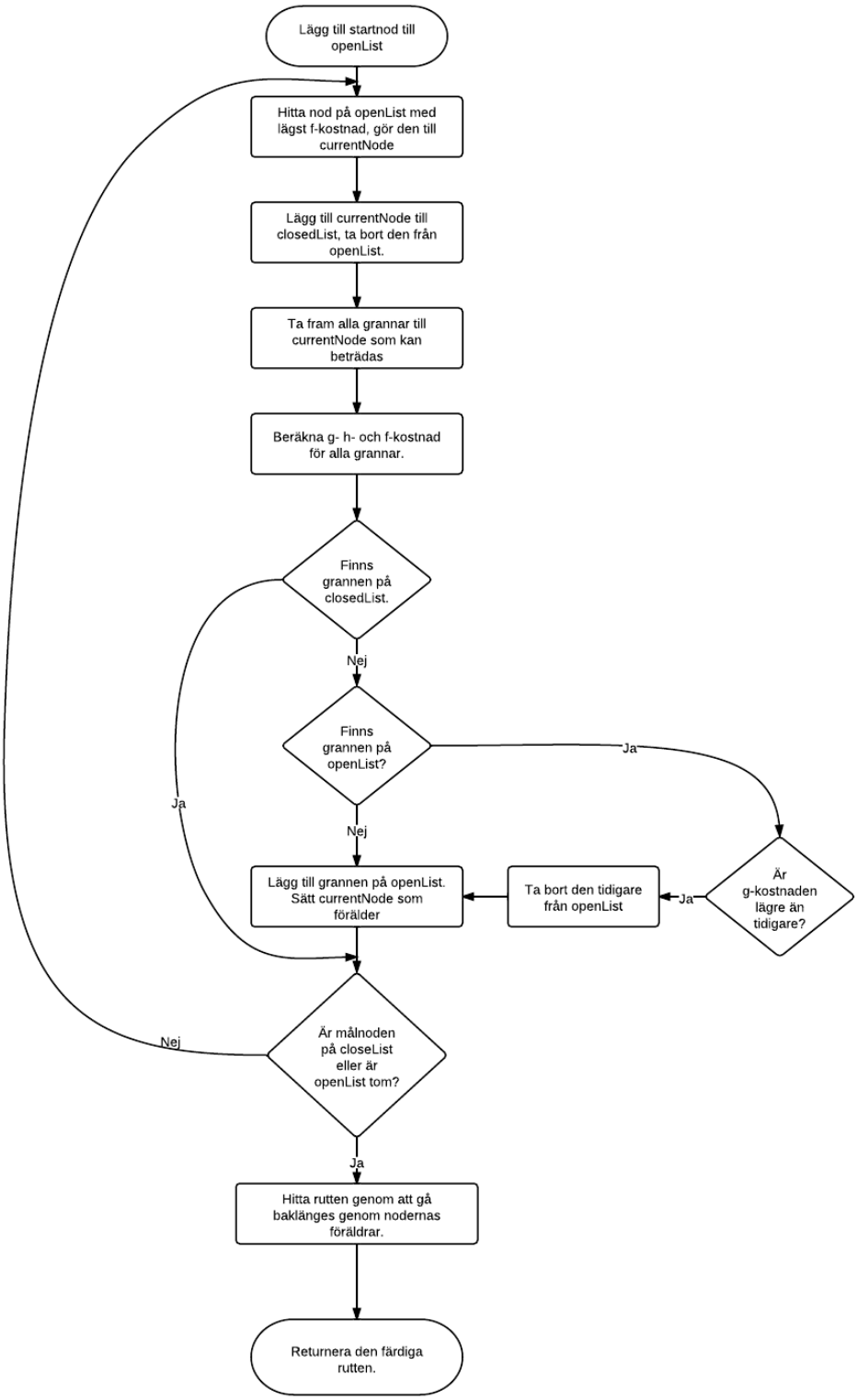
Tyvärr var det problem med TCP/IP-kommunikationen som gjorde att den behövde stängas ner och öppnas upp på nytt varje gång information skulle skickas. Det ledde till att det tog 20 sekunder att skicka och ta emot ordrar vilket gjorde kommunikationen ineffektiv.

5.5 Ruttplanering

För att hitta den bästa rutten för AGVn gjordes en omfattande litteraturstudie i ämnet. Efter litteraturstudien beslutades att ruttplaneringsalgoritmen A* skulle implementeras (se avsnitt 2.5 om A*-algoritmen). Inledningsvis var tanken att CompactRIOn skulle sköta ruttplaneringen och att den följaktligen skulle programmeras i National Instruments eget programmeringsspråk LabVIEW. LabVIEW visade sig då ha vissa begränsningar i att hantera matriser och fält. Dessutom vore det önskvärt att kunna använda ett objektorienterat programmeringsspråk. Då AGVn utöver CompactRIOn även är utrustad med en laptop beslöts att även ruttplaneringen skulle skötas av denna. För enkelhets skull valdes C# som programmeringsspråk eftersom bildbehandlingen och kommunikationen med CompactRIOn redan nyttjade detta språk.

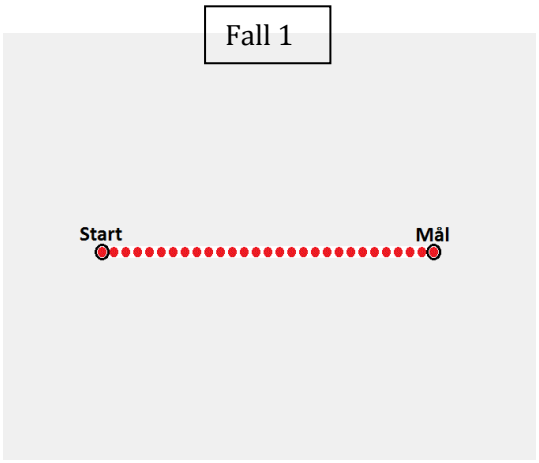
För att implementera A*-algoritmen behövs en karta eller en graf över det utrymme som ska användas. Enklast och i det aktuella fallet mest lämpade är att dela upp den berörda ytan, d.v.s. PPU-labbet, i ett rutnät. Varje ruta i nätet motsvarar sedan en nod. Upplösningen på rutnätet kan naturligtvis väljas fritt beroende på de krav man har att förhålla sig till. I det aktuella fallet krävs hög precision endast inne i själva produktionscellen. Där finns dock inte något behov av att beräkna rutten med hjälp av A*-algoritmen då AGVns tillåtna område endast består av en smal rak korridor. I övriga delar av PPU- labbet saknas kravet på samma höga precision som i själva produktionscellen. Med detta i beaktande har rutnätet bestämts med kvadratiska rutor med sidan 0,375 m. Detta efter en avvägning mellan krav på noggrann navigering mot den ökade beräkningskraft som erfordras och även svårigheterna att skapa en relevant karta.

I implementeringen antar kartan över området en matris av booleska värden, d.v.s. sant eller falskt. Sant i detta fall innebär att det finns ett hinder i denna nod, alltså en vägg eller annat statiskt föremål som AGVn inte kan passera. Falskt innebär att noden är framkomlig för AGVn. För en given startpunkt, som är AGVns aktuella position, och den önskade målpositionen kan den bästa rutten nu beräknas enligt flödesschemat i figur 12. Programmet kan nu returnera rutten i form av ett fält av punkter som AGVn ska färdas genom.

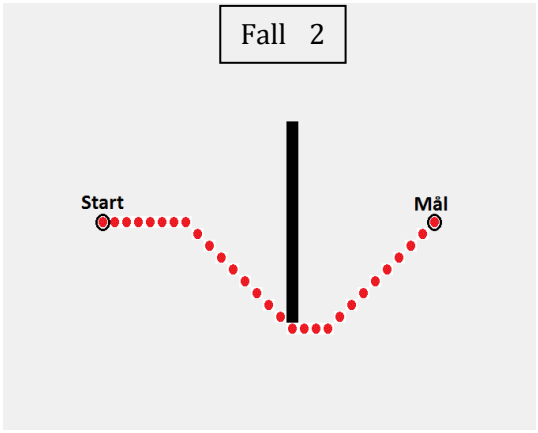


Figur 12: Flödesschema över A*-implentationen.

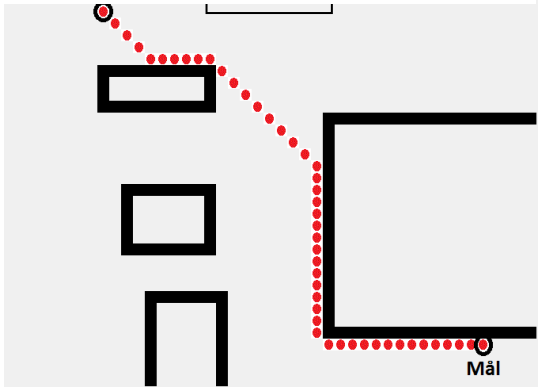
För att säkerställa att beräkningen av rutten inte är orimligt resurskrävande och därmed en möjlig flaskhals för AGVn utfördes simuleringar av några möjliga kartkonfigurationer. Upplösningen på dessa kartor är likvärdig med den karta över PPU-labbet som kommer nyttjas. Fall 1 är enklast möjliga fall, start och mål ligger på en rak linje utan hinder mellan sig, i fall 2 finns ett hinder i vägen och fall 3 har flera hinder. Dessa kartor kan beskådas i figur 13-15, svartmarkeringar symboliserar hinder och rött den beräknade rutten. Resultatet visade (se tabell 2) att rutten i fall 3 tog ca 0,8 sekunder att beräkna. Vid behov av snabbare beräkningar eller högre upplösning finns troligen utrymme för effektivisering av algoritmens implementering.



Figur 13: Enklast möjliga test av A*-algoritmen



Figur 14: Relativt enkelt test av A*-algoritmen.

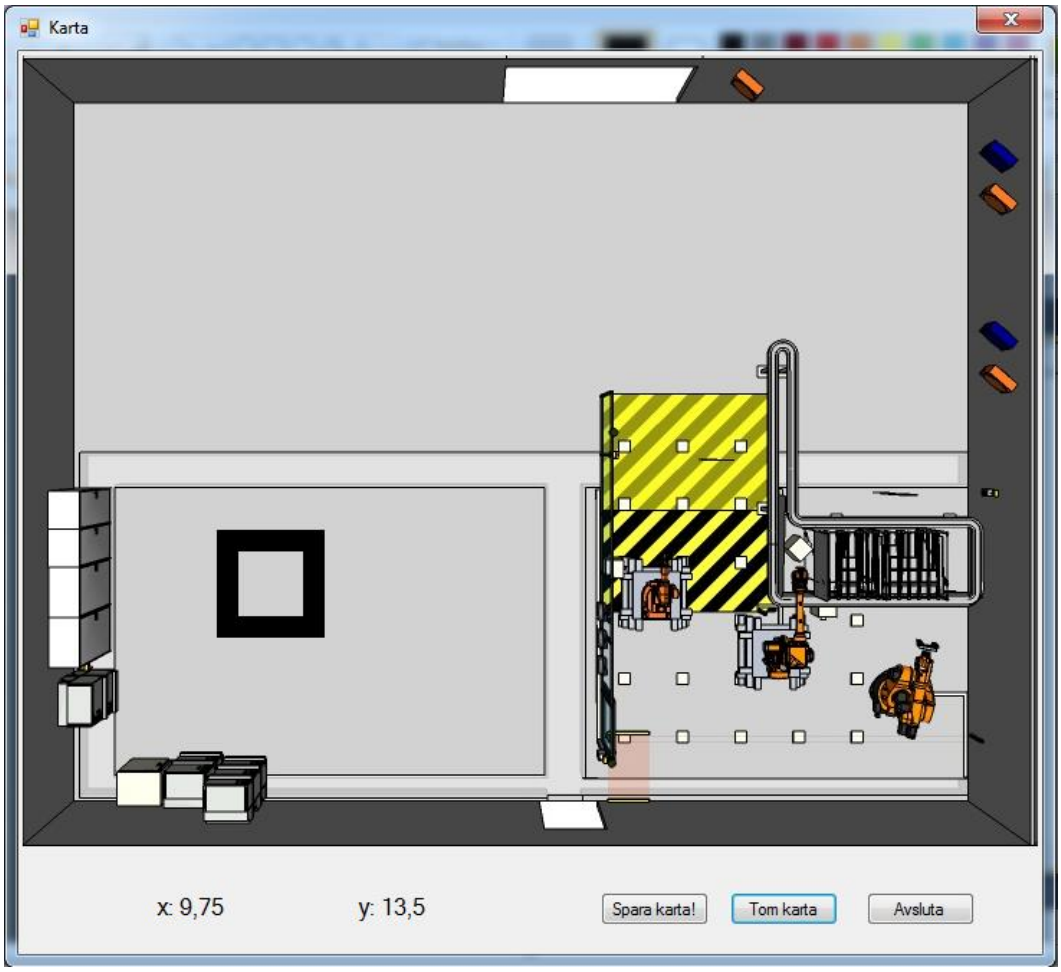


Figur 15: Mer avancerat test av A*-algoritmen.

Fall 1	0,002 s
Fall 2	0,241 s
Fall 3	0,809 s

Tabell 2: Resultat från de olika testerna av A*-algoritmen.

För att hantera den karta över PPU-labbet som behövdes i programmet har ett grafiskt gränssnitt skapats. Där kan man på ett enkelt sätt ändra kartan om föremål ställs in eller tas bort i PPU-labbet. Den karta man redigerar i programmet sparas i en textfil på laptoppen. A*-programmet läser sedan in den uppdaterade kartan vid nästa körning. En skärmdump av programmet kan beskådas i figur 16.



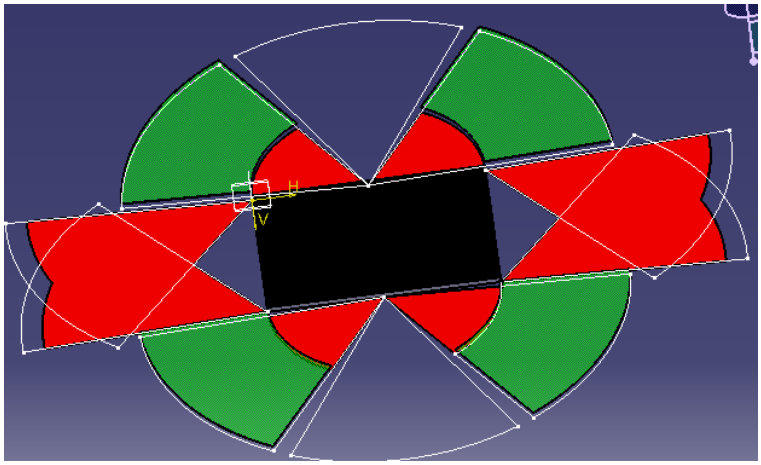
Figur 16: Grafiskt användargränssnitt för kartritning.

5.6 Kollisionsundvikning

Vid diskussion om vilken metod för att undvika föremål som var bäst lämpad lades stor vikt vid att lösningen skulle vara enkel att implementera i systemet. Dessutom sattes det som krav att sensorerna som behövdes skulle ha ett tillräckligt lågt pris så att projektbudgeten inte överskreds och att de skulle vara enkla att installera. Den metod som valdes var den så kallade *bug*-algoritmen (se punkt 2.5 i teorikapitlet). *Bug*-algoritmen valdes eftersom det är den enklaste av de föreslagna metoderna och den går fort att implementera.

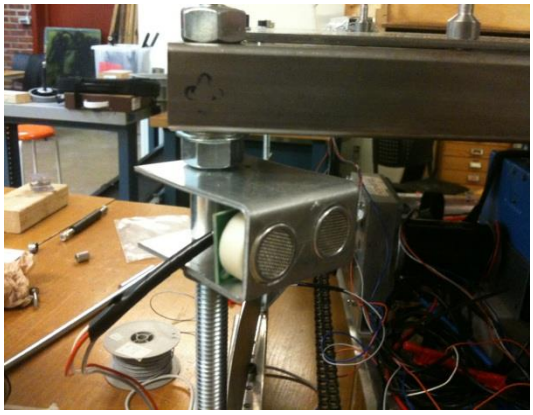
Med *Bug*-algoritmen krävs det en sensor som kan tala om hur långt roboten befinner sig från objektet som ska undvikas även när roboten har sidan åt objektet. Detta uteslöt Kinect-kameran eftersom den bara ser rakt fram och är begränsad i sin rörlighet. Budgeten räckte inte för att köpa flera Kinect-kameror, vilket vore en tänkbar lösning. Även laser-sensorer uteslöts eftersom priset på dem var över budgeten för projektet. De återstående sensorförslagen var då IR-sensorn och ultraljudssensorn. IR-sensorn och ultraljudssensorn är båda ganska billiga och lätta att implementera. Dock är IR-sensorn begränsad i att den endast ser en punkt och inte har någon direkt spridning att tala om och det skulle i så falla krävas fler IR-sensorer än ultraljudssensorer för att täcka in de viktigaste områdena. Resultatet av lösningsframtagningen blev således att *bug*-algoritmen valdes i kombination med ultraljudssensorer.

Ett fåtal av de valda ultraljudssensorerna köptes in för att genom tester bekräfta att de egenskaper som fanns specificerade på ultraljudens datablad stämde. Resultatet av testerna visade att spridningsvinkeln för ultraljudssensorerna var ungefär 55 grader på avstånd upp till en meter, vilket ansågs vara tillräckligt bra för detta ändamål. Resultatet av testerna användes sedan för att bestämma en konfiguration för samtliga ultraljud. Resultatet från konfigurationen visas i figur 17.



Figur 17: Varje cirkelsektor representerar en ultraljudssensors area. De rödmarkerade områdena är de kritiska områdena där AGVn behöver använda någon form av objektsundvikning. De grönmarkerade områdena används för att bestämma vilket håll AGVn bör åka.

Den valda konfigurationen krävde även att hållare för ultraljuden konstruerades. Resultatet av konstruktionen visas i figur 18.



Figur 18: Hållare för ultraljudssensorer i två millimeter tjock aluminiumplåt.

Vid konstruktionen av den färdiga lösningen för kollisionsundvikning upptäcktes det att ett par ultraljudsmoduler var defekta. Detta gjorde att *Bug*-algoritmen var tvungen att förenklas då de sensorer som var tänkta att användas för detta ändamål ej fanns tillgängliga.

Det kompletta kollisionsundvikningssystemet testades och resultatet från testerna visade att systemet fungerade. AGVn kolliderade inte med omgivning eller människor, i vissa fall var undvikningsmanövern mer tidskrävande än vad som ansågs önskvärt.

5.7 Integration med tillverkningslinan

På grund av förseningar i arbetet med PPU-labbet har inga fullskaliga tester kunnat utföras innan denna rapport skrivs. Dockstationen har dock kunnat monteras och AGVns uppgifter har kunnat testköras utan att några problem har identifierats. Kandidatgruppen som programmerar industrirobotar var nöjda med precisionen i materialleveransen.

6 Diskussion

AGVn kan leverera delar till tillverkningsprocessen med de precisionskrav som ställs på den och projektets syfte anses därför vara uppfyllt. Dockstationen fungerar bättre än vad någon i gruppen hade väntat sig. AGVn har aldrig fastnat i dockstationen och precisionen är mycket god. Faktum är att gruppen anser den vara onödigt god, industrirobotarna borde utrustats med verktyg som kan lokalisera var AGVn står så att precisionen från laserpositioneringssystemet hade räckt. Lösningen med en dockstation är inte flexibel eftersom delarna bara kan plockas upp på ett ställe och dockstationen tar upp en hel del golvyta i tillverkningscellen. Eftersom tillverkningscellen dessutom har en fixturanordning som ska ge rätt precision vid monteringen av delarna är det orimligt och onödigt att kräva så hög precision av AGVn som det gjordes.

LabVIEW fungerar väl för programmering av styrning men arbetet hade kunnat göras lättare om LabVIEW 2010 eller senare version användes istället för 2009. Senare versioner innehåller färdiga subrutiner för många av AGVns funktioner och FPGA hade inte behövt användas. Programmeringen försvårades av att den gång som AGVn måste åka genom i tillverkningscellen är mycket trång (fem centimeter på varje sida). Dessutom är utrymmet framför tillverkningscellen där AGVn ska ställa sig i rätt vinkel för att åka in i cellen precis bredvid en vägg vilket gör det svårt för AGVn att ställa sig rätt utan att kollidera med omgivningen.

Programmeringen av Kinectkameran underlättades avsevärt genom att köpa in en laptop istället för att som förra årets kandidatarbete använda en beagleboard (Linuxdator med låg beräkningskapacitet). Skräpinsamlingen är endast ett sidouppdrag vilket har medfört att det inte har prioriterats. Tidsbrist gjorde därför att ingen burksamlarmodul konstruerades. Att avgränsa skräpinsamling till identifiering av röda läskburkar förenklar programmeringen till en rimlig nivå. Programmering av en Kinectkamera för bildanalys kan annars göras hur avancerat som helst och skulle kunna vara ett eget kandidatarbete. Att AGVn hittar röda läskburkar och kan navigera till dem anses av gruppen som tillräckligt avancerat för att vara ett rimligt sidouppdrag. De avrundningsfel som uppstår på grund av approximationen i programmeringen som antar att alla trianglar är rätvinkliga är så liten att den inte tycks påverka AGVns förmåga att navigera till burkarna.

Användandet av en algoritm för ruttplanering ökar AGVns flexibilitet på grund av det grafiska gränssnitt som har skapats. Gränssnittet gör att AGVns karta kan uppdateras så att AGVn hittar snabbaste vägen även efter en ommöblering i PPU-labbet. Att inte ha någon typ av ruttplaneringsalgoritm gör att AGVn bara kan färdas i raka linjer från punkt A till B och svänga endast med hjälp av ultraljudssensorer. Sådana manövrar tar betydligt längre tid än att navigera efter en uträknad optimal färdväg. Utan en ruttplaneringsalgoritm kan inte AGVn anses vara helt autonom.

Ultraljudssensorer för kollisionsundvikningen var ett gott val och de har varit mycket stabila. Den algoritm som användes var en typ av *bug*-algoritm vilken visade sig fungera relativt väl. Den är ineffektiv i mycket trånga utrymmen och när AGVn befinner sig inne i tillverkningscellen måste därför sensorerna vara helt avstängda. När AGVn identifierar ett hinder avvaktar den ett tag och tittar om hindret har försvunnit. Eftersom det vanligaste hindret är människor räcker det oftast att vänta några sekunder tills människan försvunnit vilket gör att inga onödiga undanmanövrar behöver utföras.

Projektets experimentella natur gör att testkörningar av lösningarna är mycket viktiga. Tyvärr fanns inte möjlighet att testköra inne i PPU-labbet fören den fjärde maj (en och en halv vecka innan deadline för denna rapport) eftersom PPU-labbet inte var färdigställt. Arbetet tvingades därför att anpassas till förseningarna vilket medförde viss frustration. Hade PPU-labbet varit färdigställt i början av projektet skulle arbetet ha gått smidigare. Framförallt är det problemen med OPC-server och nätverk som försvårar arbetet.

Att vara flera kandidatgrupper som arbetar mot delvis samma mål skapar många situationer där grupperna får anpassa sig efter varandra. Det försvårar arbetet och en del problem har uppstått på grund av detta. Tydligare kommunikation mellan grupperna behövs och skulle kunna lösas genom tydligare direktiv och mer inblandning i arbetet av handledare.

7 Slutsatser

Resultatet visar att en AGV är lämplig för att leverera råmaterial till en tillverkningscell. Trots mycket höga krav på god precision och trånga utrymmen för AGVn att navigera genom utför den sina uppgifter med gott resultat. Eftersom tillverkningscellen innehåller industrirobotar är det en farlig miljö för människor och robotarna måste stoppas om människor ska beträda cellen. Användandet av en AGV kan därför öka både säkerhet och produktivitet. Följande slutsatser kan dras av de olika delarna av projektet:

- En mekanisk lösning för att säkerställa repeternoggrannheten är en robust metod som har visat sig fungera mycket väl tillsammans med AGVns OmniWheels.
- Programmering i LabVIEW för navigering är effektivt och fungerar väl ihop med Sick NAVen.
- Kinectkameran kan användas för burkidentifiering med god precision. Att avgränsa sidouppdraget till att leta efter just röda läskburkar förenklade programmeringen avsevärt.
- Ultraljudssensorer fungerar väl för att undvika kollision. Att AGVns första reaktion är att vänta några sekunder om ett hinder upptäcks är effektivt eftersom hindret oftast är en förbipasserande människa.
- A*-algoritmen för ruttplanering är mycket beprövad och har gett goda resultat. I PPU-labbet är ytorna mycket öppna vilket gör att den i dagsläget inte är helt nödvändig. Den ökar dock flexibiliteten genom att AGVn kan anpassas för ommöbleringar av PPU-labbet.
- I skrivande stund har inte kommunikationen kunnat färdigställas vilket gör att några fullständiga slutsatser inte kan dras. Det verkar dock som att en OPC-lösning är att föredra jämfört med en TCP/IP-socket.

Sammantaget är en AGV ett säkert, robust och precist alternativ för materialleverans i en tillverkningsmiljö. Den AGV som har färdigställts kan användas i framtida utbildningssammanhang på Chalmers.

8 Källförteckning

1 Savant Automation. History of Automatic Guided Vehicle Systems. 2011[hämtat: 2012-04-23]. Tillgänglig: http://www.agvsystems.com/res/history.htm

2 Meyer, Tim. AGVs: The Future Is Now. 2009 [hämtat: 2012-04-23]. Tillgänglig: http://mhlnews.com/powered-vehicles/news/agvs-future-now-5422/

3 Society of Robots. OMNI-WHEEL ROBOT – FUZZY. Publikationsdatum ej angivet [hämtat 2012-05-12] Tillgänglig: http://www.societyofrobots.com/robot_omni_wheel.shtml#omniwheel

4 Vexmen. What's your robot drive style? Publikationsdatum ej angivet [hämtat 2012-02-29] Tillgänglig: http://www.vexmen.com/2011/11/whats-your-robot-drive-style/

5 Nasir, Nurulhuda Binti Muhamad. Mechanical design of an Automated Guided Vehicle (AGV). Malaysia. Kandidatarbete vid National Technical University of Malaysia. 2006.

6 Pissarenko, Dimitri. Eigenface-based facial recognition. 2003 [hämtat 2012-04-26] Tillgänglig: http://openbio.sourceforge.net/resources/eigenfaces/eigenfaces-html/facesOptions.html

7 Turk, Matthew och Pentland, Alex. Eigenfaces for Recognition, Journal of Cognitive Neuroscience, 3 (1), 1991 [hämtat 2012-04-26] Tillgänglig: http://www.cs.ucsb.edu/~mturk/Papers/jcn.pdf

8 OPCFoundation, What is OPC? 2012 [hämtat 2012-05-12] Tillgänglig: http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC

9 Hart, Peter, Nilsson, Nils & Raphael, Bertram, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, 1968, IEEE Transactions on Systems Science and Cybernetics, 4(2): 100-107

10 Viagger, Mikkel. Analysis of Kinect for mobile robots. 2011 [hämtat 2012-05-12] Tillgänglig: http://www.scribd.com/doc/56470872/Analysis-of-Kinect-for-Mobile-Robots-unofficial-Kinect-data-sheet-on-page-27

11 Lumelsky, Vladimir och Stepanov, Alexander. Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles or Arbitrary Shape. 1985. *Algorithmica* (1987) s. 403-430.

12 Ribeiro, Maria Isabel. Obstacle Avoidance. 2005. Instituto de Sistemas e Robótica, Instituto Superio Técnico. [hämtat 2012-05-04] Tillgänglig: http://users.isr.ist.utl.pt/~mir/pub/ObstacleAvoidance.pdf

13 Borenstein, Johann och Koren , Yoram. The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots. 1991. *IEEE Transactions on Robotics and Automation,* vol. 7, nr. 3, s. 278-288.

14 Andersson Viktor, Ewnert Carl, Li Dan, Stenback Daniel. Programmering av AGV för skräpuppsamling och transport av däck. Göteborg. Kandidatarbete vid Chalmers Tekniska Högskola. 2011.

15 Texas Instruments. Produktbeskrivning av Stellaris(MDL-BDC24) motorkontroller. 2012 [hämtat 2012-05-14] Tillgänglig: http://www.ti.com/tool/mdl-bdc24&DCMP=STELLARIS&

16 AndyMark. Produktbeskrivning av likströmsmotorer AM802-001A. 2012 [hämtat 2012-05-14] Tillgänglig: http://www.andymark.com/product-p/am-0255.htm

17 National Instruments. Produktbeskrivning av CompactRIO (cRIO-9074). 2011 [hämtat: 2012-05-14] Tillgänglig: http://sine.ni.com/ds/app/doc/p/id/ds-204/lang/en

18 Asus. Produktbeskrivning av Eee PC 1011PX. Publikationsdatum ej angivet [hämtat: 2012-05-14] Tillgänglig: http://www.asus.se/Eee/Eee_PC/Eee_PC_1011PX/#specifications

19 SICK. Produktbeskrivning av NAV200. 2012 [hämtat: 2012-05-14] Tillgänglig: https://www.mysick.com/eCat.aspx?go=FinderSearch&Cat=Row&At=Fa&Cult=English&FamilyID=344&Category=Produktfinder&Selections=34430,34243

20 Cisko. Produktbeskrivning av Linksys WGA600N. Publikationsdatum ej angivet [hämtat:2012-05-14] Tillgänglig: http://homesupport.cisco.com/en-us/wireless/lbc/WGA600N

21 Skansholm, Jan. Skarp programmering med C#. Göteborg. Studentlitteratur. 2008.

9 Bilagor

Bilaga 1 - Övergripande kravspecifikation

En kravspecifikation för hela AGVn skapades vid projektets start och den ses i tabell 3.

Intressent	Kriterier	Kontrollmetod	Målvärde	Krav/Önskemål
Produktion	Konstruktionskostnad	Bokföring	<10000 kr	K
Användare	Navigera från A-B	Testkörning	positionering +-5 cm, vinkel +- 2 grader	K
Användare	Konsekvent slutposition	Testkörning	Repeternoggrannhet på 1mm i x, y och z-led	K
Användare	Framkomlighet	Testkörning	Forcera ojämnheter i golv i PPU-labb	K
Service	Utbytbara moduler	Analys av konstruktion	samtliga moduler demonterbara	K
Användare	Form	Analys av konstruktion	Ha form av ett rätblock	K
Användare	Undvika kollision	Test	Ej krocka med fasta eller rörliga objekt	K
Användare	Användarvänlighet	Test	Försedd med dokumentation	K
Användare	Samarbeta med andra maskiner	Test och gemensam utvärdering	Kunna kommunicera med övrig utrustning	K
Användare	Vara försedd med fixtur	Analys av konstruktion	Fixtur som garanterar tillräcklig noggrannhet för plockrobot	K
Produktion	Miljövänlighet	Analys av konstruktion	I så stor utsträckning som möjligt bestå av återvinningsbart material	K
Användare	Tåla last	Analys av konstruktion	Kunna lastas med 25 kg	K
Användare	Identifiera skräp	Testkörning	Kunna identifiera skräp av någon typ	Ö4
Användare	Plocka skräp	Testkörning och analys av konstruktion	Kunna plocka det skräp som identifierats	Ö4
Användare	Vikt	Analys av konstruktion	vikt <50 kg	Ö3
Användare	Smidig laddning	Test	Laddstation som kräver mindre än 1 minuts arbete för att starta laddning	Ö3
Användare	Utseende	Bedömning		Ö1
Användare	Hastighet	Testkörning	Inte en flaskhals i cellen	Ö1

Tabell 3: Kravspecifikation för hela AGVn

Bilaga 2 – Funktionsstruktur

En uppdelning av projektet i mindre delproblem resulterade i det funktionsträd som ses i figur 19.

Figur 19: Funktionsstruktur för AGVn

Bilaga 3 - Lösningssökning för framdrivning

Efter att huvudproblemet för framdrivningen identifierats som för hög friktion mellan hjul och golv för att kunna svänga inleddes en lösningssökning tillsammans med AGV2. En specifik kravspecifikation (tabell 4) för framdrivningen skapades genom brainstorming, informationssökning på internet och studier av äldre kandidatprojekt.

Område	Kriterienr.	Kriterier	Kontrollmetod	Målvärde	Krav/Önskemål
Prestanda	1.1	Ta sig över springor i golvet	Test	förändra riktning så lite som möjligt	K
	1.2	Lämna delar på specifik position upprepade gånger	Test	Repeternoggranhet ± 1 mm	K
	1.3	Kunna specifikt avgöra kring vilken punkt roboten svänger	Test	Tolerans ± 2 cm i x,y och z-riktning	K
	1.4	Kunna leverera en full uppsättning av delar/hämta färdig bil	Test		K
	1.5	Kunna köra i gånghastighet	Mått	Hastighet ≥ 2 km/h	K
	1.6	Kunna bära angiven last	Test	10 kg	K
	1.7	Kunna docka i flera stationer	Test	2 stationer i ”rännan”	Ö
	1.8	Svängradie	Test/Beräkning	Rotera kring en punkt, radie 0	Ö
	1.9	Position på svängpunkt	Test/Beräkning	Rotera kring så central punkt som möjligt	Ö
	1.10	Stabilitet	Test	Skall ej tippa	Ö
	1.11	Manövrerbarhet	Test	Så bra som möjligt	Ö
Produktion	2.1	Återanvända material	Konstruktionsanalys	Använda så mycket befintligt material som möjligt	Ö
	2.2	Tillverkningskostnad	Konstruktionsanalys	Så billig som möjligt	Ö
	2.3	Mekanisk Komplexitet	Konstruktionsanalys	Så enkel som möjligt	Ö
Dimension	3.1	Penetrera dörr och säkerhetsutrustning	Mått	Bredd < 830 mm	K
	3.2	2 sammanhängande ytor som bil och material får plats på	Mått	Flak-yta bredd ≥ 200 mm, längd ≥ 500 mm	K
	3.3	Sick NAV200 ska blockeras så lite som möjligt	Test	Aldrig tappa bort sig på grund av blockering	Ö
Användning	4.1	Programmeringskomplexitet		Så enkel som möjligt	Ö

Tabell 4: Kravspecifikation för framdrivningen

Önskemålen viktades därefter mot varandra enligt tabell 5 där 1 betyder prioriterat önskemål, 0 nedprioriterat önskemål och 0.5 likvärdigt önskemål.

Önskemål	1.7	1.8	1.9	1.10	1.11	2.1	2.2	2.3	3.3	4.1	Resultat
1.7		0	0	0	0	0	0,5	0	0	0	0,5
1.8	1		0	0	0	1	1	1	0	1	5
1.9	1	1		1	1	1	1	1	0	1	8
1.10	1	1	0		0,5	1	1	1	0	1	6,5
1.11	1	1	0	0,5		1	1	1	0	1	6,5
2.1	1	0	0	0	0		1	1	0	0	3
2.2	0,5	0	0	0	0	0		0	0	0	0,5
2.3	1	0	0	0	0	0	1		0	0	2
3.3	1	1	1	1	1	1	1	1		1	9
4.1	1	0	0	0	0	1	1	1	0		4

Tabell 5: Viktningsmatris av önskemålen

Tabell 5 visade alltså att önskemålen har följande inbördes rangordning(se tabell 6)

Önskemål	**Resultat från tabell 4**
3.3	9
1.9	8
1.10,1.11	6,5
1.8	5
4.1	4
2.1	3
2.3	2
1.7,2.2	0,5

Tabell 6: Önskemålens ordnade efter hur viktiga de ansågs vara

För att bestämma viktningsindexet av varje önskemål jämfördes de på en skala från 1-20 (se tabell 7) i den ordning som togs fram i tabell 4. Indexet för varje önskemål fås genom att dividera skalvärdet för önskemålet med totalsumman av alla skalvärden.

Skala	Önskemål	Viktningsindex
20	3.3	0,179
19	1.9	0,170
18		
17		
16	1.11	0,143
15	1.10	0,134
14		
13		
12	1.8	0,107
11		
10	4.1	0,089
9		
8	2.1	0,071
7	2.3	0,063
6		
5		
4		
3	2.2	0,027
2	1.7	0,018
1		

Tabell 7: De olika önskemålens viktningsindex

Efter viktningen av önskemålen gjordes en funktionsanalys och lösningsframtagning påbörjades genom brainstorming. Tabell 8 visar de olika lösningsförslagen.

Delfunktion					
Stödjande av AGV	Svivelhjul	Omniwheels	Vanliga hjul	Larvfötter	
Drivande av AGV	Omniwheels	Vanliga hjul	Larvfötter		
Hjulkonfiguration	Holonomic	Rektangulär (4 st)	Diamant	Rektangulär (6 st)	
Drivningskonfiguration	framhjulsdrift	bakhjulsdrift	mitthjulsdrift	fyrhjulsdrift	
Styrning	Vridbara hjul	Bandvagnsprincip			
Transport av delar	På tak i fixtur	Rack på vagn	Separat flak med fixtur		
Bra precision vid materialavlämning	Dockstation	Magneter	Sick NAV	Räls	Tejp

Tabell 8: Delfunktioner och dess lösningsförslag.

Nedan följer ett förtydligande av de olika lösningsförslagen:

- **Svivelhjul**: Är hjul som kan vridas beroende på underlag och hur AGVn färdas. Sitter på kundvagnar.
- **Omniwheels:** Hjul med rullar på rotationsbanan som gör att de lättare kan glida i sidled för att underlätta vid svängar.
- **Larvfötter**: Istället för hjul används larvband likt en pansarvagn.
- **Holonomic:** 4 hjul används med en placering som syns som bild H i figur 20.
- **Rektangulärt:** 4 eller 6 hjul placerade i drivriktningen likt en bil fast utan att hjulen kan svänga.
- **Diamant:** 2 drivande hjul i mitten av AGVn med två stödhjul i kanterna likt bild E i figur 20.
- **Vridbara hjul:** Hjul som kan svängas likt en bil.
- **Bandvagnsprincip:** Hjul eller larvfötter körs åt olika håll så att AGVn roterar utan att åka framåt eller bakåt.

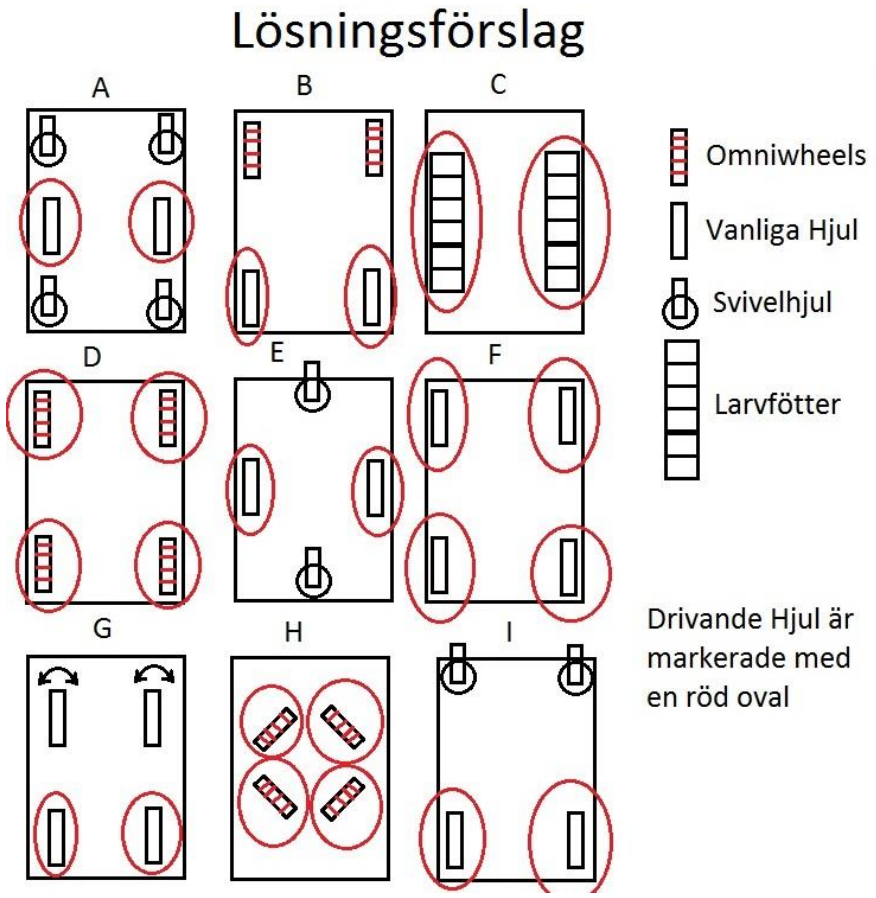
Totallösningsförslag utformades ur givna lösningsförslag för de första 5 delfunktionerna och presenteras i tabell 9.

Totallösning	**A**	**B**	**C**	**D**	**E**
Stödjande av AGV	Svivelhjul	Omniwheels	Larvfötter	Omniwheels	Svivelhjul
Drivande av AGV	Vanliga hjul	Vanliga hjul	Larvfötter	Omniwheels	Vanliga hjul
Hjulkonfiguration	Rektangulärt (6 st)	Rektangulärt (4st)	Rektangulärt (4st)	Rektangulärt (4st)	Diamant
Drivningskonfiguration	Mitthjulsdrift	Framhjulsdrift	Framhjulsdrift	Fyrhjulsdrift	Mitthjulsdrift
Styrning	Bandvagn	Bandvagn	Bandvagn	Bandvagn	Bandvagn

Totallösning	**F**	**G**	**H**	**I**
Stödjande av AGV	Vanliga hjul	Vanliga hjul	Omniwheels	Svivelhjul
Drivande av AGV	Vanliga hjul	Vanliga hjul	Omniwheels	Vanliga hjul
Hjulkonfiguration	Rektangulärt (4st)	Rektangulärt (4st)	Holonomic	Rektangulärt (4st)
Drivningskonfiguration	Fyrhjulsdrift	Framhjulsdrift	Fyrhjulsdrift	Framhjulsdrift
Styrning	Bandvagn	Vridbara Hjul	Bandvagn	Bandvagn

Tabell 9: Totallösningar till framdrivningen

I figur 20 visas en principiell sketch av de olika totallösningsförslagen.



Figur 20: Principiell beskrivning av totallösningsförslag till framdrivningen.

De olika lösningsförslagen utvärderades mot kravspecifikationen för att fastställa att samtliga krav blev uppfyllda. Resultat visas i tabell 10.

	1.1	1.2	1.3	1.4	1.5	1.6	3.1	3.2	Behåll?
A	x	x	x	x	x	x	x	x	JA
B	x	x	x	x	x	x	x	x	JA
C	x	x	-						NEJ
D	x	x	x	x	x	x	x	x	JA
E	x	x	x	x	x	x	x	x	JA
F	x	x	-						NEJ
G	x	x	-						NEJ
H	x	x	x	x	x	x	x	x	JA
I	x	x	x	x	x	x	x	x	JA

Tabell 10: Kontrollerar om totallösningsförslagen uppfyller kraven.

Vinnande lösningsförslag

De totallösningar som uppfyllde alla krav jämfördes mot varandra genom att uppskatta hur väl de uppfyller de olika önskemålen vilket visas i tabell 11.

	Totallösning:		A		B		D		E		H		I	
Kravnr:	**Önskemål:**	**Viktfaktor**	**P**	**V**	**P**	**V**	**P**	**V**	**P**	**V**	**P**	**V**	**P**	**V**
1.7	Kunna docka i flera stationer	0,018	4	0,072	4	0,072	5	0,09	4	0,072	3	0,054	4	0,072
1.8	Svängradie	0,107	5	0,535	5	0,535	5	0,535	5	0,535	5	0,535	5	0,535
1.9	Position på svängpunkt	0,117	5	0,585	1	0,117	5	0,585	5	0,585	5	0,585	1	0,117
2.1	Återanvända material	0,071	3	0,213	5	0,355	3	0,213	3	0,213	1	0,071	5	0,355
2.2	Tillverkningskostnad	0,027	2	0,054	4	0,108	1	0,027	2	0,054	1	0,027	4	0,108
3.3	Sick NAV ska blockeras så lite som möjligt	0,179	3	0,537	3	0,537	3	0,537	3	0,537	3	0,537	3	0,537
1.10	Stabilitet	0,134	5	0,67	5	0,67	4	0,536	3	0,402	4	0,536	5	0,67
2.3	Mekanisk komplexitet	0,063	2	0,126	4	0,252	2	0,126	2	0,126	1	0,063	5	0,315
4.1	Programmeringskomplexitet	0,089	5	0,445	3	0,267	4	0,356	5	0,445	1	0,089	3	0,267
1.11	Manövrerbarhet	0,143	4	0,572	2	0,286	5	0,715	4	0,572	5	0,715	2	0,286
	Resultat:			3,809		3,199		3,72		3,541		3,212		3,262

Tabell 11: Den vinnande totallösningen fås fram genom att utvärdera hur bra totallösningen uppfyller önskemålen. P = poäng (Hur väl lösningsförslaget uppfyller målet), V = värde efter multiplikation med viktfaktor.

Resultatet av lösningssökningen blev alltså att de två mest optimala lösningsförslagen i teorin var förslag A och D. De båda AGV-grupperna valde därför att utveckla var sin totallösning för att utvärdera vilken som fungerar bäst.

Bilaga 4 - Dockstationen

Industriroboten som ska plocka delar från AGVn har inte möjlighet att mäta in delarnas position vilket kräver att AGVn lämnar delar på i princip exakt samma plats varje gång. Efter litteraturstudier konstaterades att Sick Naven inte har tillräcklig precision för detta. Tejp och räls som lösningsförslag ansågs av handledare vara alltför enkla lösningar vilket gjorde att en typ av dockstation ansågs nödvändig.

Kravspecifikation

För att dockstationen ska uppfylla de krav som ställs på den gjordes en kravspecifikation (se tabell 12) genom brainstorming och testkörning av befintlig AGV.

Kravspec Dockstation				
Intressent	**Kriterier**	**Kontrollmetod**	**Målvärde**	**Krav/Önskemål**
Produktion	Konstruktionskostnad	Bokföring	<2000 kr	Ö
Produktion	Tillverkningsbarhet	Analys av konstruktion	Kunna tillverkas i prototypverkstaden	K
Produktion	Tillverkningstid	Analys av konstruktion	Kunna färdigställas innan projektets slut	K
Produktion	Tillverkningstid	Analys av konstruktion	Så kort tid som möjligt	Ö
Användare	Samarbeta med andra maskiner	Test och gemensam utvärdering	Kunna kommunicera med övrig utrustning	K
Användare	Repeternoggranhet	Beräkning, uppskattning, test	+- 0.5 mm / +- 0,5 grad	K
Användare	Utrymmesbehov	Analys av konstruktion	Få plats i cellen och på AGV	K

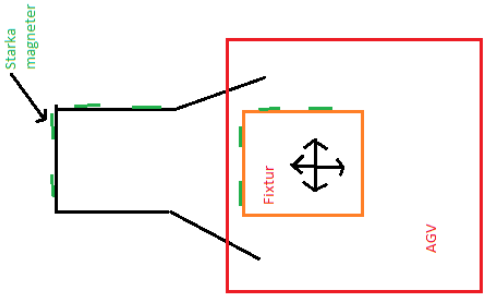
Tabell 12: Kravspecifikation för dockstation

Lösningsförslag

Lösningssökning med brainstorming, eftersökningar på internet och testkörning av befintlig AGV resulterade i följande tänkbara lösningar:

Docka – magnet

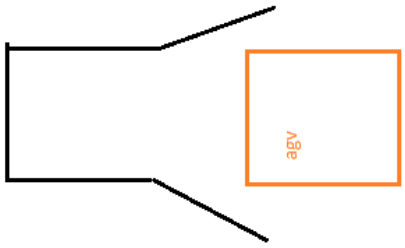
I detta lösningsförslag är en fixtur som håller bildelarna rörlig på AGVn. När AGVn kör till sin givna position kompenserar den rörliga fixturen för felpositioneringen hos AGVn. Magneter i dockstationen drar fixturen till precis samma plats varje gång, en sketch av lösningsförslaget ses i figur 21.



Figur 21: Docka - Magnet

Docka – AGV pressar sig

Denna lösning går ut på att AGVn styrs rätt i en dockstation genom att dess sidor glider mot skenor. Alternativt kan hjul montereras horisontellt på AGV:s sidor så att hjulen rullar mot dockstationen istället för att AGVn glider. En sketch av lösnignsförslaget ses i figur 22.



Figur 22: Docka - AGV pressar sig in

Docka med pneumatiska cylindrar

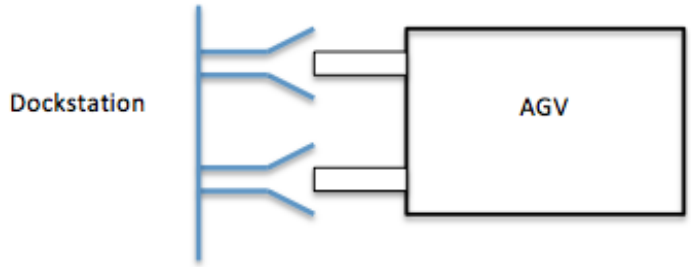
En dockstation med pneumatiska cylindrar som låser fast fixturen i rätt läge varje gång. Denna lösning är troligtvis mycket noggrann och robust men arbetet med att bygga den är omfattande.

Ingen docka (Robot mäter in fixtur)

Om industriroboten försågs med mätverktyg som mäter in fixturens position och därav kompenserar för AGVns felpositionering behövs ingen dockstation och repeternoggrannheten blir mycket god.

Docka – piggar i AGV:s front

Om dockstationen förses med koniska hål och AGVn med cylindriska stänger i framkanten skulle AGVn styras rätt av konorna och därav hamna i samma läge varje gång. Dockan skulle kunna kompletteras med styrande skenor som styr AGVn som tidigare lösningsförslag. En sketch av lösningsförslaget ses i figur 23.



Figur 23: Docka – piggar i AGV:s front

Vinnande lösningsförslag

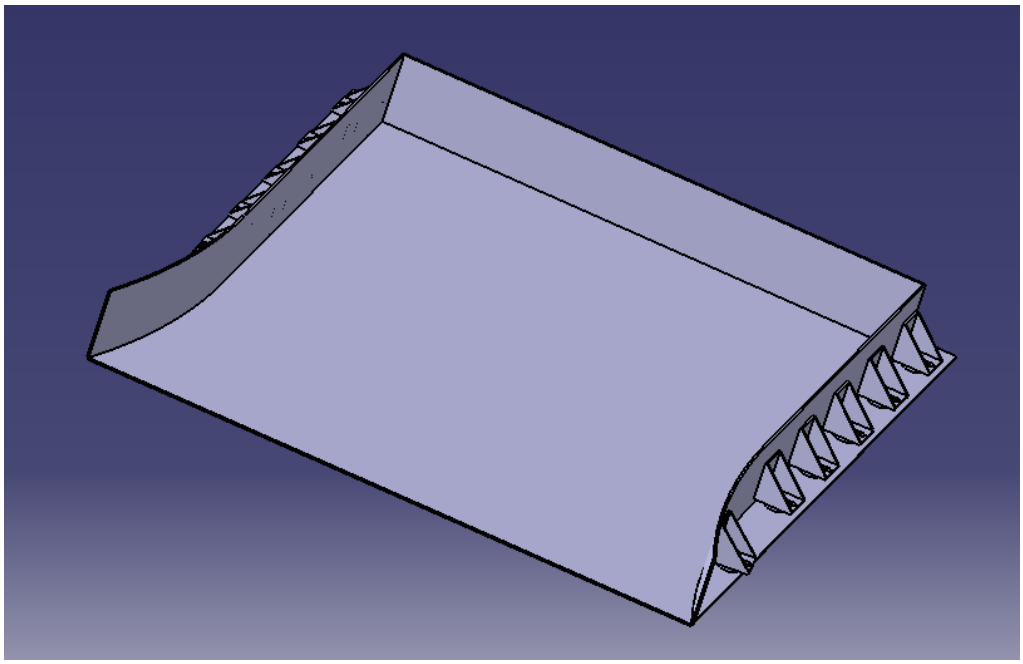
För att hitta den bästa lösningen sållades först de lösningsförslag som inte ansågs uppfylla kraven i kravspecifikationen bort:

- Docka med pneumatiska cylindrar föll på kravet på att kunna tillverkas innan projektets slut.
- Ingen docka (robot mäter in fixtur) kunde inte användas eftersom robotgruppen inte kunde åstadkomma detta.

De lösningsförslag som uppfyllde kraven var alltså Docka – magnet, Docka – AGV pressar sig, Docka – Piggar i AGVns front. För att hitta den bästa av de återstående lösningarna jämfördes de mot varandra efter hur väl de uppfyllde önskemålen i kravspecifikationen. Samtliga uppskattades uppfylla kravet på en kostnad under 2000 kr, däremot skiljde de sig åt i omfattning av tidsåtgång. Särskilt Docka – magnet ansågs mycket tidskrävande jämfört med de andra eftersom en rörlig fixtur behövde konstrueras och tillverkas. Tidsåtgången mellan Docka – AGV pressar sig och Docka – Piggar i AGVns front ansågs relativt likvärdig. Valet mellan de båda var alltså inte självklart men slutligen valdes Docka – AGV pressar sig. Framförallt eftersom den ansågs relativt enkel att tillverka och uppfattades som något mer robust än lösningen med piggar i AGVns front. Samma lösning används i industrin för så kallade smalgångstruckar.

CAD

Dockstationen konstruerades I CATIA V5R19 och resultatet visas i figur 24. Den tillverkades därefter i prototyplaboratoriet på Chalmers av tre millimeter tjock stålplåt och resultatet syns i figur 6 ovan.



Figur 24: Produktutkast på dockningsstationen.

Bilaga 5 – LabVIEWprogram

Programnamn	Beskrivning
MainPLC	Huvudprogram, tar emot och skickar information till PLCn
DriveToPos	Kör till förutbestämda punkter, med kollisionsundvikning
MainDock	Kör in AGVn i Dockan och skickar en signal när den är färdig
MainUndock	Kör ur AGVn ur dockan och skickar en signal när den är färdig
TurnSub	Beräknar hastighet för AGVn när den ska svänga
CheckSideCollision	Kontrollerar att det är fritt från hinder åt det håll AGVn ska svänga
DriveTurnRPM	Sätter svänghastighet och svängrikting samt skickar detta till motorerna
SubStraight	Beräknar hastighet för AGVn när den ska åka rakt fram till en punkt
CheckFrontCollision	Kontrollerar att det är fritt från hinder rakt fram
GetDirection	Väljer åt vilket håll AGVn ska köra runt ett hinder
DriveAndTurnRPM	Sätter framhastighet och svänghastighet åt de båda motorerna
SubDrivePast	Som SubStraight, fast den kör förbi en punkt
GetUltraljudDistance	Plockar ut avstånden till hinder ifrån en lista med ultraljudssensorer
XYToAngle	Beräknar vinkel samt avstånd mellan 2 X,Y-koordinater
AngleDistanceToXY	Beräknar ny koordinat utifrån nuvarande position, vinkel samt avstånd
DeleteElementArray	Tar bort tillagda punkter som AGVn inte längre kan åka till
WriteAndReadLaser	Skickar samt tar emot ett kommando från Sick NAVen
InitiateSickPosition	Skickar signaler till Sick NAVen så att positionsläge initieras
GetLaserPosition	Får den nuvarande positionen från Sick NAVen
StringToPosition	Gör om informationen från Sick NAVen till verkliga, decimala värden
CanCollectionMain	Huvudprogram som styr burkinsamlingen
RandomWalk	Flyttar AGVn 1 meter åt ett slumpvis valt håll, används vid burkinsamling
LabVIEWKinect	Kommunicerar med laptoppen, får info om den senast hittade burken
TurnToCan	Svänger AGVn mot en burk
KinectAngleToXY	Beräknar burkens position utifrån info från Kinecten
astar	Skickar start- & målkoordinater till laptoppen och får tillbaka vägen
MasterKill	Kontrollerar en global variabel som kan stänga av alla while-loopar
ReadWriteClearSinglPort1	FPGA-program som skickar/läser/rensar port1mod2, där lasern sitter
UltraLjudDriver1.2	FPGA-program som skickar/ läser/behandlar info från ultraljudsenor
JoystickRTMain	Tar emot joystick-info från en dator och skickar dem till motorerna
JoystickOverRun	Bryter huvudprogrammet och går över till manuell styrning av AGVn
LazyDrive	Låter datorn som är uppkopplad mot CompactRIOn styra AGVn

Eftersom många av programmen används av andra program är det svårt att på ett snyggt sätt återge hierarkin. Generellt använder DriveToPos samt program som innehåller ”main” många av de andra subrutinerna. Så fort AGVn ska förflytta sig utanför cellen så används DriveToPos som också innehåller de program som har med kollisionsundvikning att göra. Då behöver AGVn även kommunicera med Sick NAVen samt de program som styr motorerna. De båda FPGA drivrutinerna har ett motsvarande Real Time program som fungerar som ett gränssnitt mellan de två tilläggen.