Designspecifikation

Redaktör: Sofie Dam

Version 0.2

GRUPPTRUCK

Status

Granskad	Dokumentansvarig	-
Godkänd		

PROJEKTIDENTITET

2017/HT, GruppTruck
Tekniska högskolan vid Linköpings universitet, ISY

Gruppdeltagare

Namn	**Ansvar**	**Telefon**	**E-post**
Gabriel Fredriksson	Projektledare	076-294 04 49	gabfr905@student.liu.se
Sofie Dam	Dokumentansvarig	070-422 32 57	sofda068@student.liu.se
Johannes Bodin	Designansvarig, Uppdragsansvarig Delområde 1 & 4	070-246 05 66	johbo346@student.liu.se
Daniel Nilsson	Mjukvaruansvarig	070-733 23 10	danni768@student.liu.se
Emil Relfsson	Testansvarig	070-635 08 37	emire260@student.liu.se
Max Antonsson	Uppdragsansvarig Delområde 2	070-781 77 75	maxan749@student.liu.se
Jasmina Hebib	Uppdragsansvarig Delområde 3	073-672 66 28	jashe481@student.liu.se

Kund: Toyota Material Handling Manufacturing Sweden AB, 595 81 Mjölby
Kursansvarig: Daniel Axehill, 013-28 40 42, daniel.axehill@liu.se
Handledare: Erik Hedberg, 013-28 13 38, erik.hedberg@liu.se
Beställare: Andreas Bergström, 010-71 15 45 4, andreas.bergstrom@liu.se

Innehåll

1 Inledning **6**

2 Översikt av system **6**

3 Delområde 1 - Truckbeskrivning i simuleringsmiljö **8**

3.1 Bakgrund . . . 8

3.2 Syfte . . . 8

3.3 Befintlig modell . . . 9

3.3.1 Drivhjulsmodellen *drive_wheel_assembly.xacro* . . . 9

3.4 Utökad modell . . . 10

3.4.1 Alternativ 1: Införande av ny link och joint . . . 10

3.4.2 Alternativ 2: Införa offset direkt i *wheel_drive_joint* . . . 10

3.4.3 Val av implementeringsalternativ . . . 11

3.5 Validering . . . 11

4 Delområde 2 - Tillståndsmodell **13**

4.1 Systembeskrivning . . . 13

4.1.1 Dynamiska Modeller . . . 14

4.1.2 Modellutvärdering . . . 17

4.2 Störningar . . . 17

4.2.1 Systemstörningar . . . 17

4.2.2 Mätstörningar . . . 18

5 Delområde 3 - Reglering av precisionsinkörning **20**

5.1 Bakgrund . . . 20

5.2 Systembeskrivning . . . 20

5.3 Implementera PID . . . 22

5.3.1 PID . . . 22

5.4 Implementera PID med framkoppling . . . 23

5.5 Implementera MPC . . . 23

5.5.1 MPC-teori . . . 23

5.6 Problemanalys . . . 25

6 Delområde 4 - Planering av precisionsinkörningskurva 26
6.1 Nuvarande lösning 26
6.1.1 Matematisk beskrivning av Bézier-kurva 26
6.1.2 Huvudfilen *move_precise.py* 27
6.2 Problematik 28
6.3 Lösning 29
6.3.1 Implementering 30

Referenser 31

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2017-10-03	Första utkastet	Alla	SD
0.2	2017-10-06	Andra utkast efter beställarens kommentarer.	Alla	SD

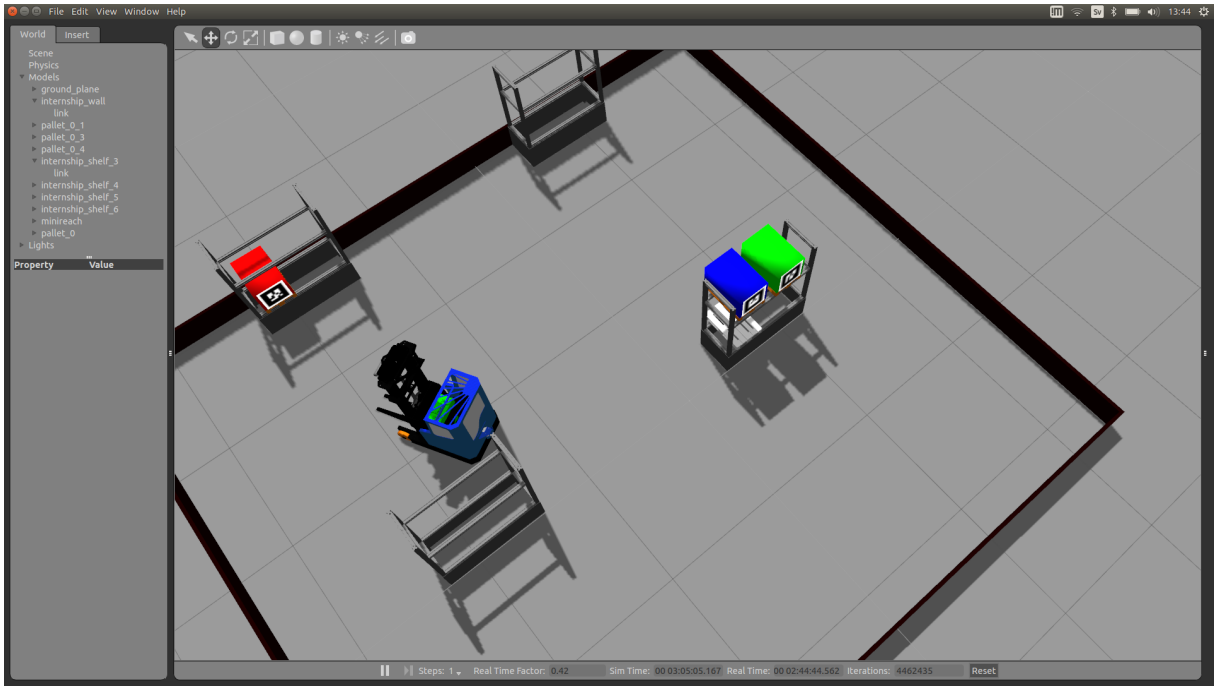
1 Inledning

I denna designspecifikation förklaras de tekniska detaljer som ska utvecklas under projektet mer på djupet. Designspecifikationen har som syfte att fungera som stöd under lösningen och implementeringen av de olika delområdena. Designspecifikationen strävar efter att ge en tillräcklig god analys av problemen så att en oinsatt person men som har de teknisk kunskaperna som krävs ska kunna ta över efter projektmedlemmarna om arbetet inte skulle färdigställas.

2 Översikt av system

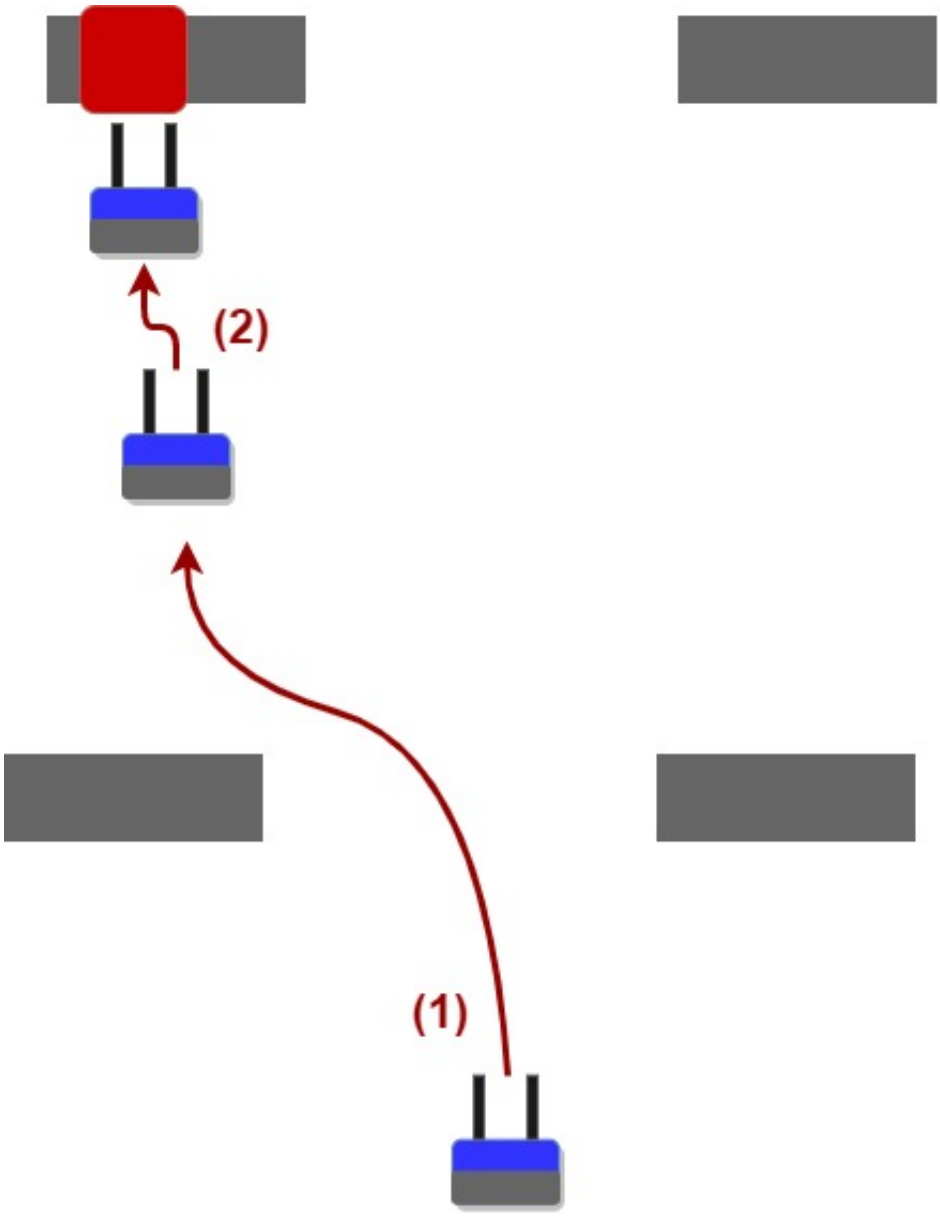
Systemet består av en eller flera minireach-gaffeltruckar samt en övergripande dator som fungerar som master. Själva gaffeltrucken delas upp i två delar: ett API som sköter manövrering av trucken samt dess delar och en abstrakt del som sköter mer avancerad funktionalitet så som planering och reglering. I vårt arbete med trucken kommer vi bara behandla den abstrakta delen av trucken.

I projektet kommer koden framförallt simuleras i simuleringsmiljön Gazebo. Men målet är också att koden fungerar bra på de verkliga truckarna. En bild över simuleringsmiljön då trucken är på väg att hämta en pall från en hylla visas i figur 1.



Figur 1: *Trucken påväg att hämta en pall i simuleringsmiljön.*

Truckens färdväg från en startposition någonstans i lagret till en hylla där en vara ska plockas upp kan delas upp i två steg, se figur 2. Det första är en lång bana som ska följas med en grov reglering till en ungefärlig position framför hyllan på ungefär 8 dm. Detta steg kommer inte utvecklas under projektet. Därefter inleds en noggrannare reglering fram till hyllan för att placera trucken precis framför pallen som ska plockas upp. Detta steg kallas precisionsinkörning och ska utvecklas med en MPC-regulator så trucken klarar svårare inkörningar till pallen.



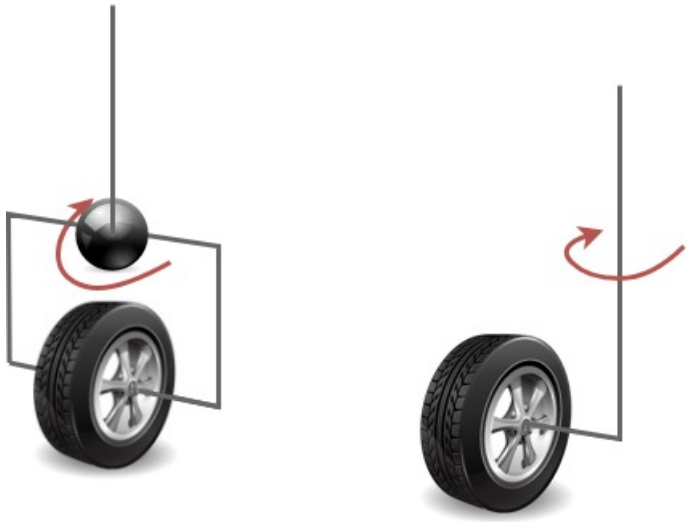
Figur 2: *Truckens förlopp vid körning fram till hylla. Först kör den från sin startposition till en ungefärlig position framför pallen med grov reglering. Därefter utförs en precisionsinkörning in framför hyllan där pallen är placerad med en noggrann reglering.*

3 Delområde 1 - Truckbeskrivning i simuleringsmiljö

3.1 Bakgrund

Utvecklingen av systemet sker främst mot en simuleringsmiljö i programmet Gazebo. Fördelen med att utveckla på detta sätt är framför allt att man inte behöver ha tillgång till de verkliga truckarna, vilket gör utvecklingen både säkrare ur ett personperspektiv och mindre tidskrävande för kunden Toyota Material Handling. Nackdelen är att simuleringsmiljön kräver en mycket precis modell av den verkliga trucken för att den mjukvara som utvecklas ska fungera i verkligheten. I dagsläget finns en någorlunda bra simuleringsmodell framtagen, men den skiljer sig lite mot hur de faktiska truckarna ser ut.

Det som framför allt skiljer modell mot verklighet är att modellen har drivhjulet centrerat rakt under dess rotationsaxel medan drivhjulet på de verkliga truckarna sitter med en liten offset. Se Figur 3. Ett sätt som detta modellfel visar sig på är när man roterar drivhjulet på en stillastående truck. I simuleringsmiljön står då trucken helt stilla under rotationen medan den i verkligheten vrider på sig lite fram och tillbaka.



Figur 3: *Den vänstra delen visar hur modellen är uppbyggd i dagsläget, och den högra hur de faktiska truckarna är konstruerade.*

3.2 Syfte

Syftet med detta delområde är att utveckla drivhjulskonfigurationen i den modell som används i simuleringsmiljön så att den simulerade trucken uppför sig mer likt den verkliga trucken.

3.3 Befintlig modell

Modellen som används i simuleringsmiljön definieras i ett antal s.k. URDF-filer (Unified Robot Description Format). Dessa filer är uppbyggda enligt XML-standard (eXtensible Markup Language) och specificerar framför allt robotens olika delar (*links*), hur delarna är kopplade till varandra (*joints*) och hur delarnas olika rörelser påverkas av aktuatorer (*transmissions*). Filerna har filändelse *.xacro*.

Den befintliga modellen är uppdelad i flertalet olika filer, som alla är kallade från en huvudfil. Nedan presenteras modellens viktigaste filer.

- *minireach.xacro*: Huvudfil. Definierar vissa links och joints såsom base_link (huvudkropp för roboten) samt kallar på de andra ingående filerna. Innehåller dessutom definitioner för olika parametrar och konstanter.
- *cab_assembly.xacro*: Beskriver uppbyggnaden av truckens hytt.
- *camera_assembly.xacro*: Specificerar hur kameran sitter monterad på trucken.
- *drive_wheel_assembly.xacro*: Beskriver drivhjulet och hur det kan röra sig relativt trucken.
- *stand_assembly.xacro*: Beskriver tornets uppbyggnad och rörelser.

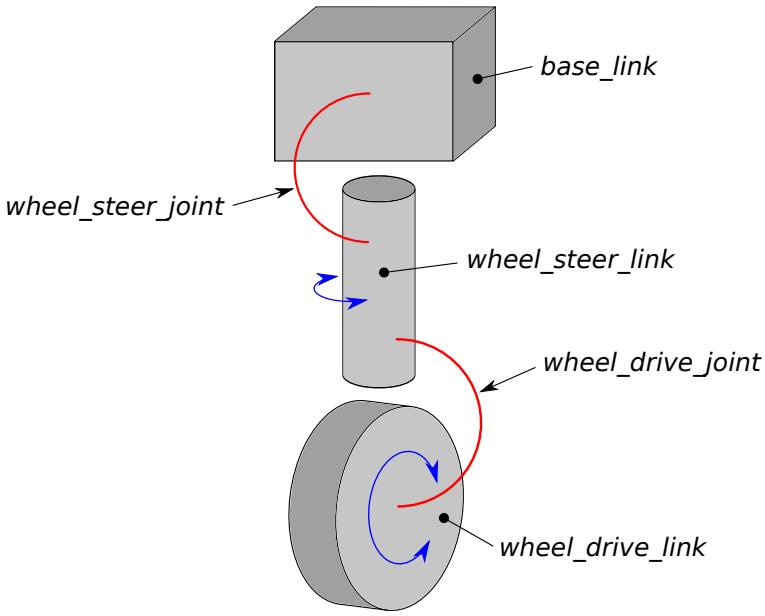
För att utöka modellen så att den även beskriver hjulets offset från styrningens rotationsaxel behöver ändringar göras framför allt i filen *drive_wheel_assembly.xacro*.

3.3.1 Drivhjulsmodellen *drive_wheel_assembly.xacro*

I filen *drive_wheel_assembly.xacro* definieras huvudsakligen två links och två joints:

- *wheel_steer_link*: Drivhjulets styrlänk vars rotation bestämmer vilken riktning drivhjulet står i (jmfr. framgaffeln på en cykel).
- *wheel_drive_link*: Själva drivhjulet.
- *wheel_steer_joint*: Specificerar hur *wheel_steer_link* kan röra sig relativt *base_link*. Relationen är av typen *revolute*, vilket innebär att den kan rotera inom vissa gränser, i detta fall mellan -175° och +175°.
- *wheel_drive_joint*: Beskriver hur *wheel_drive_link* kan röra sig relativt *wheel_steer_link*. Denna relation är av typen *continous*, vilket innebär att hjulet kan rotera obegränsat i båda riktningarna. Rotationsaxeln för denna joint är drivhjulets y-axel (dvs. dessa naturliga rotationsaxel). Förutom dessa begränsningar är även den maximala rotationshastigheten satt till 1,5 rad/s.

En principiell skiss av den befintliga drivhjulsmodellens uppbyggnad presenteras i Figur 4.



Figur 4: *Principiell hjulkonfiguration hos den befintliga modellen, beskriven i filen* drive_wheel_assembly.xacro. *Gråa objekt representerar links, röda bågar representerar joints och blå pilar representerar definierade rotationsriktningar.* base_link *är truckens huvudkropp som definieras i huvudfilen* minireach.xacro.

3.4 Utökad modell

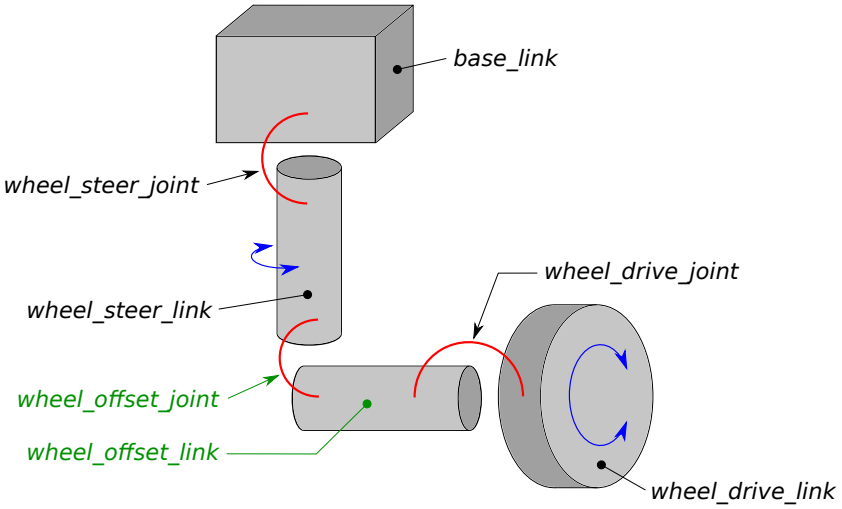
Från ritningar på truckarna kan först storleken på hjul-offseten bestämmas. Denna kan sedan implementeras i modellen på ett antal olika sätt.

3.4.1 Alternativ 1: Införande av ny link och joint

Genom att införa en ny link och en ny joint mellan *wheel_steer_link* och *wheel_drive_link* kan hjulet förskjutas en given längd från styrningens rotationsaxel. Låt oss kalla dessa nya element för *wheel_offset_link* och *wheel_offset_joint*. Den senare kommer vara av typen *fixed*, vilket innebär att *wheel_offset_link* och *wheel_steer_link* kommer rotera med varandra som om de vore fixt förbundna till varandra. I Figur 5 visas det principiella utseendet hos detta modelleringsalternativ.

3.4.2 Alternativ 2: Införa offset direkt i *wheel_drive_joint*

När en joint definieras kan offset i x-, y- och z-led från utgångspositionen specificeras (i xyz-attributet i origin-elementet för jointen). Dessa parametrar skulle kunna användas för att förskjuta hjulet från hjulstyrningens rotationsaxel. I dagsläget är dessa parametrar satta till (0 0 0) för *wheel_drive_joint*. Inledande tester har gjorts där y-ledsparametern för denna joint ändrats och simuleringen startats, och mycket riktigt så blev drivhjulet förskjutet i y-led. Det var dock oklart huruvida simuleringsmiljön faktiskt tog hänsyn till de kinematiska effekterna av hjulets förskjutning eller ej, så detta alternativ behöver utredas vidare.



Figur 5: *Principiell skiss av modellen efter införande av en ny link* wheel_offset_link *och en ny joint* wheel_offset_joint. *Grön färg indikerar de nyinförda elementen. Den nya* wheel_offset_joint *måste vara av* fixed*-typ så att* wheel_offset_link *roterar med* wheel_steer_link *som om de vore fastmonterade i varandra.*

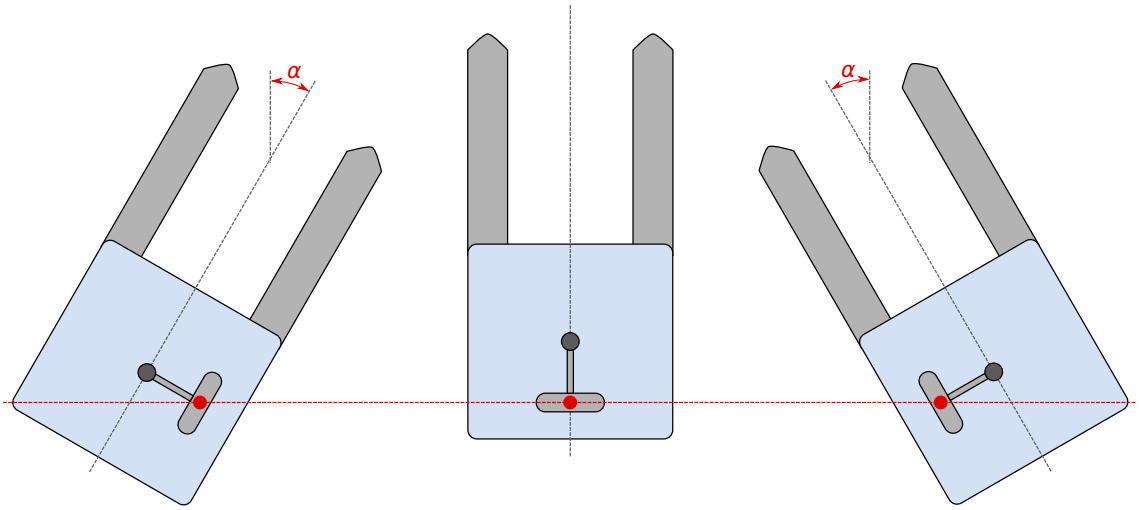
3.4.3 Val av implementeringsalternativ

Möjligheter och begränsningar hos de två olika implementeringsalternativen kommer att undersökas och det alternativ som ger bäst överensstämmelse mellan simulering och verklighet kommer att väljas. Alternativ 2 kommer vara det första att undersökas då det är mycket enkelt att implementera och skulle eventuellt kunna lösa problemet på ett smidigt sätt.

3.5 Validering

För att undersöka hur bra simuleringen efterliknar verkligheten kommer ett modellvalideringsprogram tas fram. Som beskrivits i avsnitt *3.1 Bakgrund* visar sig modellfelet bland annat då drivhjulet roteras fram och tillbaka på en i övrigt stillastående truck. Hos den verkliga trucken kommer då gaffelspetsarna att växelvis svänga till vänster och höger när trucken vinkelställs som svar på drivhjulsrotationen, medan den simulerade trucken står helt stilla. Modellvalideringsprogrammet kommer att utnyttja detta fenomen och helt enkelt utföra upprepade rotationer av drivhjulet fram och tillbaka mellan dessa ändlägen. Valideringen sker sedan genom okulär analys och jämförelse av både den verkliga och den simulerade truckens rörelser.

I Figur 6 presenteras en skiss av hur denna vinkelställning sker.



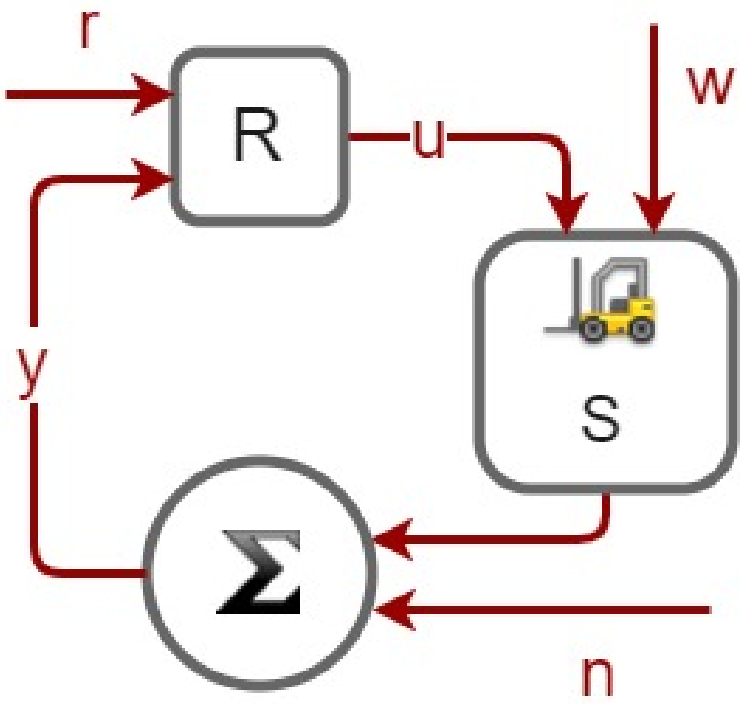
Figur 6: *Skiss av hur den verkliga trucken vinklar sig då drivhjulet vrids till olika vinklar. De röda punkterna markerar den punkt som är fix under vridningen (drivhjulets kontaktpunkt med underlaget). Vinkeln* α *indikerar truckens vinkelställning. Observera att vissa avstånd och vinklar i denna figur är ordentligt överdrivna för tydlighetens skull, så figuren är alltså inte skalenlig.*

4 Delområde 2 - Tillståndsmodell

En tillståndsmodell av trucken utvecklas med syfte att användas i en modellbaserad prediktions-regulator, MPC i delområde 4. I ett första steg ska en inledande modell som beskriver truckens dynamik utvecklas. Därefter i steg två av arbetsprocessen ska denna modell utökas så den också hanterar mätfel med en brusmodell. Slutligen utökas den i ett tredje steg så att modellen hanterar gaffelns höjd som en variabel och kan reglera denna med en utsignal som styr läget på gaffeln.

4.1 Systembeskrivning

Ett blockschema som beskriver regulatorn R, systemet (trucken) S och de olika signalerna som påverkar visas i figur 7. De olika signalerna förklaras i tabell 1.



Figur 7: *Blockschema med regulator R och truck S.*

De faktiska mätvärdena från lasersensorerna är avstånd till närmaste hinder framför respektive bakom trucken. Nuvarande version av trucken använder ett färdigt ROS-bibliotek, *Navigation* för att positionera trucken i rummet utifrån sensordata. För att undvika att utveckla en egen funktion för att beräkna truckens position används robotens position som Navigation beräknar som mätsignal. Trucken är dessutom utrustad med en positionsensor som mäter gaffelns höjd över marken.

En tillståndsmodell gör det möjligt att entydigt bestämma framtida utsignaler från ett system om dess framtida insignaler är kända [2]. Sambandet mellan ett systems in- och utsignaler skrivs:

Signal	Namn	Beskrivning
r	Referenssignal	Den bana som regulatorn ska få trucken ska följa.
u	Styrsignal	Den instruktion regulatorn beräknar att trucken ska utföra för att följa den givna referenssignalen.
ω	Systemstörning	Andra yttre faktorer som påverkar truckens tillstånd.
y	Mätsignal	Truckens uppskattade position från mätvärden från lasersensorer.
n	Mätstörningar	Fel i truckens uppskattade position till följd av mätfel i lasersensorerna.

Tabell 1: *Beskrivning av signalerna i systemet.*

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \\ y(t) &= g(x(t), u(t)) \end{aligned} \tag{1}$$

Här är x(t) tillståndsvektorn med systemets alla tillstånd. Vilka tillstånd som ska användas ska undersökas i projektet. Olika modeller ska utvecklas med olika uppsättningar av tillstånd och därefter ska de olika modellerna undersökas för att avgöra vilka tillstånd som ska användas.

$u(t)$ i ekvation 1 är styrsignalerna till trucken. Styrningen av trucken sköts med ett färdig ROS-bibliotek som styr trucken efter Twist-kommandon. Ett Twist-kommando består av 6 variabler, önskad hastighet och rotation i x-, y- och z-led. Hastigheten i y- och z-led och rotationen i x- och y-led kommer alltid vara 0 eftersom trucken inte kan röra sig i dessa riktningar. Detta ger totalt två styrsignaler som går att påverka trucken med. Dessutom används en signal för att styra gaffelns läge.

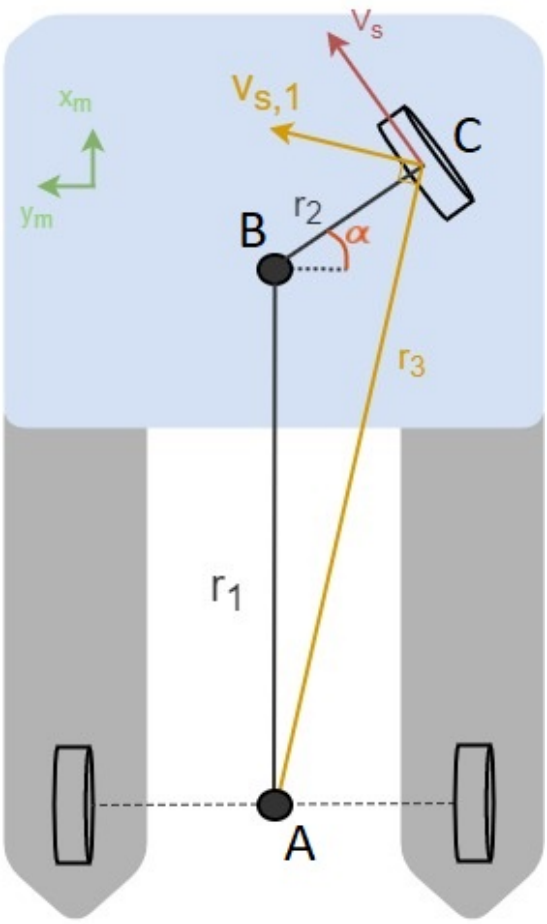
4.1.1 Dynamiska Modeller

Denna sektion beskriver den kinematiska modell som ska undersökas.

Inledningsvis utformades grundläggande kinematiska tillstånd för trucken definierade i kartesiska koordinatsystem. Genom att utgå från inertialramen som definieras av $\{X_i, Y_i\}$ kan därefter truckens koordinatsystem definieras enligt $\{X_m, Y_m\}$ (moving frame) och slutligen vinkeln θ, se figur 9. Här kan nu rotationsmatrisen $R(\theta)$ uttryckas för att beskriva inertialramens position i förhållande till truckens koordinatsystem.

$$R(\theta) = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

I figur 8 definieras vinklar och avstånd i truckens koordinatsystem vilket används för att uttrycka den kinematiska modellen. För att ta hänsyn till den offset som förekommer på styrhjulet är det viktigt att beräkna avståndet från punkten mellan framgafflarna (A) till styrhjulets mittpunkt (C) enligt figur 8 och benämns r_3. Det avståndet är nödvändigt för att bestämma ett uttryck för



Figur 8: *Trucken ovanifrån med vinklar, avstånd, punkter och hastigheter utmarkerade.*

truckens rotationshastighet runt punkten A i $\{X_m, Y_m\}$ och betecknas $\dot{\theta}$ i $\{X_i, Y_i\}$. Avståndet r_3 utvecklades enligt figur 8 och uttrycks enligt ekvation 2.

$$r_3 = \sqrt{(r_1 + r_2 sin(\alpha))^2 + (r_2 cos(\alpha))^2} = \sqrt{{r_1}^2 + 2 r_1 r_2 sin(\alpha) + {r_2}^2} \tag{2}$$

Rotationshastigheten $\dot{\theta}$ kan nu skrivas enligt ekvation 3.

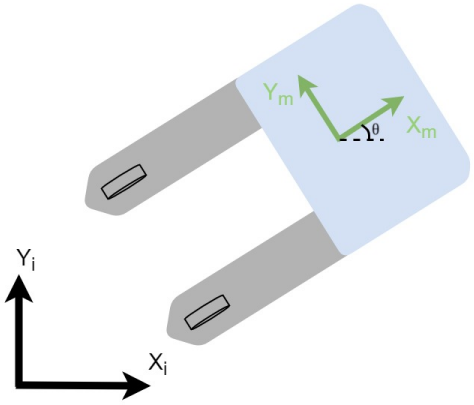
$$\dot{\theta} = tan^{-1}\left(\frac{V_s}{r_3}\right) \tag{3}$$

Truckens hastighet i X- och Y-led i $\{X_m, Y_m\}$ definieras som V_x och V_y och V_s står för styrhjulets hastighet enligt figur 8. Förutsatt ingen glidning ges hastigheterna i de olika leden av ekvation 4 och 5.

$$V_x = V_s \, cos(\alpha) \tag{4}$$

$$V_y = 0 \tag{5}$$

Nu kan den kinematiska modellen uttryckas i både inertialramen $\{X_i, Y_i\}$ och truckens koordinatsystem $\{X_m, Y_m\}$ med hjälp av ekvation 3, 4, 5 och rotationsmatrisen.



Figur 9: *Trucken i dess omgivning.*

Kinematisk model i truckens koordinatsystem $\{X_m, Y_m\}$ (moving frame)

$$\dot{x}_m = \begin{cases} \dot{x} = V_s \, cos(\alpha) \\ \dot{y} = 0 \\ \dot{\theta} = tan^{-1}\left(\frac{V_s}{r_3}\right) \end{cases} \tag{6}$$

Kinematisk model i inertialramen $\{X_i, Y_i\}$

$$\dot{x}_i = R[\theta]\dot{x}_m = \begin{cases} \dot{x} = V_s \, cos(\alpha) \, cos(\theta) \\ \dot{y} = V_s \, cos(\alpha) \, sin(\theta) \\ \dot{\theta} = tan^{-1}\left(\frac{V_s}{r_3}\right) \end{cases} \tag{7}$$

Hastigheten för hjulet i kontaktytan V_s beräknas med hjälp av insignalerna till systemet som är hastigheten $\dot{x}_A$ och rotationshastigheten $\dot{\psi}_A$ för punkten A mellan framgafflarna enligt ekvationen

$$V_s = \dot{\psi} r_3 - \dot{x}_A \text{sgn}(\cos(\alpha))$$

där sgn anger signumfunktionen som ges av

$$\text{sgn}(x) = \begin{cases} -1 & om \quad x < 0 \\ 0 & om \quad x = 0 \\ 1 & om \quad x > 0 \end{cases}$$

Detta är en mycket grov förenkling av systemet som utgår från att hastigheter kan ställas om momentant utan accelerationer. Dessutom antas att vinkeln α mellan styrhjulet och x-axeln

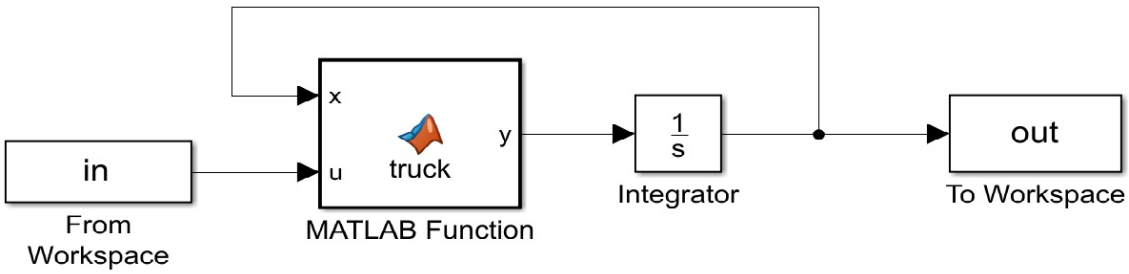
ställs om momentant och att punkterna C och B är på samma ställe dvs. att $r_2 = 0$.

För att göra modellen mer verklig kan ett lågpassfilter som filtrerar styrsignalerna till systemet och endast släpper igenom långsamma förändringar, användas. Ett sådant filter anpassar styrsignalerna till systemet och förhindrar stora momentana förändringar av hastigheter. Hur ett sådant ska designas för att efterlikna det verkliga systemet på bästa sätt ska undersökas.

4.1.2 Modellutvärdering

Metoden för att utvärdera de olika tillståndsmodellerna är relativt enkel. Trucken styrs med en förutbestämd insignal i simulatorn och verkligheten. Truckens trajektoria sparas och jämförs med de trajektorior som fås då insignalen används som insignal till modellen.

Både en okulär och en matematisk jämförelse genomförs. Den okulära jämförelsen genomförs genom att plotta trajektoriorna mot varandra och se vilken modell som ger en trajektoria som efterliknar den verkliga och simulerade bäst. Den matematiska jämförelsen summerar avståndet mellan modellernas trajektorior och verkligheten respektive simulatorns trajektorior i varje samplingsögonblick. En översiktlig Simulink modell för att utvärdera en given insignalsekvens ses i figur 10.



Figur 10: *Simulink modell för att utvärdera tillståndsmodeller.*

4.2 Störningar

Störningar delas in i två kategorier, systemstörningar ω och mätstörningar n. Deras påverkan på systemet samt hur de ska kompenseras beskrivs i detta avsnitt. Denna teori gäller endast med en linjärt system och gäller därför inte för den tillståndsmodell som använts i detta kapitel.

4.2.1 Systemstörningar

Förutom styrsignalen som regulatorn skickar till trucken påverkas trucken också av systemstörningar. Det kan vara störningar i truckens omgivning, *belastningsstörningar*, exempelvis att underlaget i lagret förändras, variationer i processer och ställdon i trucken till exempel på grund av slitage eller att förenklingar gjorts i modellen så att den inte följer verkligheten exakt.

Av de uppräknade faktorerna antas modellfel vara den mest betydelsefulla störningskällan. För att modellera modellfelet utnyttjas att truckens exakta värde på alla tillstånd kan fås ur simulatorn. Skillnaden i de exakta värden och värdena i tillståndsmodellen utifrån givna insignaler benämns systemfelet, ω_f och kan användas för att modellera systemstörningarna.

En signal som det inte går att förutsäga någon information om framtida värden utifrån gamla värden kallas vitt brus. Vitt brus har ett konstant spektrum vilket betyder att dess energi är fördelad jämt över alla frekvenser. Med hjälp av systemfelen vill vi undersöka om man kan förutsäga framtida systemfel utifrån gamla värden och i så fall vilka tillstånd i systemfel påverkar vilka framtida tillstånd i systemfelet. Det vill säga om alla tillstånd i systemfelet har konstant spektrum eller om vi kan utnyttja information om systemfelet i vår tillståndsmodell.

För de tillstånd som inte har konstant spektrum bestäms en överföringsfunktion H så att

$$\omega_f(t) = H(p)v_\omega(t)$$

där $v_\omega(t)$ är en vektor med vitt brus, uppfylls. ω_f är de tillstånd som inte har ett konstant spektrum och v_ω representerar vitt brus på de tillstånd som påverkar tillstånden i ω_f. Om något tillstånd har konstant spektrum och alltså anses vara vitt brus men ändå påverkar ett annat tillstånd som inte har konstant spektrum kommer detta tillstånd vara representerat i v_ω men inte i ω_f. Dessa vektorer kan alltså ha olika dimensioner.

Om H skrivs på tillståndsform fås

$$\begin{aligned} \dot{x}_\omega &= A_\omega x_\omega + B_\omega v_\omega \\ \omega &= C_\omega x_\omega + D_\omega v_\omega \end{aligned}$$

som kombinerat med ekvation 1 ger

$$\begin{aligned} \dot{x}_b &= A_b x_b + B_b u + N_b v_\omega \\ y &= C_b x_b + D_b u \end{aligned} \tag{8}$$

där

$$x_b = \begin{bmatrix} x \\ x_\omega \end{bmatrix} \quad A_b = \begin{bmatrix} A & C_\omega \\ 0 & A_\omega \end{bmatrix} \quad B_b = \begin{bmatrix} B \\ 0 \end{bmatrix}$$
$$N_b = \begin{bmatrix} D_\omega \\ B_\omega \end{bmatrix} \quad C_b = \begin{bmatrix} C & 0 \end{bmatrix} \quad D_b = D$$

4.2.2 Mätstörningar

Mätstörningar uppkommer på grund av fel i mätningarna. Dessa måste tas hänsyn i regulatorn så den inte ”blint litar på” att mätvärdena den får från trucken är korrekta utan att den kompenserar för att dessa kan vara något avvikande. Hur mycket regulatorn ska lita på mätvärdena

avgörs av storleken på mätfelen. För att avgöra storleken utnyttjas även här att simulatorn kan ge de exakta värdena på tillstånden som mäts. Med hjälp av skillnaden på de exakta värdena och de uppmätta värden kan man få ut information om störningens utseende.

Värdena på mätfelet används för att undersöka om det går att förutsäga något om framtida mätfel utifrån gamla värden. Det är exempelvis möjligt att om trucken uppskattar sin position något förskjuten i en riktning att nästa mätning också kommer ge en position förskjuten i denna riktning.

Om mätfelet $n(t)$ inte har konstant spektrum vill vi hitta en överföringsfunktion H_n så att

$$n(t) = H_n(p)v_2(t)$$

där $v_n(t)$ är vitt brus. Om H_n representeras på tillståndsform

$$\begin{aligned} \dot{x}_n(t) &= A_n x_n + B_n v_n \\ n &= C_n x_n + D_n v_n \end{aligned} \tag{9}$$

kan tillståndsmodellen i ekvation 8 utökas till att också beskriva mätfelen genom

$$\begin{aligned} \dot{x} &= Ax + Bu + Nv_1 \\ y &= Cx + Du + v_n \end{aligned} \tag{10}$$

där

$$x = \begin{bmatrix} x_b \\ x_n \end{bmatrix} \quad A = \begin{bmatrix} A_b & 0_\omega \\ 0 & A_n \end{bmatrix} \quad B = \begin{bmatrix} B_b \\ 0 \end{bmatrix} \quad v_1 = \begin{bmatrix} v_\omega \\ v_n \end{bmatrix}$$
$$N = \begin{bmatrix} N_b & 0 \\ 0 & B_n \end{bmatrix} \quad C = \begin{bmatrix} C_b C_n & 0 \end{bmatrix} \quad D = D_b$$

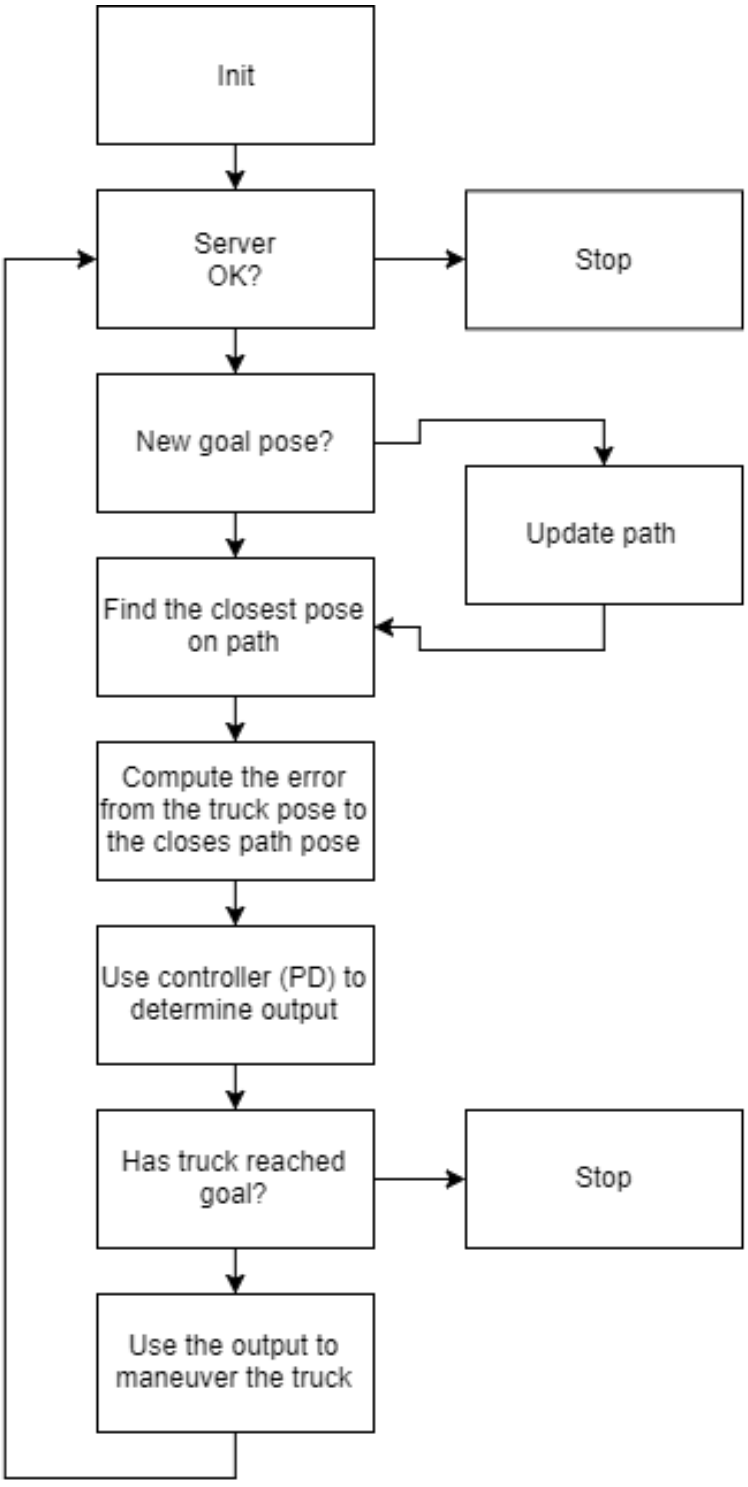
5 Delområde 3 - Reglering av precisionsinkörning

5.1 Bakgrund

Systemet klarar i dagsläget att reglera in trucken tillräckligt bra för att kunna hämta en pall i ett ställage. Denna reglering sker dock i olika delmoment med olika planerare och regulatorer och kräver relativt strikta förhållanden. För att få ett mer generellt system planerar vi att utveckla en bättre regulator som både kan implementeras när trucken rör sig fritt i lagret samt när trucken genomför en precisionsinkörning för att hämta en pall. Detta delområde kommer att vara beroende av delområde 2 då en regulator kan komma att kräva en tillståndsmodell. Projektet planerar att undersöka PID, framkoppling och MPC-reglering.

5.2 Systembeskrivning

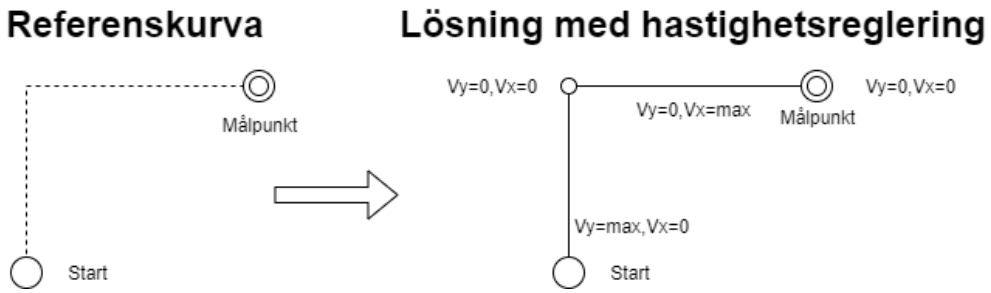
I det nuvarande systemet regleras precisionskörningen med hjälp av en PD-regulator enligt figur 11. Regleringsberäkningarna utförs i noden ”move-precise” som skickar önskad hastighet och vinkelhastighet som ett meddelande till drivnoden som utför själva styrningen. Detta betyder att styrnoden har en egen reglering. Denna reglering kommer göra det svårare för vår regulator att reglera då den döljer en del av den mekaniska dynamiken av systemet.



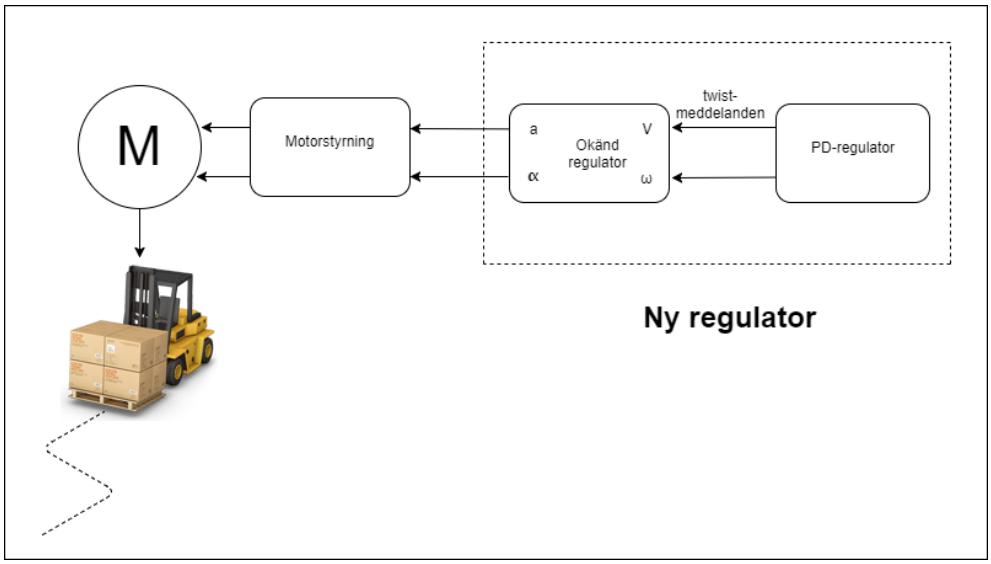
Figur 11: *Flödesschemat för den nuvarande regleringen.*

En viktig observation angående den nuvarande regleringen är att det som regleras är hastighet och vinkelhastighet istället för acceleration och vinkelacceleration. Detta medför att om problemet skulle ställas upp som ett optimeringsproblem som i en MPC skulle den optimala lösningen vara den som visas i figur 12. Alltså att man kör med full hastighet tills man ska svänga stoppar hastigheten helt och sen kör full hastighet till mål. Detta är naturligtvis inte genomförbart i praktiken och det kommer därför vara viktigt att om man vill implementera en MPC se till att använda acceleration och vinkelacceleration som styrsignal. Problemet är dock att det i nuläget

endast finns twist meddelanden som används för styrning till hastigheter. Omvandlingen till accelerationer sker därför antagligen med någon slags reglering som därför måste ändras. Se figur 13 för en bättre beskrivning på detta problem.



Figur 12: *Bilden visar problemet med att se hastigheten som styrsignal*



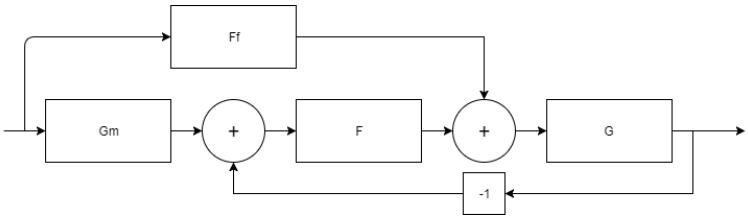
Figur 13: *Schema som visar de nuvarande styrsignalerna samt problemet med att kunna styra med acceleration,a och vinkelacceleration,*α

5.3 Implementera PID

5.3.1 PID

Eftersom nuvarande regulator redan innehåller en P- och en D-del kommer vi bara behöva utveckla en I-del. Projektgruppen kommer dock att implementera regulatorn enligt [1]:

ReadInput(r,y)
e := r-y;
v := a + b*e;
if (v > u_{max})
    u := u_{max};
else if (v < u_{min})
    u := u_{min};



Figur 14: *Flödesschema för hur framkopplingen och återkopplingen ska implementeras*

else
    u := v;
Out(u);
I := I + K$\frac{T_S}{T_i}$e
a := I - K$\frac{T_d}{T_S}$e
b := K + K$\frac{T_d}{T_S}$ + K$\frac{T_S}{T_i}$
Wait;
där r = referenssignalen, y = utsignal och u = styrsignal.

Referenssignalen kommer vara truckens önskade position och vinkel enligt den genererade kurvan. Utsignalen kommer vara truckens faktiska position och vinkel. Styrsignalen kommer vara den acceleration och vinkelacceleration. Parameterinställning för PID sker genom stegsvar och avläsning av K,Ti,Td enligt standardmetoder.

5.4 Implementera PID med framkoppling

Framkopplingen kommer kopplas på utanför PID-regleringen och påverka styrsignalen direkt enligt figur 14. Implementeringen av framkopplingen beror väldigt mycket på hur den dynamiska modellen över systemet ser ut, vilket gör att den inte går att beskriva i detalj i nuläget.

5.5 Implementera MPC

Vid implementering av MPC-regulatorn behöver man komma åt och ersätta den okända regulatorn som är implementerad i drivnoden eftersom det finns många dolda tillstånd i den. Detta verkar dock väldigt omfattande då större delen av koden är uppbyggd runt denna okända regulator. Ett annat alternativ är att modulera den okända regulatorn som ett lågpassfilter eller att straffa för stora ändringar styrsignalen via MPC-regulatorn. Detta gör att regulatorn inte planerar en utsignal som är omöjlig att genomföra.

Vi kommer eventuellt vilja implementera MPC-regulatorn med C++ för att få effektivare beräkningar. I detta falla kommer man behöva skapa en ny nod för regleringen. Vid själva utvecklingen av regulatorn planerar projektgruppen att använda Matlab.

5.5.1 MPC-teori

Eftersom banföljningen kommer ske i realtid är det extra viktigt att MPC-regulatorn som senare implementeras är beräkningseffektiv och kan hantera korta samplingstider. Eftersom offseten

på styrhjulet leder till en olinjäritet kan detta ha påverkan om man väljer att göra en linjär MPC. Väljer man istället en olinjär MPC är optimeringsproblemet icke konvext och man kan därför inte använda en kvadratisk lösare. Dessutom kommer den framtagna modellen i delområde 2 i det fallet behöva vara olinjär. Detta gör MPC designen mycket mer tidskrävande vilket alltså helst bör undvikas.

Målfunktionen för en grundläggande MPC är:

$$\begin{aligned}\sum_{j=0}^{N-1} ||z(k+j)||^2_{Q_1} + ||u(k+j)||^2_{Q_2} &= /linjrmpc/ = X^T \mathscr{M}^T Q_1 \mathscr{M} X + U^T Q_2 U \\ &= (\mathscr{F}x(k) + \mathscr{G}U)^T \mathscr{M}^T Q_1 \mathscr{M}(\mathscr{F}x(k) + \mathscr{G}U) + U^T Q_2 U\end{aligned} \tag{11}$$

I ekvation 11 är Q_1 och Q_2 diagonala viktmatriser där relationen mellan dessa bestämmer om regulatorn ska lägga mest vikt vid att använda små styrsignaler, stort Q_2, eller ha ett litet reglerfel, Q_1. $\mathscr{Q}_1$ och $\mathscr{Q}_2$ är blockdiagonala matriser med Q_1 och Q_2 repeterade N gånger. $\mathscr{M}$ är en blockdiagonal matris med M repeterad N gånger.

Målfunktionen för MPC-reglering kan sedan utvecklas med integralverkan och skrivs då enligt ekvation 12.

$$\sum_{j=0}^{N-1} ||z(k+j) - r(k+j)||^2_{Q_1} + ||u(k+j) - u(k+j-1)||^2_{Q_3} \tag{12}$$

Prestandamåttet kan därför skrivas som i ekvation 13 ur vilket man kan härleda QP-problemet då integralverkan och referensföljning används. Det man bör tänka på vid härledningen av QP-problemet är att det endast är de termer som beror på U som är intressanta då de andra är konstanta.

Omskrivningen av målfunktion med integralverkan för linjär mpc:

$$\begin{aligned}&(\mathscr{M}(\mathscr{F}x(k) + \mathscr{G}U) - R)^T \mathscr{Q}_1 (\mathscr{M}(\mathscr{F}x(k) + \mathscr{G}U) - R) + (\Omega U - \delta)^T \mathscr{Q}_3 (\Omega U - \delta) => \\ &\quad /\text{delar som är beroende av U}/ => U^T \mathscr{G}^T \mathscr{M}^T \mathscr{Q}_1 \mathscr{M} \mathscr{F} x(k) - U^T \mathscr{G}^T \mathscr{M}^T \mathscr{Q}_1 R + \\ &(x(k))^T \mathscr{F}^T \mathscr{M}^T \mathscr{Q}_1 \mathscr{M} \mathscr{G} U - R^T \mathscr{Q}_1 \mathscr{M} \mathscr{G} U - U^T \Omega^T \mathscr{Q}_3 \delta - \delta^T \mathscr{Q}_3 \Omega U + U^T \mathscr{G}^T \mathscr{M}^T \mathscr{Q}_1 \mathscr{M} \mathscr{G} U + U^T \Omega^T \mathscr{Q}_3 \Omega U\end{aligned}$$

Nedanför ser man ekvationen för det problem som regulatorn löser för varje samplingsögonblick. Denna gäller såväl för olinjär som för linjär MPC. Om en linjär mpc ska göras implementerar man ovanstående omskrivning av målfunktionen. Medan man för det olinjära fallet implementerar målfunktionen som gavs i ekvation: 12. Det räcker dessutom inte att bara generera denna en gång utan man kommer behöva generera flera och sedan minimera med avseende på U, för att plocka fram den bana som blir bäst. Detta kan dock också göra MPC:n allt för beräkningskrävande. Därför kan det finnas behov att antingen minska prediktionshorisonten eller placera ut en mindre antal punkter längs kurvan att reglera efter.

$$\min_U \text{målfunktion} \tag{13}$$

$$\text{bivillkor } |u| \leq ubounds \tag{14}$$

För att sedan lösa minimeringsproblemet finns för det linjära fallet en funktion quadprog i Matlab medan för det olinjära fallet finns funktionen fmincon. Motsvarighet till dessa eller liknande funktioner finns antagligen också i Python eller C++.

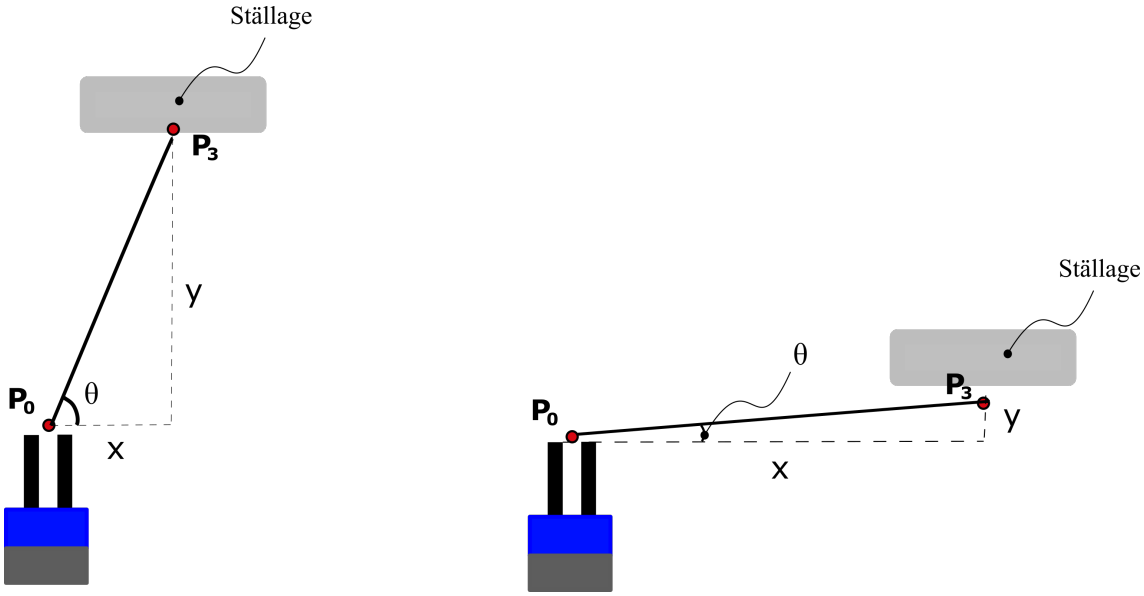
5.6 Problemanalys

Efter en analys av problemet ser man att olinjäriteten i trucken beror av vinkeln på styrhjulet. Detta tillsammans med det faktum att trigonometriska funktioner (som kommer av vinklarna) alltid är olinjära, innebär att utan linjärisering kommer problemet behöva lösas olinjärt. Med tanke på att en linjärisering skulle innebära begränsningar på styrhjulsvinkeln och detta inte är önskvärt i detta fall då målet är att kunna göra skarpa kurvor, innebär det att den bästa lösningen borde bli olinjär. Om regleringen görs med en MPC betyder det att en olinjär MPC kommer behöva implementeras vad detta betyder diskuteras i avsnittet MPC-teori. Om regleringen löses med framkoppling och återkoppling behöver dock olinjäriteten inte ha någon större betydelse för problemlösningsgången mer än att regleringen blir suboptimal. I återkopplingsfallet, om man skulle implementera en PID som ju är en linjär regulator kan man behöva ändra parameterinställningarna beroende på hjulets vinkel. Olinjäriteten kan teoretiskt också komma att påverka framkopplingen.

6 Delområde 4 - Planering av precisionsinkörningskurva

6.1 Nuvarande lösning

Referenskurvan som krävs för precisionsinkörningen i regulatorn (se delområde 3) genereras i skrivande stund som en Bézier-kurva. Den genereras genom att två kontrollpunkter placeras ut baserat på banans start- och slutposition: en kontrollpunkt rakt framför startpositionen och en rakt bakom slutpositionen (se figur 16). Avståndet som dessa punkter placeras på bestäms genom att först beräkna avståndet fågelvägen mellan start- och slutpositionen (se sträckan P_0 till P_3 i figur 15) och sedan skala kontrollpunkternas avstånd baserat på detta fågelvägsavstånd.



Figur 15: *Bilden till vänster och höger visar vinkeln mellan fågelvägen (P_0 till P_3) och avståndet i x- resp. y-led mellan trucken och stället.*

6.1.1 Matematisk beskrivning av Bézier-kurva

Bézier-kurvan genereras med hjälp av så kallade kontrollpunkter som ligger nära kurvan enligt följande ekvation:

$$B(t) = \sum_{i=0}^{n} b_{i,n}(t) P_i, \quad t \in [0,1] \tag{15}$$

där P_i motsvarar en given sekvens av kontrollpunkter och n motsvarar grad. Kontrollpunkterna P_0 och P_n är alltid start-och slutpunkterna på kurvan. Kurvans fart och riktning ges av de övriga punkterna $P_{1..(n-1)}$ som inte behöver ligga på kurvan.

Bernstein polynomet $b_{i,n}(t)$ ges i sin tur av

$$b_{i,n}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-1} t^i \tag{16}$$

där $\binom{n}{i}$ är binomialkoefficienter.

6.1.2 Huvudfilen *move_precise.py*

Den fil som körs när precisionsinkörningen påbörjas är filen *move_precise.py* i paketet *minireach_tasks*. Denna fil består av många olika funktioner, där de viktigaste sammanfattas nedan.

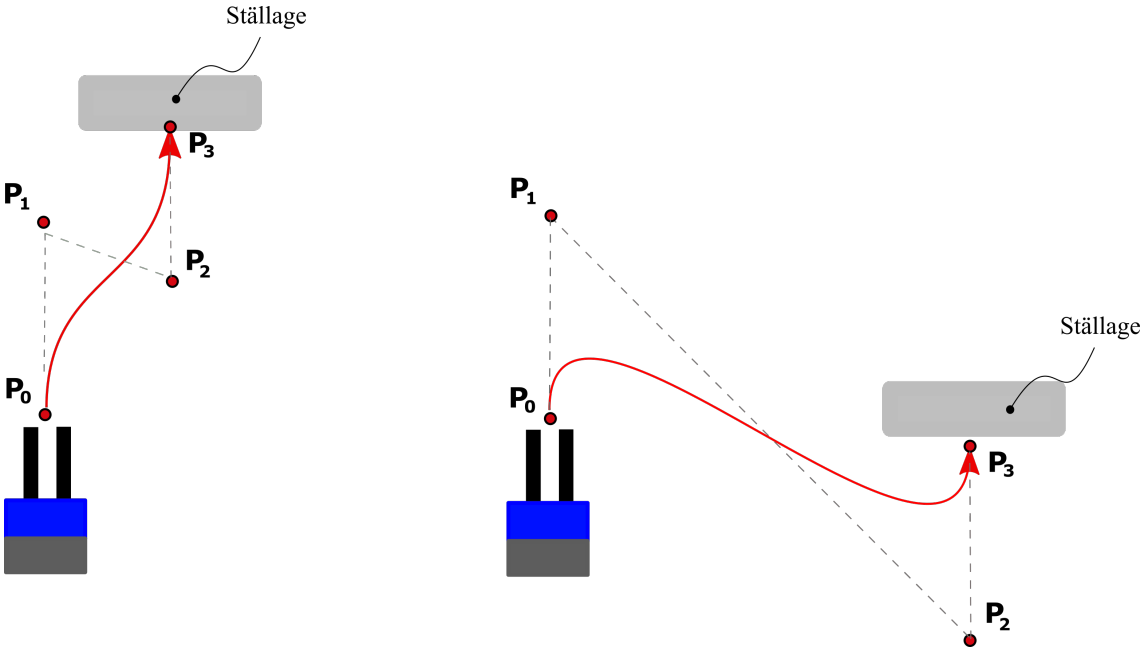
- *__execute*: Huvudfunktionen som körs när precisionsinkörningen börjar
- *__path_from_start_and_goal*: Skapar körbana givet en startposition och en slutposition. Kallar bland annat på funktionen *__generate_path*
- *__generate_path*: Funktion som genererar Bézier-kurva från givna kontrollpunkter

Början av funktionen *__execute* är av speciellt intresse inför den vidare utvecklingen av banplaneringen, så den presenteras så som den ser ut i dagsläget nedan.

```
def __execute(self, goal):
    rospy.loginfo("Execute move precise action")

    self._new_goal_pose = None
    update_goal_sub = rospy.Subscriber(self._action_name + '_update_goal',
            Pose, self.__update_goal_cb, queue_size=1)

    (start_pos, start_quaternion) = self._tf_listener.lookupTransform("map",
            "base_footprint", rospy.Time())
    start_pose = Pose()
    start_pose.position = self.__from_array_position(start_pos)
    start_pose.orientation = self.__from_array_quaternion(start_quaternion)

    goal_pos = self.__to_array_position(goal.target_pose.pose.position)

    path = self.__path_from_start_and_goal(start_pose, goal.target_pose.pose)

    # Funktionen fortsätter...
    # ...
    # ...
```

6.2 Problematik

I vissa fall fungerar den genererade Bézier-banan bra. Kurvan är lämplig t.ex. då start- och slutpositionen ligger långt ifrån varandra med en tillräckligt stor vinkel θ mellan dessa (se vänster sida av figur 15). Kurvan lämpar sig däremot inte för fall då ändpunkterna ligger nära varandra längs med en horisontell linje, enligt höger sida av figur 16. I detta fall genereras istället en ytterst snäv kurva med begränsade regleringsmöjligheter.



Figur 16: *Bilden till vänster och höger visar två olika fall av hur trucken och ställaget står i förhållande varandra. I den vänstra bilden följer trucken en mjukare Bezier-kurva medan bilden till höger visar fallet då trucken följer en snävare bana.*

6.3 Lösning

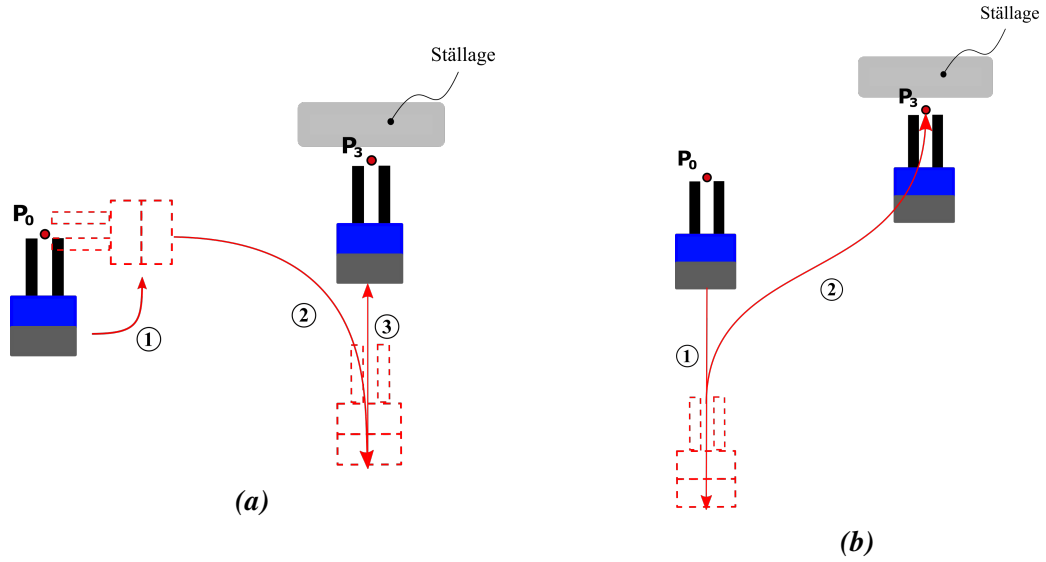
Med bakgrund av ovanstående resonemang krävs en noggrann planering som tar hänsyn till olika fall. Dessa fall borde innefatta startpositionen och dess vinkel i förhållande till slutpositionen och dess vinkel. Givet dessa förutsättningar skulle en justering av truckens startläge först kunna genomföras genom att trucken roterar med en viss vinkel runt sin egen axel så att truckens startposition är riktad från dess slutposition. Ett exempel på detta går att skåda i figur 17a där trucken kör sträckan P_0 till P_3 på följande sätt:

1. Trucken roterar 90°moturs.
2. Backar bakåt.
3. Kör en mjuk Bézier-kurva till slutpositionen.

Ett annat alternativ kan vara att trucken flyttar sig till en lämplig startposition genom att åka framåt, bakåt eller åt sidorna. Figur 17b visar ett exempel på hur detta kan gå till, enligt följande:

1. Trucken backar rakt bakåt.
2. Kör en mjukare Bézier-kurva till slutpositionen.

När justeringen av startläget utförts så följer precisionsinkörningen en Bézier-kurva enligt redan befintlig metodik. På detta sätt ges ett uttryck för en ”mjukare” kurva.



Figur 17: *Delfigurerna (a) och (b) visar två olika justeringsfall av truckens startläge.*

6.3.1 Implementering

I dagsläget körs funktionen *__execute* i filen *move_precise.py* när precisionsinkörningen börjar (se avsnitt 6.1.2). I början av denna funktion genereras en Bézier-bana med hjälp av funktionen *__path_from_start_and_goal* och givna start- och slutpositioner. För att använda olika banplaneringsprocedurer beroende på t.ex. start- och slutpositioner och vinklar så behöver denna *__execute*-funktion modifieras. I koden nedan kommenteras ett av de ställen där ingrepp behöver göras för att ändra banplaneringen.

```
def __execute(self, goal):
    rospy.loginfo("Execute move precise action")

    self._new_goal_pose = None
    update_goal_sub = rospy.Subscriber(self._action_name + '_update_goal',
            Pose, self.__update_goal_cb, queue_size=1)

    (start_pos, start_quaternion) = self._tf_listener.lookupTransform("map",
            "base_footprint", rospy.Time())
    start_pose = Pose()
    start_pose.position = self.__from_array_position(start_pos)
    start_pose.orientation = self.__from_array_quaternion(start_quaternion)

    goal_pos = self.__to_array_position(goal.target_pose.pose.position)

    # Denna del av funktionen genererar Bézier-kurvan. Bland annat här
    # kommer ingrepp behöva göras för att använda olika
    # banplaneringsprocedurer baserat på t.ex. start- och slutpositioner och
    # -vinklar
    path = self.__path_from_start_and_goal(start_pose, goal.target_pose.pose)

    # Funktionen fortsätter...
    # ...
    # ...
```

Referenser

[1] *Industriell reglerteknik ,Kurskompendium*, Institutionen för systemteknik. Linköpings universitet, 2014.

[2] *Reglerteori, Flervariabla och olinjära metoder*, Glad, Ljung. Studentlitteratur 2003