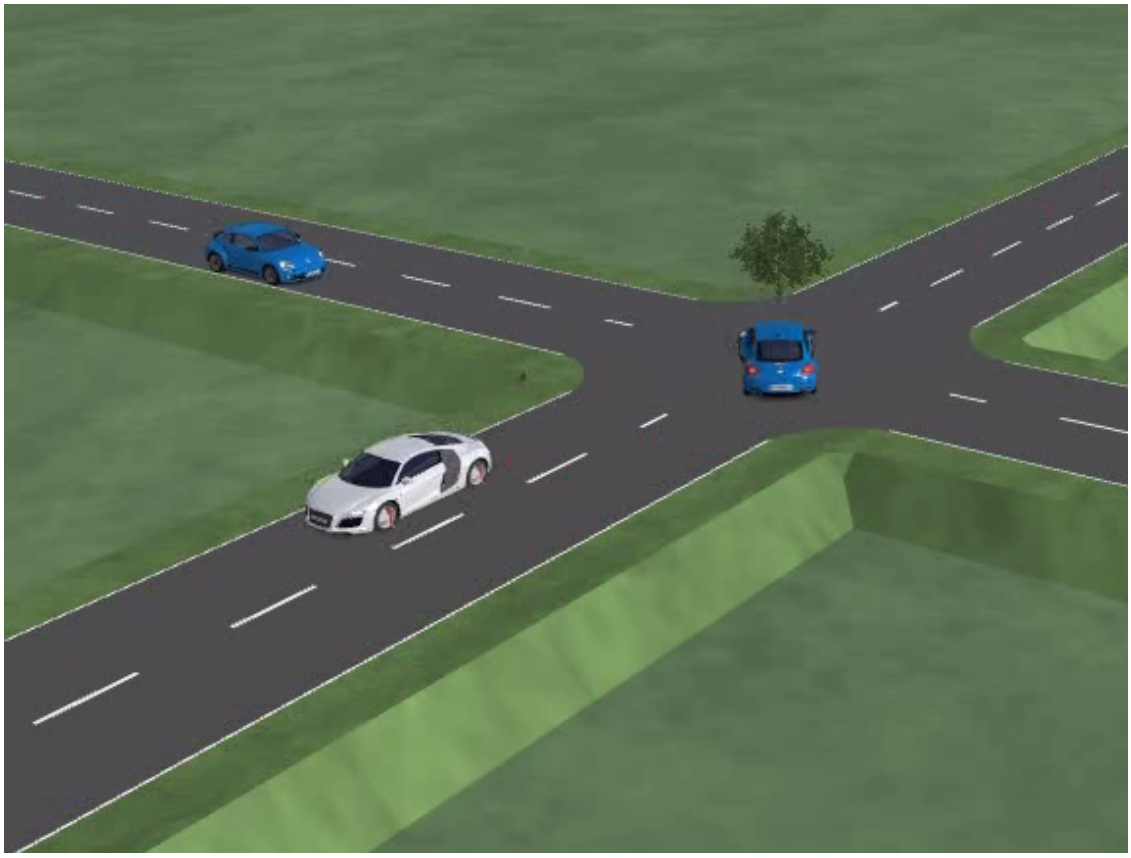
CHALMERS



Validering av algoritmer för korsning av autonoma fordon

Kandidatarbete vid SysCon 2018:EENX15-18-16
Handledare: Nikolce Murgovski
Examinator: Jonas Sjöberg

Carl Svensson
Fredrik Johnsson
Linnea Sundberg
Wilmer Kierkegaard

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2018

CHALMERS

Kandidatarbete vid SysCon 2018:EENX15-18-16

Validering av algoritmer för korsning av autonoma fordon

Carl Svensson
Fredrik Johnsson
Linnea Sundberg
Wilmer Kierkegaard

Handledare: Nikolce Murgovski
Examinator: Jonas Sjöberg

Institutionen för Elektroteknik
SysCon
Chalmers tekniska högskola Göteborg, Sverige 2018

Förord

Tack till Nicolce Murgovski för att han alltid tålmodigt varit beredd att svara på frågor, stora som små. Han har varit till stor hjälp genom hela arbetet och alltid varit entusiastisk till projektet.

Carl Svensson, Fredrik Johnsson, Linnea Sundberg, Wilmer Kierkegaard,
Gothenburg, May 2018

Sammandrag

För att på ett säkert sätt styra autonoma fordon genom korsningar så kan en global regulator användas för att undvika olyckor och maximera genomströmmningen av fordon.

Denna rapport behandlar validering av en regleralgoritm för autonoma fordon i en korsning. En MPC-algoritm har implementerats för att styra fordon på ett säkert sätt genom en korsning. Denna algoritm valideras sedan i en simuleringsmiljö, CarMaker.

Simuleringen visar på att fordonen kör igenom korsningen utan att kollidera. Resultaten indikerar dock att referenshastigheten behöver ses över när fordonen svänger.

Abstract

In order to safely direct autonomous vehicles through an intersection a global controller can be used to avoid accidents and maximize the flow of vehicles through the intersection.

This report treats the validation of a control-algorithm for autonomous vehicles in an intersection. An MPC-algorithm has been implemented to in a safe manner direct vehicles through an intersection. This algorithm is then validated in a simulation environment, CarMaker.

The simulation shows that the vehicles pass thought the intersection without collision. The results indicate that the reference velocity needs to be modified for when the vehicles turn in the intersection.

Symbolförteckning

Tabell 0.1: Tabell över använda variabler

x	Tillståndsvektor i tidsdomän.	$\hat{x}$	Tillståndsvektor i rumsdomän
p	Position	z	Letargi
u	Styrsignal i tidsdomän	$\hat{u}$	Styrsignal i rumsdomän
N_v	Antal fordon	N_s	Antal samplingar
t	Tid	$\hat{t}$	Tid i rumsdomänen
v	Hastighet	$\hat{v}_{ir}$	Referenshastighet
L	Position där fordon åker in i system	H	Position där fordon åker ut ur system
ω	Viktningskonstanter för kostnadsfunktion	ds	Steglängd i rumsdomän

Innehåll

1
Introduktion

Då antalet autonoma fordon på vägarna ökar så ökar även säkerheten på vägarna, och därmed även i korsningar [1]. Statistik från ett flertal europeiska länder visar att 43 % av alla trafikolyckor sker i samband med vägkorsningar [2].

Eftersom det är så stor andel av olyckorna som sker i korsningar är de en av de mest bevakade sektionerna i trafiken. En direkt följd av detta är att ha trafikljus som reglerar flödet. Trafikljus är bra ur säkerhetssynpunkt, eftersom de minskar flödet i korsningen men de skapar i gengäld flaskhalsar i trafiken [3]. Införandet av fler autonoma fordon i trafiken öppnar upp för möjligheten att ta bort trafikljusen och istället styra flödet genom korsningen globalt. Global styrning av trafiken i en korsning gör det möjligt att eliminera flaskhalsar och samtidigt sen säkerheten bibehålls. En global styrning av trafiken i korsningar eliminerar dessutom de olyckor orsakade av den mänskliga faktorn [4].

Fordonen i korsningarna kan styras gemensamt för att minska utsläpp. En global reglering gör det möjligt att ta hänsyn till alla fordon som närmar sig en korsning. Det kan åstadkommas med en Model Predictive Control. Flödet kan då optimeras mot bränsleförbrukning vilket minskar accelerationer och retardationer, vilket är fördelaktigt för både miljön och komforten under färd. Idealt arbetar även regleringen för att minska totala accelerationer och att låta tyngre fordonen, som till exempel buss och lastbil, hålla jämn hastighet genom korsningen. Det sker då på bekostnad av lätta fordon som tvingas sakta ner. På så vis kan bränsleförbrukning minskas ytterligare.

Detta arbete omfattar implementation av algoritmer för trafikflödet genom en korsning och validering av dessa. Algoritmen vidareutvecklas i MATLAB och simuleringen sker i ett program/simuleringsmiljö kallad CarMaker.

1.1 Tidigare arbeten

En algoritm föreslås av J. Lee et al. [5] som centralt koordinerar fordon igenom en korsning genom att lösa ett optimeringsproblem. En formalism utvecklas

som tillåter kollisionsundvikande att formuleras som ett bivillkor i optimeringen.

Säker koordinering av fordon genom en korsning formuleras av N. Murgovski et al. [6] som ett optimalt reglerproblem. Optimeringsproblemet löses för alla möjliga permutationer av sekvenser som fordonen tillåts passera igenom korsningen. Murgovski et al. visar att optimeringsproblemet med krav på kollisionsundvikande, kan formuleras som ett konvext optimeringsproblem med kvadratiska kostnadsfunktioner och linjära bivillkor.

I [7] vidareutvecklar L. Riegger et al. metoden i [6] för att designa en *Model Predictive Controller*, som globalt styr autonoma fordon igenom en korsning. En preliminär evaluering av algoritmen görs i simuleringsmiljön CarMaker via Simulink. Rimligheten av algoritmen i verkliga tillämpningar är avhängig på att den kan exekveras snabbt.

1.2 Syfte

Detta arbete syftar till att validera den regleralgoritm som föreslås av [7] för styrning av autonoma fordon igenom en korsning. Detta görs genom att implementera och vidareutveckla den simuleringsmiljö och MPC-reglator som föreslås av [7]. Denna simuleringsmodell tillåter validering av systemet med en högre grad av realism.

1.2.1 Avgränsningar

- I detta arbete implementeras och valideras en algoritm som koordinerar autonoma fordon genom en korsning. Denna MPC-algoritm kalibreras manuellt för att dess processmodell ska stämma överens med de situationer som simuleras. Ytterligare sätt att kalibrera MPC-algoritmen för de situationer som simuleras behandlas inte.

- *Model Mismatch*, eller *Modellfelanpassning* kan uppstå då sensorbrus är med i simuleringen, eller som ett resultat av felkalibrering då processmodellen och och simuleringsmiljöns mer detaljerade modell divergerar. Detta kan lösas genom att omoptimera den tänkta styrsignalen, alternativt införa en mer detaljerad processmodell. Hur en mer detaljerad processmodell kan konstrueras för att minska denna divergens diskuteras inte.

- Den MPC-algoritm som implementeras kan optimeras för att exekvera snabbare. Ytterligare sätt att optimera algoritmen är med exempelvis

kondensering [8]. Detta görs inte i detta arbete.

- CarMaker tillåter olika typer av fordon att simuleras i olika miljöer. Hur MPC-algoritmen kan kalibreras beroende på vilken typ av fordon, och i vilken miljö de körs i, diskuteras inte.

1.3 Rapportens upplägg

Denna rapport består av kapitlena Introduktion, Teknisk Bakgrund, Metod, Resultat och Diskussion. I Introduktion gavs en kort bakgrund till detta arbete, samt dess syfte. I Teknisk bakgrund ges en kort beskrivning av Model Predictive Control, MPC. Detta följs av en kort beskrivning av den processmodell som används i den MPC-algoritm som valideras. Därefter beskrivs det simuleringsverktyg som har använts för att validera algoritmen. I kapitlet Metod beskrivs de tekniker som möjliggör att lösa det underliggande optimeringsproblemet. Därefter beskrivs hur simuleringsmiljön har konstruerats. I kapitlet Resultat presenteras de simuleringar som har körts. Dessa simuleringar diskuteras sedan i kapitlet Diskussion.

2

Teknisk bakgrund

Algoritmen som valideras bygger på *Model Predictive Control*, eller MPC. Därför ges en kort genomgång av principerna bakom MPC-styrning. Den underliggande processmodell som används av Model Predictive Controller beskrivs också här. Den aktuella MPC-algoritmen genererar styrsignaler genom att lösa ett kvadratiskt programmeringsproblem, QP, varför en kort genomgång av QP presenteras nedan. Den simuleringsmiljö som har använts för validering av algoritmen presenteras också.

2.1 Model Predictive Control

De regleralgoritmer som föreslagits i [6] och [7] bygger på Model Predictive Control. Model Predictive Control är inte en specifik reglerstrategi, utan en mängd strategier som alla utnyttjar en modell av processen som regleras för att genom att minimera en målfunktion ta fram en reglersekvens [9]. Modellen används för att förutsäga framtida skeenden, kallad *horisont*. Denna klass av algoritmer kallas av denna anledning även för Receding Horizon Predictive Control, RHPC.

Fördelarna med Model Predictive Control är många [10]:

- En MPC kan hantera stora tidsförskjutningar, instabila system, icke minfassystem etc.
- MPC-algoritmer är lätta att kalibrera.
- MPC-algoritmer kan hantera strukturella problem (e.g trasiga sensorer), tack vare att algoritmen är adaptiv.

Regulatorns prestanda beror av dess modell av processen som regleras, kallad processmodell [11]. Då processmodellen inte stämmer överens med det verkliga systemet som regleras så kallas detta *model mismatch*, eller modellfelanpassning. Modellfelanpassning kan också uppstå då MPC:n inte är kalibrerad korrekt.

2.2 Processmodell i tidsdomänen

MPC-algoritmen kräver en intern modell av systemet som regleras, kallad processmodell. Modellen för systemet konstrueras som en tillståndsrumsmodell, samplad i tid. $x_i(t)$ kallas för tillståndsvektorn och består av $x_i(t) = [p_i(t)\, \dot{p}_i(t)]^T$, $p_i(t)$ är longidutinell position och $\dot{p}_i(t)$ dess tidsderivata, hastighet. Longitudinell position definieras för varje fordon i dess körbana [6].

Tillståndsekvationerna för processmodellen av ett system av N_v fordon skrivs som

$$\dot{x}_i(t) = Ax_i(t) + Bu_i(t), \forall i \in 1, ..., N_v. \tag{2.1}$$

A kallas för systemmatrisen, B kallas för styrmatrisen och $u(t)$ kallas för styrvektorn [6].

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.2}$$

Tillstånden $x_i(t)$ och styrsignalerna $u_i(t)$ begränsas av

$$x_i(t) \in [x_{imin}(t), x_{imax}(t)], \forall i \in 1, .., N_v \tag{2.3}$$
$$u_i(t) \in [u_{imin}(t), u_{imax}(t)], \forall i \in 1, .., N_v. \tag{2.4}$$

Här är x_{imin} och x_{imax} startposition respektive slutposition för fordon $i \in 1, .., N_v$. Styrsignalerna $u_i(t)$ begränsas av u_{imin} respektive u_{imax}.

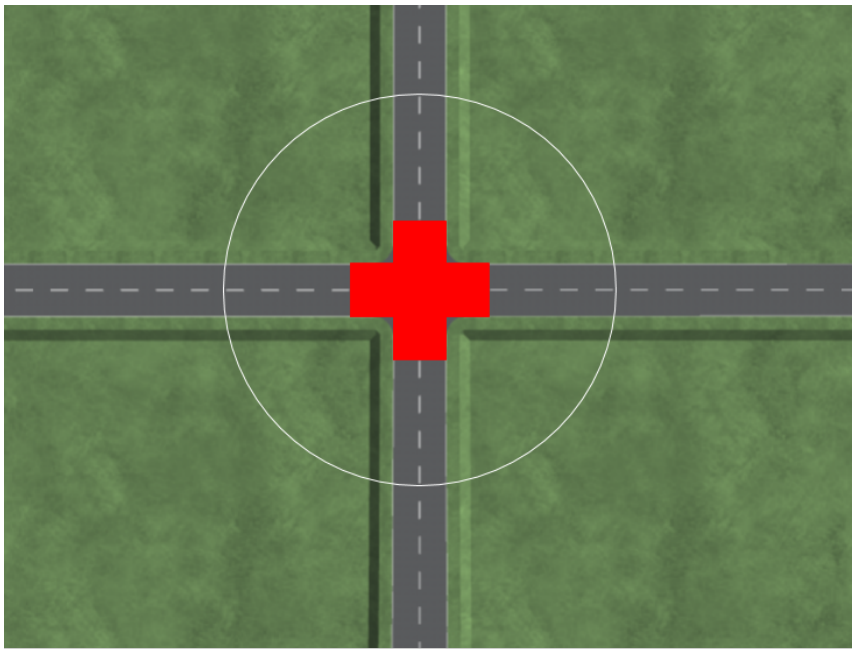
För en säker genomfart genom korsningen skall fordonen, vid ett visst avstånd från korsningen, kontrolleras av ett centralt system. Den vita cirklen i Figur 2.1 illusterar den så kallade *kontrollradien*. Det är när fordonen befinner sig innanför denna cirkel som de kontrolleras av MPC:n. Systemet arbetar enbart mot fordon som befinner sig i denna region. Det röda området i mitten av korsningen i Figur 2.1 visar den *kritiska zonen*. Den kritiska zonen är ett område bestäms genom att bara ett fordon får befinna sig där samtidigt, vilken definieras av ekvation enligt:

$$\zeta_i = \{p_i(t) \in [0, p_{i,f}] \mid p_i(t) \in [L_i H_i]\}. \tag{2.5}$$

Positionen där fordon i kör in i den kritiska zonen respektive lämnar den beskrivs av L_i och H_i. $p_{i,f}$ är slutgiltig longitudinell position för fordon i.

Tiden som ett fordon befinner sig i den kritiska zonen, *ockupansintervallet*, defineras som:

$$G_i(u_i(t)) = \{t \in [0, t_{i,f}] \mid p_i(t) \in \zeta_i\}. \tag{2.6}$$



Figur 2.1: *Illustration över en korsning där det röda området markerar den så kallade kritiska zonen, i vilken enbart ett fordon får befinna sig åt gången. Den vita cirkeln visualiserar den så kallade kontrollradien. Det är när fordonen befinner sig innanför denna kontrollradie som MPC:n börjar kontrollera fordonen.*

Det är under detta tidsintervall som ett fordon befinner sig i den kritiska zonen, $t_{i,f}$ är den slutgiltiga tiden för fordon i. Bivillkoret på den kritiska zonen, eller kravet att två fordon inte får befinna sig i den kritiska zonen samtidigt skrivs som:

$$G_i(u_i(t) \cap G_j(u_j(t)) = \emptyset, \forall i, j, j \neq i \tag{2.7}$$

Detta säger att snittet mellan två fordons ockupansintervall ska vara den tomma mängden, d.v.s att två fordon inte får vara i den kritiska zonen samtidigt. Det fullständiga optimeringsproblemet kan nu skrivas

$$\min_{u_i(t), t_{if}} \sum_{i=1}^{N_v} J_i(x_i(t), u_i(t), \dot{u}_i(t), x_i(t_{if}), t_{if}) \tag{2.8a}$$

med bivillkoren

$$\dot{x}_i(t) = Ax_i(t) + Bu_i(t), \forall i \in 1, ...N_v \tag{2.8b}$$

$$x_i(t) \in [x_{imin}(t), x_{imax}(t)], \forall i \in 1, ..N_v \tag{2.8c}$$

$$u_i(t) \in [u_{imin}(t), u_{imax}(t)], \forall i \in 1, ..N_v \tag{2.8d}$$

$$x_i(0) = x_{i0}, \forall i = 1, ...N \tag{2.8e}$$

$$G(u_i(t) \cap G_j(u_j(t)) = \emptyset, \forall i, j, j \neq i \tag{2.8f}$$

Detta problem är svårhanterligt eftersom det är icke-konvext och blandade heltal. Denna typen av problem lämpar sig dåligt för MPC, då problemet måste kunna lösas många gånger, med så låg tidsåtgång som möjligt för att undvika modellfelanpassning [9] . Avsnitt 3.1 förklarar hur optimeringsproblemet som beskrivs i ekvation (2.8) kan transformeras från att sampla i tidsdomänen till att sampla i rumsdomänen med hjälp av ett lämpligt variabelbyte och på så vis undvika heltatsproblem. Kostnadsfunktionen från ekvation (2.8a) utvecklas och beskrivs explicit i avsnitt 3.2.

2.3 Kvadratisk programmering

För att snabbt kunna lösa det optimeringsproblem som MPC-algoritmen kräver så formuleras optimeringsproblemet i ekvation (2.8) som ett kvadratiskt problem, eller QP. För att optimera ett system minimeras dess tillhörande kostnadsfunktion, vanligtvis beror kostnaden på ett antal ingående variabler. Ett kvadratiskt problem är ett problem vars kostnadsfunktion är kvadratisk och vars variabler har krav som är linjära [12]. Standardform för ett kvadratiskt problem är:

$$\min_X \frac{1}{2} X^T H X + f^T X \tag{2.9a}$$

med bivillkoren

$$A_{in} X \leq b_{in} \tag{2.9b}$$

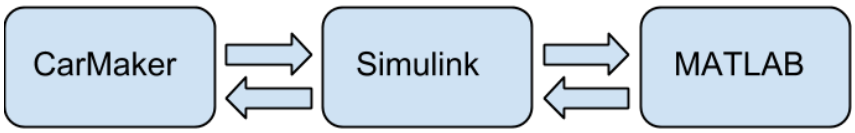
$$A_{eq} X = b_{eq} \tag{2.9c}$$

$$lb \leq X \leq ub \tag{2.9d}$$

Bland de många fördelarna med att formulera problemet som ett kvadratiskt problem är att teorin för lösning av kvadratiska problem är väl utvecklad. Exempelvis kan kvadratiska optimeringsproblem lösas med så kallade *inre punktmetoder*. Dessa metoder är nästan lika effektiva som metoder för linjära optimeringsproblem [13].

2.4 Simuleringsmiljö

MPC-algoritmen valideras med användande av simuleringsverktyget CarMaker, framtaget av IPG AUTOMOTIVE. CarMaker är ett program som används för virtuell testkörning och har detaljerade och verkliga fordonsmodeller med realistiska beteenden [14]. En simuleringsmiljö över en trafikmodell med korsningar och trafikflöden kan konstrueras i CarMaker. Simuleringsmodellen kan



Figur 2.2: *Visualisering över informationsflödet mellan optimeringsalgoritmen i MATLAB, simuleringsmiljön CarMaker och gränsnittet däremellan i Simulink.*

anpassas efter hur vägen ser, vilket väglag det är och hur omgivningen runt omkring ser ut. Det går dessutom att inkludera trafik i simuleringsmodellen. I en simulering av en vägkorsning i CarMaker har varje specifikt fordon en given färdväg och information om hur de ska styras. Det finns även ett verktyg som kallas IPG Movie som används för att visualisera simuleringen. IPG Movie ger en film över simuleringen där resultatet åskådliggörs och kan därför användas som grund för att validera resultatet. När informationen ska överföras mellan CarMaker och MPC-algoritmen används ett gränssnitt i Simulink.

När ett projekt skapas i CarMaker medföljer en av CarMaker fördefinierad Simulinkmodell. Den måste anpassas och byggas om för att kunna användas i det tänkta syftet. Figur 2.2 visar hur informationen ska skickas mellan de olika delarna. Information ska hämtas från CarMaker, via speciella Simulinkblock som kan läsa variabler för de olika fordonens tillstånd. Dessa variabler ska sedan användas i optimeringsalgoritmen i MATLAB. Det optimerade värdet skickas därefter tillbaka till Simulink som i sin tur skickar vidare den till simuleringen i CarMaker.

2.4.1 Begränsningar i CarMaker

CarMaker har parametrar som måste sättas innan en simulering startar, och sen inte kan ändras under simuleringens gång. Det innebär att en del parametrar som hade varit önskvärda att styra externt (från till exempel MPC:n) istället har behövts sättas som initiala värden i CarMaker. Detta rör till exempel vilka starthastigheter fordonen ska och vilken väg de ska ta i korsningen. Det här innebär att det inte går att lägga till och ta bort fordon under simuleringens körning, vilket hade varit önskvärt för en verklighetstrogen simulering där nya fordon dyker upp i korsningen. För att komma runt detta problem kan flera olika fordon placeras ut på på olika positioner på vägarna. Då efterliknar simuleringen slumpmässig miljö med nya fordon som närmar sig korsningen. Men dessa fordon har alltså fördefinierade rutter, hastigheter och startpositioner istället för att slumpas fram under simuleringens gång.

En annan begränsning i CarMaker är att det endast är ett av fordonen , huvud-

fordonet, som simuleras med en detaljerad fordonsmodell. De andra fordonen i simuleringen fungerar som trafikobjekt, vilket innebär att de är som en miljö för huvudfordonet i simuleringen. Trafikobjekten kan röra på sig men de saknar den detaljerade fordonsdynamiken som huvudfordonet har. Trafikobjekten fungerar istället som punktmassor och påverkas inte av bland annat centrifugalkraft vid svängar och skjuvspänningar på däcken. Huvudfordonet kan, till skillnad från trafikobjekten, köra av vägen och krocka med andra objekt i simuleringen. Det är alltså bara simuleringen av huvudfordonet som är verklighetstrogen.

När CarMaker körs i Simulink är simuleringen mycket långsam i jämförelse med en simulering som enbart körs i CarMaker. Detta gäller speciellt om ytterligare tillägg görs i de fördefinierade modellerna [15]. När en prestandakrävande optimering läggs till i ett funktionsblock i Simulink blir simuleringen därför ännu mer tidskrävande.

3
Metod

Algoritmen implementeras först, för att sedan simuleras och valideras i en realistisk miljö med hjälp av CarMaker via ett gränssnitt i Simulink.

Algoritmen som valideras bygger på några matematiska tekniker som låter det underliggande optimeringsproblemet lösas som ett kvadratiskt problem. Detta låter i sin tur optimeringsproblemet lösas på ett effektivt sätt. För att beskriva den implementation som gjorts presenteras därför här de principer som låter optimeringslösare lösa problemet på ett effektivt sätt. Formulering av kostnadsfunktion, tillståndsekvation och bivillkor för optimering redovisas.

Att i praktiken implementera den faktiska algoritmen kantas av några svårigheter och hinder. De algoritmiska principer, problem och lösningsförslag bakom den färdiga solvern, QPsolve i Appendix C, diskuteras.

Simuleringsmiljön utformas i två steg. Dels utformas simuleringsmiljön i CarMaker, och dels kopplas algoritmen ihop med CarMaker via ett gränssnitt konstruerat i Simulink.

3.1 Konvex modellering och sampling i rumsdomänen

Om processmodellen i avsnitt 2.2 samplas i tidsdomänen så kommer optimeringskravet (2.8f), att två fordon inte får befinna sig i den kritska zonen samtidigt uppträda som ett heltalsoptimeringsproblem. Detta problem kan kringås genom att istället sampla i rumsdomänen. Detta görs genom att definera om tillståndsvektorerna som $\hat{x}_i = [t, z]$. Variabeln z, kallad letargi, införs och defineras som $z = 1/v$. Resterande variabel är definerade som rumsderivatan av z kallad z', som är kontrollsignalen, $u = z'$.

För att optimera systemet krävs en diskretisering. Antagen diskretisering resulterar i ekvation (3.1b). Explicit uttryck kommer från ekvation (3.3).

Kraven och kostnadsfuntionerna från ekvation (2.8) kan nu formuleras om i termer av dessa nya variabler. Det fullständiga optimeringsproblemet skrivs i termer av de nya rumsvariablerna som:

$$\min_{\hat{u}_i(p)} \sum_{i=1}^{N_v} \hat{J}_i(\hat{x}_i(p), \hat{u}_i(p)) \tag{3.1a}$$

med bivillkoren:

$$\hat{x}_i(p+ds) = \hat{A}\hat{x}_i(p) + \hat{B}\hat{u}_i(p), \forall i \in 1, ..., N_v \tag{3.1b}$$

$$\hat{x}_i(p) \in [\hat{x}_{imin}(p), \hat{x}_{imax}(p)], \forall i \in 1, ..., N_v \tag{3.1c}$$

$$\hat{u}_i(p) \in [\hat{u}_{imin}(p, z_i(p)), \hat{u}_{imax}(p, z_i(p))], \forall i \in 1, ..., N_v \tag{3.1d}$$

$$\hat{x}_i(0) = \hat{x}_{i0}, \forall i \in 1, ..., N_v. \tag{3.1e}$$

$$\hat{t}_k(E_k) \leq \hat{t}_l(S_l), k = \mathcal{O}_{m,n}, l = \mathcal{O}_{m,n+1} \forall n = 1, ..., N_v - 1 \tag{3.1f}$$

Med den givna formuleringen så blir (3.1f) mer lätthanterlig. Problemet kan lösas som ett kvadratiskt problem för en given korsningssekvens $\mathcal{O}_{m,n}$. För en optimal lösning krävs dock att optimeringen görs för alla möjliga korsningsekvenser. För att göra optimeringen mer effektiv så introduceras en heuristik, som säger att det fordon som är närmast korsningen vid start får passera korsningen först. På så vis behöver optimeringsproblemet endast lösas en gång varje omoptimering.

Den diskretiseringsmetod som har valts är Euler-framåt. Matriserna $\hat{A}$ och $\hat{B}$ i tillståndsekvationen (3.1b) kan på så vis bestämmas.

$$\hat{t}_i(p+ds) = \hat{t}_i(p) + ds \cdot \hat{z}_i(p) \tag{3.2a}$$

$$\hat{z}_i(p+ds) = \hat{z}_i(p) + ds \cdot \hat{u}_i(p) \tag{3.2b}$$

På matrisform:

$$\hat{x}_i(p+ds) = \begin{pmatrix} 1 & ds \\ 0 & 1 \end{pmatrix} \hat{x}_i(p) + \begin{pmatrix} 0 \\ ds \end{pmatrix} \hat{u}_i(p) \tag{3.3}$$

3.2 Kostnadsfunktioner i rumsdomänen

Den kostnadsfunktion som ligger till grund för optimeringen behöver också skrivas om för sampling i rumsdomänen. För att optimeringen ska bli effektiv krävs även att de konstrueras på ett sätt som är kompatibla med kvadratisk programmering. Kostnaden för varje enskilt fordon, indexerat av i, är en summa av tre kostnadsfunktioner .

$$J_i = J_{i1} + J_{i2} \tag{3.4}$$

Den totala kostnaden är en summa över kostnaden för varje enskillt fordon. Den första termen J_{i1} penaliserar avvikelser från en referenshastighet.

$$J_{i1}(\dot{p}_i(t)) = \omega_{i1} \int_0^{t_{if}} (\dot{p}_i(t) - v_{ir}(t))^2 dt \tag{3.5}$$

Kostnadsfunktionen J_{i1} kan uttryckas exakt i letargi, z.

$$J_{i1}(z_i(p)) = \omega_{i1} \int_0^{p_{if}} (\frac{1}{\sqrt{z_i(p)}} - v_{ir}\sqrt{z_i(p)})^2 dp \tag{3.6}$$

För att kunna uttrycka kostnadsfunktionen som en kvadratisk funktion Taylorutvecklas integranden runt referenshastigheten.

$$J_{i1}(z(p)) \approx \omega_{i1} \int_0^{p_{if}} v_{ir}^3 (z_i(p) - \frac{1}{v_{ir}(p)})^2 dp \tag{3.7}$$

Termen J_{i2} penaliserar acceleration [6].

$$J_{i2} = \omega_{i2} \bar{v}_{ir}^5 \int_0^{p_{if}} u_i(p)^2 dp \tag{3.8}$$

Den totala kostnadsfunktioner blir en viktad summa av dessa två termer, summerat över varje fordon.

$$\min_{u_i(p)} \sum_{i=1}^{N_v} J_i(x_i(p), u_i(p)) \tag{3.9}$$

Värt att notera är viktningskonstanterna i ekvationerna (3.7) och (3.8). Dessa uppkommer som ett resultat av att kostnadstermerna har Taylorutvecklats runt en referenshastighet för att omvandla kostnaderna från ett andra ordningens konprogram till ett kvadratiskt program. De viktningskonstanter som uppkommer vid Taylorutvecklingen kan antingen tas som skalärer, eller så kan det göras rumsberoende beroende på ifall referenshastigheten är konstant eller rumsberoende. Detta kan vara användbart om man inte vill ha konstant hastighet genom hela kurvan [6].

3.3 Omvandling till standardiserad kvadratisk form

Den optimeringslösare för kvadratiska program som har valts är MATLABs Quadprog. Quadprog kräver matriser som definierar de tillståndsekvationer, kostnadsfunktion och bivillkor som ligger till grund för optimeringen. Dessa skapas av en rutin, Qpsolve.m, se appendix. För att på ett korrekt sätt generera dessa matriser så skrivs optimeringsproblemet om på standard kvadratisk

form.

En vektor innehållandes alla variabler som optimeras introduceras. Dessa innefattar samtliga tillstånd och styrvariabler för samtliga fordon och samplingar. Denna vektor skrivs som:

$$X_i = \begin{pmatrix} \hat{x}_i & \hat{u}_i \end{pmatrix}^T. \tag{3.10}$$

Där $\hat{x}_i$ är tillståndsvektorn i rumsdomänen och $\hat{u}_i$ är styrvariabeln i rumsdomänen. Dessa variabler förklaras mer utförligt i ekvation (3.3).

Definiera t_{ij} som tiden för fordon i vid sampling j och z_{ij} och u_{ij} för letargi respektive styrvariabeln för fordon i vid sampling j. Dessa återfinns i X enligt följande:

$$t_{i,j} = X_{3 \cdot N_s \cdot (i-1)+4 \cdot j-2} \tag{3.11a}$$

$$z_{i,j} = X_{3 \cdot N_s \cdot (i-1)+4 \cdot j-1} \tag{3.11b}$$

$$u_{i,j} = X_{3 \cdot N_s \cdot (i-1)+4 \cdot j} \tag{3.11c}$$

För en systematisk generering av matriser definieras en radvektor $P_{i,j,var}$. Vektorn har värdet 1 på positionerna som motsvarar variabeln för fordon i, sampling j och var, som är t, z eller u i X. Resterande positioner i vektorn har värdet 0. För att hitta rätt index i matrisen används ekvation (3.11).

3.3.1 Tillståndsekvation och kritisk zon-bivillkor

För tillståndsekvationen och bivillkoret för den kritiska zonen används matrisen $P_{i,j,var}$ som är definierad i slutet avsnitt 3.3.

Villkoret för kritisk zon som beskrivs av ekvation (3.1f) på formen som anges av (2.9b) genereras med $P_{i,j,var}$ där i är ett fordon, j är en sampling och var anger vilken variabel som behandlas.

$$P_{i+1,entryIndex(i+1),t} X \leq P_{i,exitIndex(i),t} X \rightarrow \tag{3.12}$$

$$(P_{i+1,entryIndex(i+1),t} - P_{i,exitIndex(i),t}) X \leq 0 \tag{3.13}$$

Där $entryIndex(i)$ och $exitIndex(i)$ är indexet för samplingen där fordon i åker in i respektive åker ut ur korsningen. Matrisen A_{in} från ekvation (2.9b) byggs upp där varje rad följer av högerledet av ekvation (3.13) där $\forall i \in [1, ..., N_v - 1]$.

För att konstruera tillståndsekvationen som beskrivs av ekvation (3.3) på formen som anges av (2.9c) används $P_{i,j,var}$.

$$P_{i,j+1,t}X = P_{i,j,t}X + ds \cdot P_{i,j,z}X \tag{3.14a}$$

$$P_{i,j+1,z}X = P_{i,j,z}X + ds \cdot P_{i,j,u}X \tag{3.14b}$$

Vilka skrivs om till:

$$(P_{i,j+1,t} - P_{i,j,t} - ds \cdot P_{i,j,z})X = 0 \tag{3.15a}$$

$$(P_{i,j+1,z} - P_{i,j,z}X - ds \cdot P_{i,j,u})X = 0 \tag{3.15b}$$

Matrisen A_{eq} från ekvation (2.9c) byggs upp av rader som motsvarar högerledet i ekvation (3.15) $\forall i \in [1, ..., N_v]$ och $\forall j \in [1, ..., N_s - 1]$.

3.3.2 Kostnadsfunktionen

Matrisen H genereras för att motsvara de kostnadsfunktioner som beskrivs av (2.9a). Detta kan göras på ett systematiskt sätt genom att introducera en delmatris $H_{sub,i}$, som svarar mot kostnaderna för varje fordon i. För att kunna optimera systemet (3.1) behöver kostnadsfunktionen formuleras på formen (2.9a). Matrisen H i (2.9a) kan konstrueras av delmatriser H_{sub}. För ett godtyckligt fordon i skrivs dessa delmatriser som:

$$H_{sub,i} = 2\begin{pmatrix} Q_{1,1} & 0 & 0 & 0 & 0 \\ 0 & Q_{2,1} & 0 & 0 & 0 \\ 0 & 0 & Q_{3,1} & 0 & 0 \\ 0 & 0 & 0 & \cdot & 0 \\ 0 & 0 & 0 & 0 & Q_{3,Ns} \end{pmatrix} \tag{3.16}$$

Där elementen Q bestäms av

$$Q_{1,j} = 0, \forall j \in [1, ..., N_s] \tag{3.17a}$$

$$Q_{2,j} = \omega_v \cdot v_{ref}(i,j)^3, \forall j \in [1, ..., N_s] \tag{3.17b}$$

$$Q_{3,j} = \omega_{dv} \cdot v_{ref}(i,j)^5, \forall j \in [1, ..., N_s] \tag{3.17c}$$

Matrisen H konstrueras i termer av H_{sub} enligt

$$H = \begin{pmatrix} H_{Sub,1} & 0 & 0 & 0 \\ 0 & H_{sub,2} & 0 & 0 \\ 0 & 0 & \cdot & 0 \\ 0 & 0 & 0 & H_{sub,N_v} \end{pmatrix} \tag{3.18}$$

Vektorn f_i som definieras i ekvation (2.9a), innehåller de återstående ickekvadratiska termerna för varje fordon i.

$$f_i = -2\omega_{i1}\bar{v}_{ir}^3 \frac{1}{v_{ir}(p)} \begin{pmatrix} C & \dots & C \end{pmatrix}^T \tag{3.19}$$

Där $C = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$

3.3.3 Startvärde och övre/undre begränsningar

Startvärden för optimeringen sätts med ekvationer på formen som anges i (2.9c). För varje fordon i gäller för den första samplingen.

$$X_i = \begin{pmatrix} t_{start} \\ z_{start} \\ u_{start} \end{pmatrix} \tag{3.20}$$

Begränsningar av tillstånden görs genom formulering av villkor på formen (2.9d). Variabler som inte har en begränsing, till exempel tid sätts dess motsvarande element i lb och ub till godtyckligt stora, negativa respektive positiva värden, som betecknas vbn och vsn. För varje fordon och sampling gäller att:

$$\begin{pmatrix} vsn \\ z_{min} \\ u_{min} \end{pmatrix} \leq X \leq \begin{pmatrix} vbn \\ z_{max} \\ u_{max} \end{pmatrix} \tag{3.21}$$

Vektorerna lb och ub från ekvation (2.9d) består av vänsterledet respektive högerledet från olikheten (3.21).

3.3.4 Glesa matrisproblem

De matriser som genereras då tillståndsvektorn formuleras som (3.10) blir snabbt så pass stor att beräkninga börjar gå märkbart långsamt. De flesta elementen är dock 0. En matris med denna egenskap kallas för en gles matris, och denna typen av struktur går att utnyttja för att göra beräkningarna något snabbare [16]. För att ytterligare göra beräkningarna snabbare så går det att identifiera vilka bitar av A-matrisen som är konstanta över optimeringsiterationer, och på så sätt undvika att behöva generera matrisen i varje optimeringscykel.

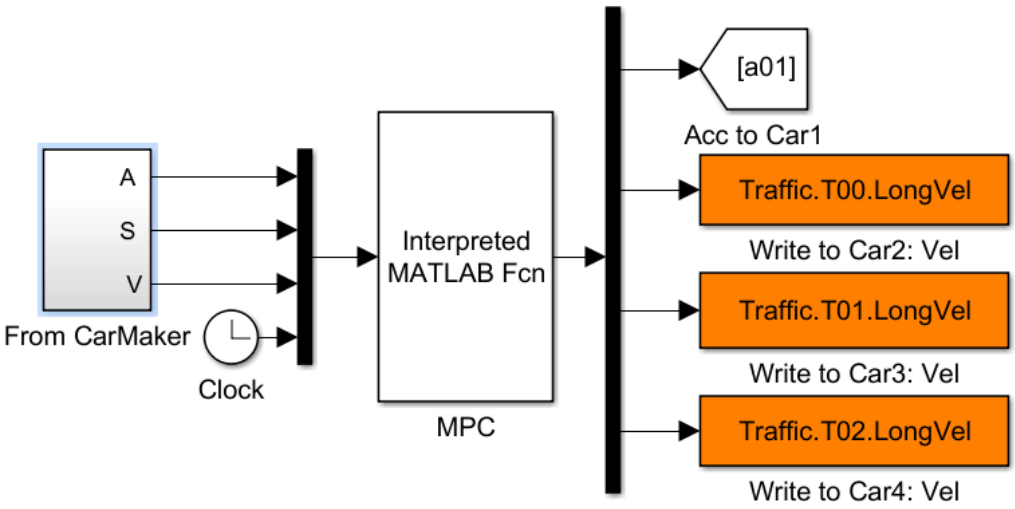
3.3.5 Den färdiga MPC-algoritmen

Den algoritm som har implementerats initialiseras med indata för fordons positioner och hastigheter. Denna data omvandlas till indata för en subrutin.

En optimering genererar utdata bestående av varje fordons tider, letargier och styrsignaler. Dessa signaler är en funktion av position och inte av tid.

Varje fordons tidsvektor är inte linjärt fördelade. Detta, samt att signalerna är rumsberoende gör att det krävs en global klocka mot vilka fordonens tider interpoleras. Denna interpolant används för att skapa styrsignaler som är en funktion av det globala systemets klocka. Interpolanten och fordonens styrsignaler sparas undan.

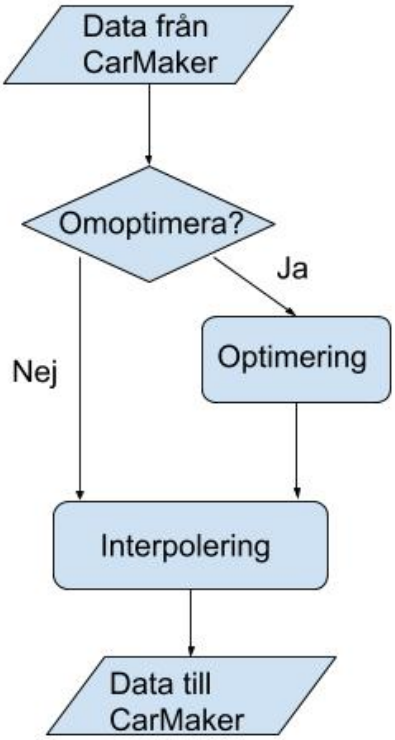
Systemklockan läses av i en loop. De rumsberoende variablerna interpoleras emot systemklockan för att ta fram de korrekta styrsignalera. Dessa skickas sedan vidare till systemet som regleras, i detta fall CarMaker. Omoptimering av styrvariablerna kan göras i varje iteration, eller mer sällan beroende på hur MPC-algoritmen kalibreras, se Figur 3.2. I blocket interpreted MATLAB Fcn görs interpoleringen, och beslut kan tas om huruvida en omoptimering ska göras.



Figur 3.1: *Illustration över vilken information som skickas mellan CarMaker och MPC-algoritmen i MATLAB med hjälp av Simulink.*

Den optimeringsalgoritm som har implementerats utgår från de ekvationer och bivillkor som har presenterats i 3.1 och 3.1 för att generera de matriser som optimeraren har som indata. Den optimerare som har valts för denna implementationen är Quadprog. För att spara exekveringstid så förgenereras matriserna. MATLABs rutiner för glesa matrisproblem utnyttjas. Koden finns

bifogade i appendix C.

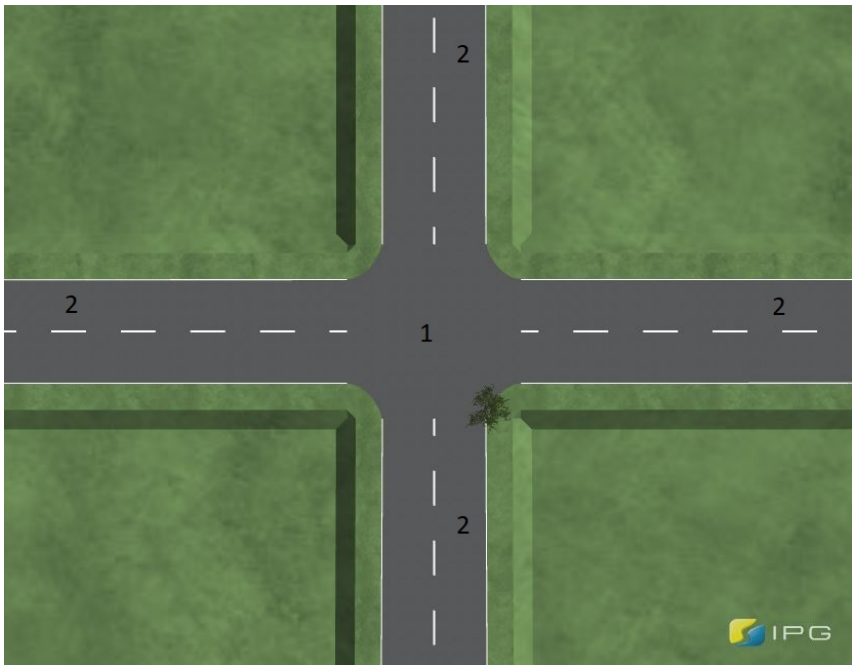


Figur 3.2: Schematisk bild över MPC-blocket som ingår i Figur 3.1

3.4 Utformning av simuleringsmiljö

För att validera algoritmerna används simuleringsprogrammet CarMaker version 5.1.4. I CarMaker finns detaljerade modeller över fordon, vägar och trafik, vilket ger en möjlighet att utföra realistiska simuleringar [17].

För att kunna simulera vägkorsningssituationen krävs en korsning för fordonen att köra i modellerad i CarMaker. En korsning i CarMaker byggs enligt exempelkoden nedan. Första raden definierar var knytpunkten, alltså mittpunkten av korsningen ska befinna sig. Från knutpunkten utgår sedan armar. Hur många armar korsningen har bestäms genom att ange med vilka vinklar de är anslutna till knutpunkten. CarMaker tillåter mellan två och sex armar. Korsningen som används i detta fallet är en fyrvägskorsning, med räta vinklar mellan de fyra armarna, se Figur 3.3. För att sedan bygga den faktiska vägen som ansluts till korsningen används olika så kallade länkar. Varje del av vägen består av en länk, ett exempel på hur länkar kan definieras kan även det ses i



Figur 3.3: *Bild över en korsning i CarMaker, där 1 markerar mittpunkten och 2 markerar armarna.*

exempelkoden nedan. Länkarna ansluts till knytpunkten och har bland annat parametrar som hur lång den är, om den ska vara rak eller hur många grader den ska svänga. Det är också i länkarna som det definieras hur de olika filerna ska se ut och om vägarna ska ha några vägmarkeringar eller vägskyltar.

```
Junction.0.Knot = 150.0000 150.0000 0
Junction.0.ArmAlpha = 0.0000 90.0000 180.0000 270.0000
Junction.0.ArmLength = 5.0000 5.0000 5.0000 5.0000
Junction.0.MainArms = 0 2

Link.0.Junctions = 0 0 -1 -1
Link.0.Node0 = 0.0000 0.0000 0.0000 0.0000
Link.0.Seg.0.Type = Straight
Link.0.Seg.0.Param = 150.000 0.0000 0.0000 0.0000 0.0000 0.0000
    0.0000 0.0000
Link.0.RoadMarking.0 = 150 3 0 3 0 0 0.12 0 2 0 0 2 2 1 1 2 ""
Link.0.RoadMarking.1 = 0 2 0 1 0 2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.0.RoadMarking.2 = 0 2 0 1 0 -2 0.12 0 1 0 0 5 2 1 1 2 ""

Link.1.Junctions = 0 1 -1 -1
Link.1.Node0 = 150.0000 -150.0000 0.0000 0.0000
Link.1.RST = contryroad
Link.1.Seg.0.Type = Straight
Link.1.Seg.0.Param = 150.0000 0.0000 0.0000 0.0000 0.0000
```

```
    0.0000 0.0000 0.0000
Link.1.RoadMarking.0 = 150 3 0 3 0 0 0.12 0 2 0 0 2 2 1 1 2 ""
Link.1.RoadMarking.1 = 0 2 0 1 0 2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.1.RoadMarking.2 = 0 2 0 1 0 -2 0.12 0 1 0 0 5 2 1 1 2 ""
```

När vägarna är definierade skapas de olika färdvägarna genom korsningen. Färdvägar skapas genom att de olika länkarna som ska ingå i respektive färdväg anges, alltså vilken väg in och vilken väg ut ur korsningen som motsvarar färdvägen. Vidare anges hur färdväggarna ska se ut på respektive länk. Det specificeras till exempel vilken fil fordonet ska köra i på de olika färdvägarna på respektiva länkar. Färdvägarna tilldelas olika index beroende på vilka kombinationer av in- och ut-farter de består av. I detta fall finns det tolv olika färdvägar, en som går åt höger, en som går åt vänster och en som går rakt fram för vardera av de fyra infartsvägarna. De olika färdvägarna och vilka infartsvinklar och utfartsvinklar de motsvarar finns i Tabell 3.1.

Tabell 3.1: De olika färdvägarna och vilka infartsvinklar och utfartsvinklar de motsvarar.

Infartsvinkel	Utfartsvinkel	Sväng	Länkar	Färdväg
0°	90°	Höger	1-2	4
0°	180°	Rakt	1-3	5
0°	270°	Vänster	1-0	3
90°	0°	Vänster	2-1	7
90°	180°	Höger	2-3	8
90°	270°	Rakt	2-0	6
180°	0°	Rakt	3-1	10
180°	90°	Vänster	3-2	11
180°	270°	Höger	3-0	9
270°	0°	Höger	0-1	0
270°	90°	Rakt	0-2	1
270°	180°	Vänster	0-3	2

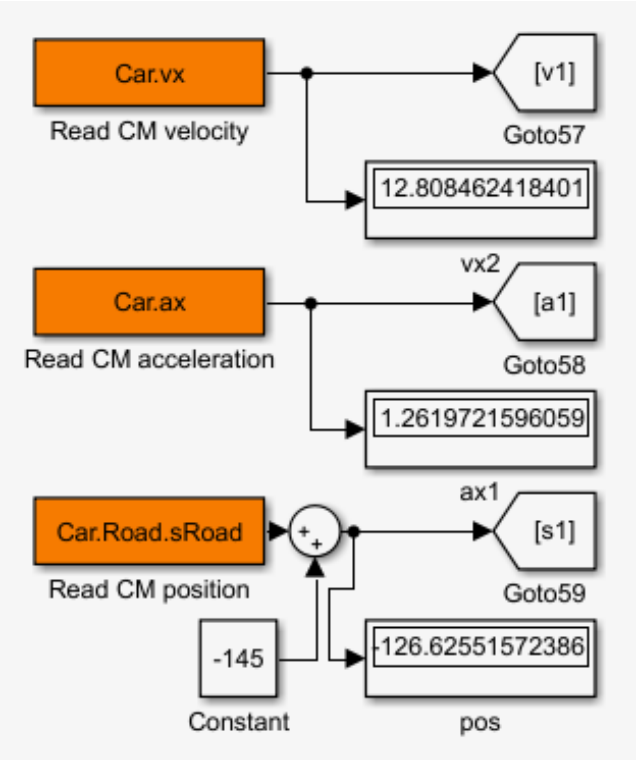
Efter att färdvägarna är definierade ska ett fordon, en manöver och en färdväg för fordonet bestämmas i CarMaker [18]. Om ytterligare trafik läggs till i CarMaker är det endast huvudfordonet som är en realistisk modell för ett fordon. Övriga fordon, kallade trafikobjekt, är punktmassor och agerar som miljö/omgivning till huvudfordonet. Alla fordon behöver parametrar som starthastighet, startposition, vilken rutt den ska följa och vilken sida på vägen den ska befinna sig. Huvudfordonet behöver också veta vilken startväxel den ska ha, för att minska på den fördröjingen det innebär om fordonet ska växla till lämplig växel innan det kan börja gasa enligt anvisningarna från MPC:n.

För att använda CarMaker för att validera optimeringsalgoritmerna måste CarMaker kopplas samman med algoritmen i MATLAB-scripten. Detta görs via ett gränssnitt i Simulink.

3.5 Integration av simuleringsmiljön och optimeringsalgoritmen

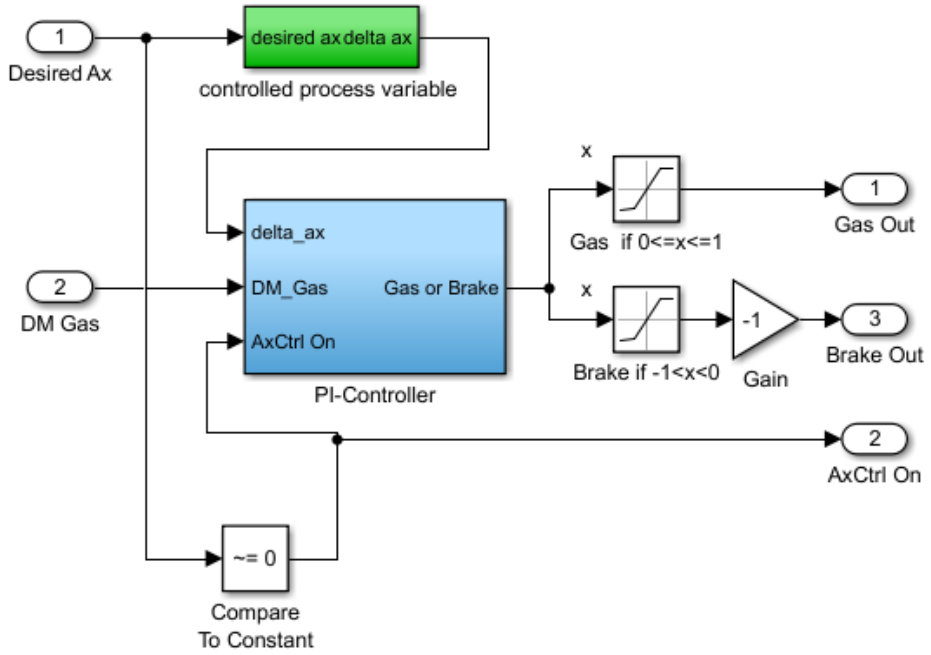
CarMaker är kompatibelt mot MATLAB (Version r2016b används i detta fall) via Simulink och för att använda styrsignaler från MATLAB till CarMaker används Simulink.

Figur 3.4 visar hur hastighet, acceleration och position hämtas från CarMaker till Simulink. I samma Figur framgår det att positionen från CarMaker adderas med en konstant (-145) innan den skickas vidare. Detta görs eftersom CarMaker anser att nollpunkten för varje fordon är i där färdvägen startar. För att få få samma nollpunkt för alla fordon förflyttas nollpunkten in till mitten av korsningen. Dessa värden är inparametrar till MPC-funktionen som visas i Figur 3.1. Optimeringen sker och börvärden för hastigheter skickas tillbaka till CarMaker. I Simulinkmodellen i Figur 3.1 finns även en klocka som inparamter till MPC-funktionen. Denna klocka används för att interpolera värdena, så att bilarna får hastighet och acceleration som motsvarar tidpunkten.



Figur 3.4: *Bild över hur information hämtas från CarMaker till gränssnittet i Simulink.*

Accelerationen för huvudfordnet regleras genom Simulink via en PI-regulator som tar in den önskade accelerationen och ger ut ett värde för hur mycket gas och broms som ska ges till fordonet, se Figur 3.5. Det är bara huvudfordonet som fungerar som ett verkligt fordon, därför är det bara detta fordon som har en PI-regulator för att reglera accelerationen. De övriga fordonen i simuleringen styrs genom att den önskade hastigheten anges till dem.



Figur 3.5: *Simulinkmodell över regulatorn som styr huvudfordonets gas och broms med en önskad acceleration som indata.*

3.6 Simulering

Simuleringsmiljön kopplas ihop med MPC-algoritmen. Därefter körs simuleringar. Datan från simuleringar analyseras sedan för att se huruvida MPC-algoritmen fungerar på önskat sätt. Signaler från MPC-algoritmen jämförs med signaler från CarMaker. Simuleringen inspekteras även visuellt med hjälp av visualiseringsverktyget IPG Movie.

4

Resultat

MPC-algoritmen har utvecklats och validerats i simuleringsmiljön CarMaker. Den implementerade MPC-algoritmen anger med vilka hastigheter de olika fordonen ska närma sig korsningen för att undvika kollision. Resultatet illustreras i olika grafer och visualiseringar över egenskaper hos fordonen som indikerar att kollisioner undviks.

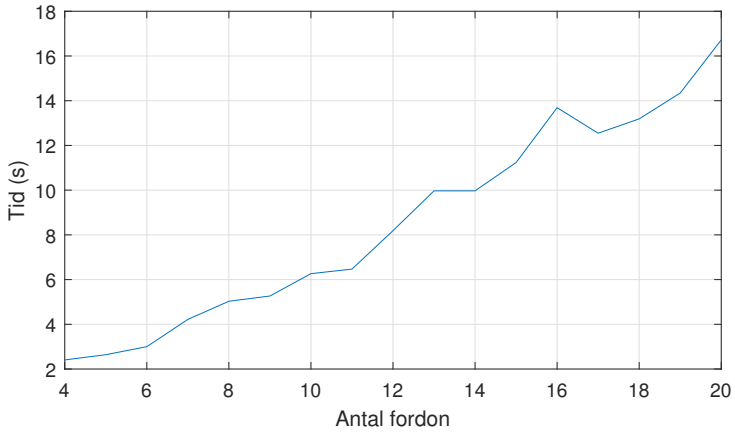
Simuleringen sätts upp med parametrar. Kontrollradien är satt till 145 m, samplingsavståndet, ds är 1 m, den kritiska zonens radie är 7.5 m och antal fordon, Nv är 4. Optimeringen sker en gång och då med startvärden från Tabell 4.1.

Tabell 4.1: Startvärden för fordon i MPC.

Avstånd till kritiska zonen (m)	Starthastighet ($\mathrm{m\,s^{-1}}$)	Startacceleration ($\mathrm{m\,s^{-2}}$)
$\begin{pmatrix}137.5\\137.5\\137.5\\137.5\end{pmatrix}$	$\begin{pmatrix}12\\12\\12\\12\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\\0\end{pmatrix}$

4.1 Implementation av optimeringsalgoritmen

En MPC har implementerats användandes av optimeringslösaren Quadprog. Tidsåtgången för optimeringen ställs mot antalet fordon i Figur 4.1. Det är en tydlig ökning i tidsåtgånen när fler fordon ingår i optimeringen.



Figur 4.1: *Tidsåtgång för optimering med avseende på antal fordon som ingår i optimeringen. Testet är kört på en dator med Intel(R) Core(TM) i5-5200U CPU på 2.20 GHZ och 8 GB ram.*

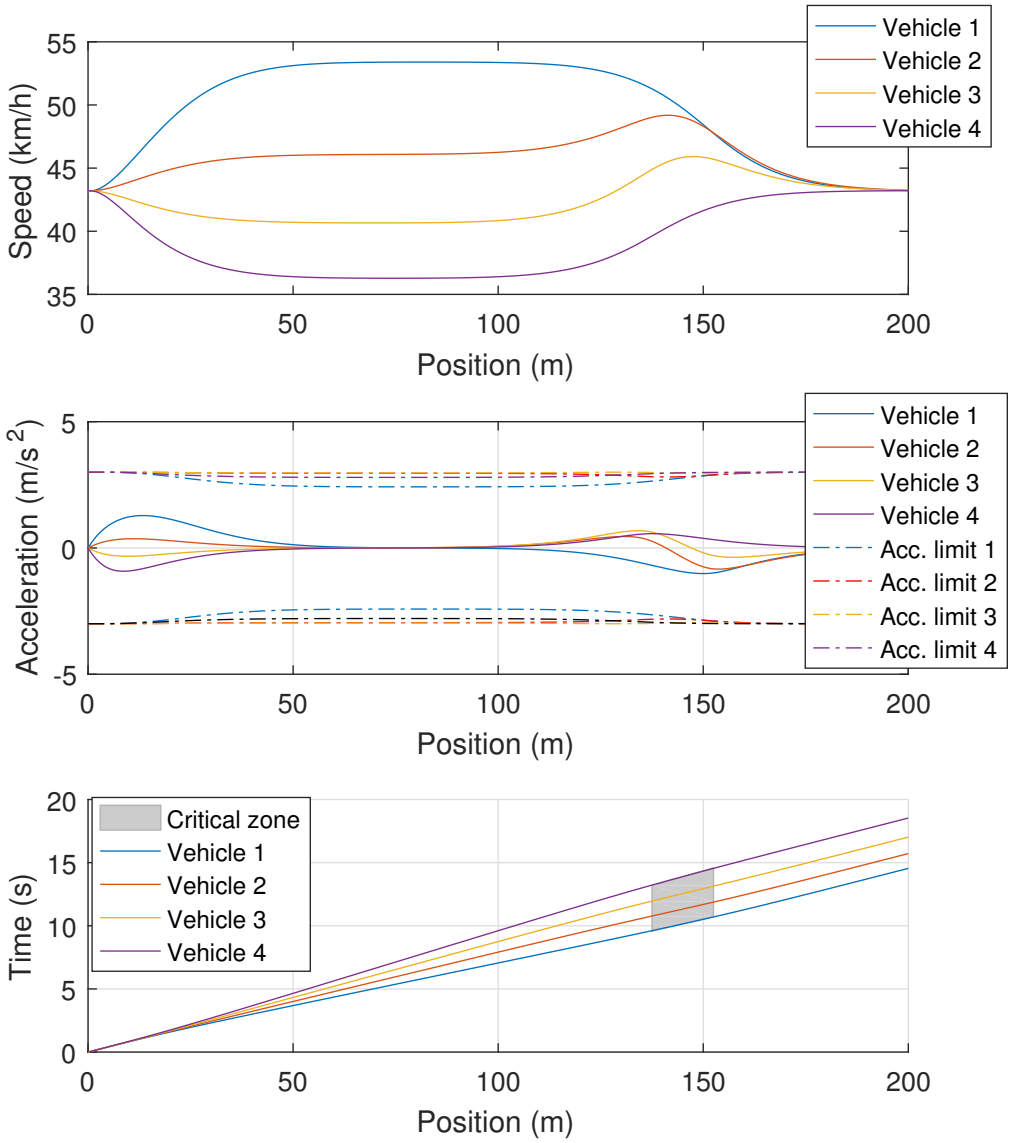
När de angivna värdena i Tabell 4.1 används i optimeringsalgoritmen fås resultatet som presenteras i Figur 4.2. Det är denna data som sedan används i simuleringen och som är jämförs mot de faktiska värdena i avsnitt 4.2.

4.2 Validering av MPC-algoritmen

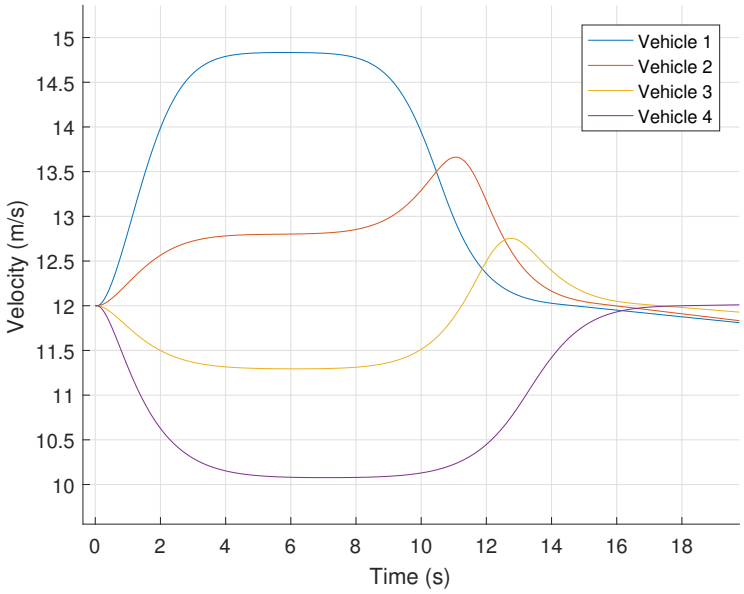
MPC-algoritmen har validerats mot CarMakers simuleringsmiljö. I CarMaker sätts huvudfordonets växel till 3, CarMaker startar fordon förinställt på tomgång och det leder till att fordon retarderar intialt om starthastgiheten sätts till $12\,\mathrm{m\,s^{-1}}$.

Resultatet indikerar att flödet genom korsningen ökar, samt att fordonen undviker varandra på ett säkert sätt. Resultatet illustreras i Figur 4.3, 4.5, 4.7 och 4.8.

Det framgår av Figur 4.3 att de två första fordonen accelererar för att så fort som möjligt ta sig förbi korsningen och den kritiska zonen. Acceleration sker initialt och sedan bibehålls en konstant hastighet tills den kritiska zonen är passerad. Därefter återgår hastigheten till referenshastigheten. De andra två fordonen bromsar istället in för att undvika kollision med de första två.



Figur 4.2: *Grafer över den optimerade datan för fyra fordon som åker igenom en korsning. Alla bilar håller sig till inom sina accelerationsgränser och inga bilar befinner sig i den kritiska zonen samtidigt.*



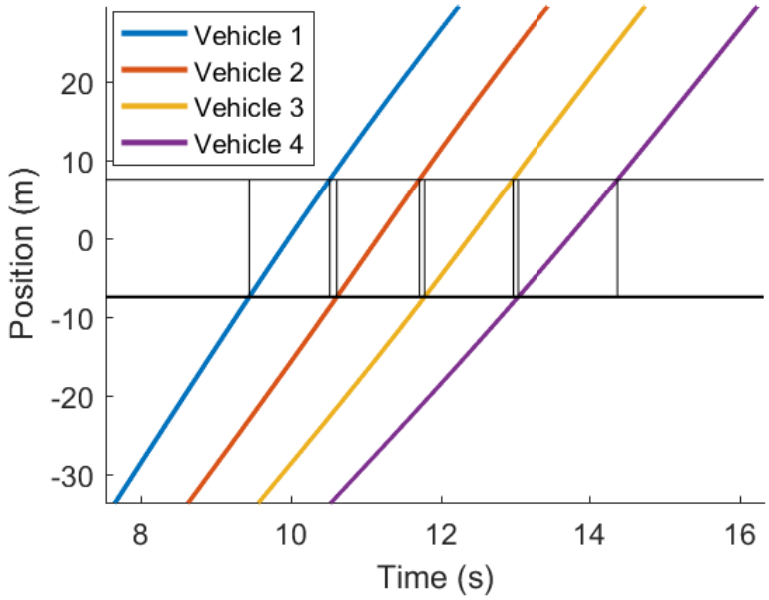
Figur 4.3: *Graf över fyra fordons hastigheter, där datan är hämtad från CarMaker och samplad i tidsdomänen*

Figur 4.4 visar ett fordon med verkliga egenskaper som svänger i korsningen. Fordonet klarar inte av ta ta kurvan med den använda referenshastigheten utan kör i diket.

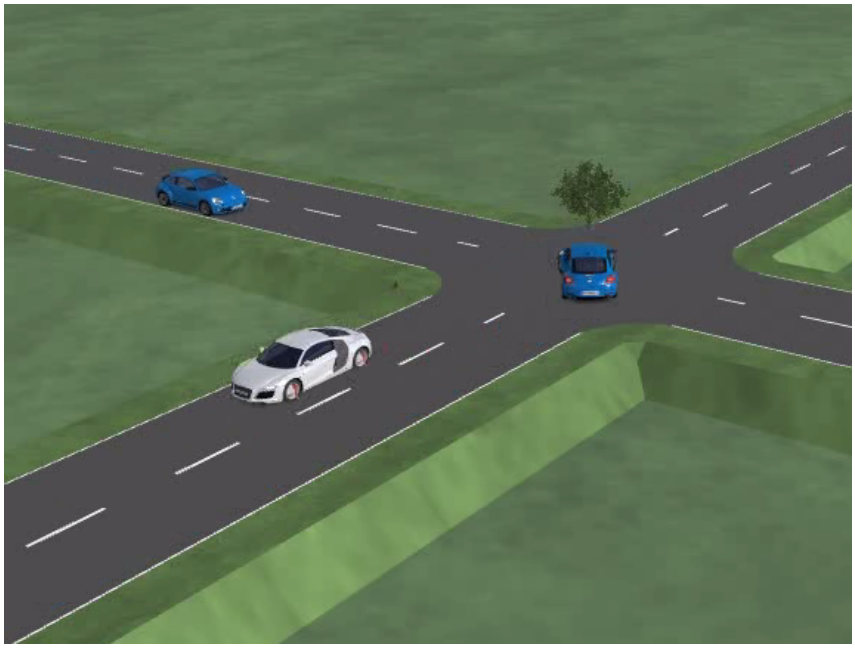
När algoritmen valideras i CarMaker fås grafen över de olika fordonens positioner runt den kritiska zonen i Figur 4.5. De horisontella linjerna visar var gränsen för den kritiska zonen är och de vertikala linjerna visar vid vilken tidpunkt de olika fordonen kör in i och lämnar den kritiska zonen. Eftersom föregående fordon hinner lämna den kritiska zonen innan nästa kör in i den så är säkerhetskravet att endast ett fordon får befinna sig i den kritiska zonen samtidigt uppfyllt. Genom att använda verktyget IPG Movie för att evaluera resultatet fås bilden i Figur 4.6. Denna bilden föreställer tre fordon i en korsning, som inte befinner sig i den kritiska zonen samtidigt, och därmed visar även denna bild på att de kör igenom korsningen säkert.

Huvudbilens acceleration är minimalt förskjuten i tid relativt den acceleration som begärs av MPC-regulatorn. Detta illustreras i Figur 4.7. Skillnaden i den önskade och verkliga accelerationen medför en skillnad i den önskade och verkliga hastigheten. Detta illusteras i Figur 4.8.

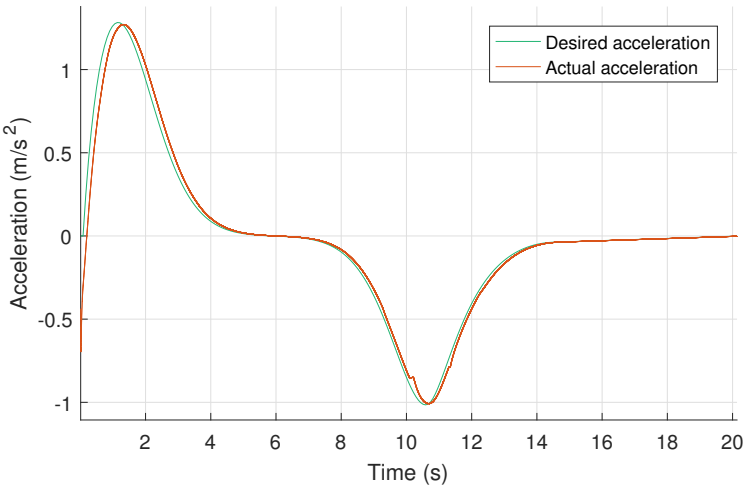
Figur 4.4: Bild över när ett fordon kör i diket i en sväng i CarMaker.



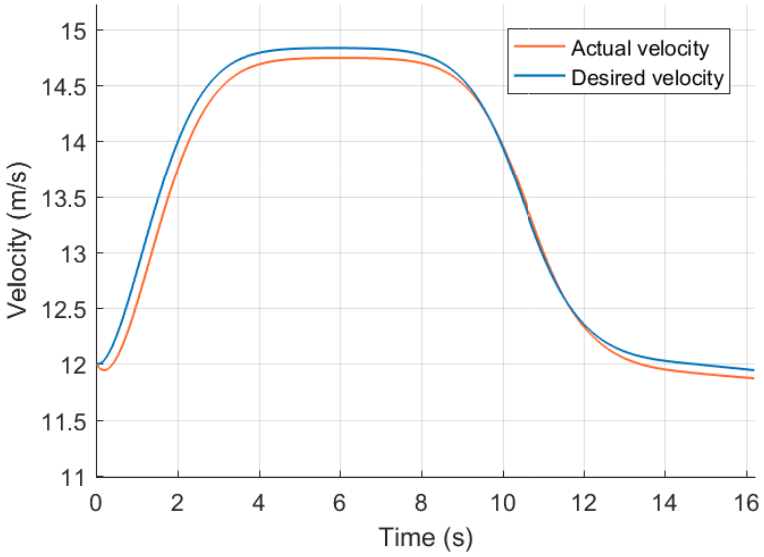
Figur 4.5: *Graf över fyra simulerade fordons position, där det första fordonet är ett fordon med fysikaliska egenskaper. Positionsvärdet 0 motsvarar mitten av korsningen. De horisontella, svarta linjerna visar var gränsen för den kritiska zonen är. De vertikala linjerna visar när fordonen kommer in i, och lämnar den kritiska zonen.*



Figur 4.6: *Bild av bilar, styrda av MPC:n, som kör igenom korsningen utan att krocka. De blå fordonen är trafikobjekt och den vita bilen har fysikaliska egenskaper.*



Figur 4.7: *Önskad acceleration och verklig acceleration för ett fordon med fysikaliska egenskaper mot tiden. Där grön kurva motsvarar önskad acceleration och röd kurva motsvarar verklig acceleration.*



Figur 4.8: *Graf över önskad och verklig hastighet för huvudfordonet i CarMaker*

5
Diskussion

En MPC-algoritm för kontroll av autonoma fordon i en vägkorsning har validerats med hjälp av simulering i CarMaker. Simuleringar visar att algoritmen fungerar väl. Fordonen befinner sig inte i den kritiska zonen samtidigt, och en hög genomströmning av fordon erhålls.

5.1 Validering av MPC-algoritmen

Figur 4.7 och 4.8 visar att diskrepansen mellan önskad och verklig acceleration och hastighet är nästintill obefintilig. Detta tyder på regulatorn som använts i Simulink är tillförlitlig. Det framgår tydligt i graferna att kurvorna följer varandra med stor precision. För vidare arbete kan det dock vara relevant att titta efter detta eftersom regulatorer generellt är mottagliga för sensorbrus [19].

I Figur 4.8 framgår det att fordonet med fysikaliska egenskaper har en starthastighet på 12 m/s som initialt minskar lite. Detta kan förklaras med att regulatorn inte ger någon gas till fordonet i första iterationen. Skillnaden är väldigt liten vilket måste ses som ett lyckat resultat. Latensen är inte tillräckligt stor för att fordonen ska kollidera, vilket visas i Figur 4.5 och 4.6. Figur 4.5 visar fyra fordon och deras position i förhållande till den kritiska zonen. Figuren visar att det tidigare fordonet hinner lämna den kritiska zonen innan nästa fordon befinner sig i den. Figur 4.6 visar samma situation fast med hjälp av verktyget IPG Movie.

Figur 4.5 visar även att tiden mellan att den föregående bilen lämnar den kritiska zonen och att nästa kör in i den är minimal. Detta innebär att flödet igenom korsningen är bra.

Tidigare valideringar med hjälp av simulering har inte behandlat en korsning där fordon tillåts svänga. Resultatet i avsnitt 4.2 indikerar att om ett fordon håller för hög hastighet i en kurva kan fordonet inte hålla sig till färdrutten, utan kör av vägen. Detta kan åtgärdas genom att sänka referenshastigheten, vilket dock kan sänka prestandan på genomströmningen i systemet. Ett förslag

är att göra referenshastigheten positionsberoende, så att fordon tillåts hålla en hög hastighet fram till korsningen för att sedan sakta in vid svängen.

5.1.1 Modellmissanpassning

Vid en stor skillnad mellan MPC-algoritmens interna systemmodell och det verkliga systemet sker en gradvis missanpassning, så att styrsignalen som genereras inte ger det resultat som förväntas. Vid simulering i CarMaker utan sensorbrus är missanpassningen minimal, förutsatt att MPC:ns interna parametrar är välkalibrerade. Vid simulering med sensorbrus förväntas missanpassningen bli mer betydande. En lösning på det är att omoptimera systemet med högre frekvens. Detta kräver att optimeringen är snabb, eftersom det annars blir tidskrävande. En tidskrävande optimering innebär att omoptimering inte kan genomföras i realtid.

5.2 Implementation av optimeringsalgoritmen

En optimeringsimplementation av algoritmen har utvecklats på kvadratisk form. Figur 4.1 visar sambandet mellan tidsåtgång och antal fordon i optimeringen. Ett tydligt ökande hos tidsåtgången fås vid ökande antal fordon. Optimeringen för ett stort antal fordon är inte tillräckligt snabb för en implementation av en MPC med hög omoptimeringsfrekvens.

5.3 Framtida arbete

I den akutella versionen av CarMaker (5.1.4) tillåts inte skapande och elimnation av fordon under tiden som en simulering körs. I en simulering vars syfte är att efterlikna verkligheten så bör fordon ha möjlighet att skapas och elimineras fritt och helt slumpmässigt. Därefter skall de kunna styras globalt vid kontrollradien. Men det är alltså begränsat i CarMaker. Detta leder till att antal fordon i en given simulering hålls konstant. En lösning som är formulerad men inte genomförd är att skapa en depå av fordon som kan placeras på godtycklig plats i simuleringen. Då fordon i är aktuell för att användas flyttas den till korrekt plats. När den är använd flyttas den tillbaka till depån.

Enbart huvudbilen har verkliga fysiskaliska egenskaper. I en bättre simulering skulle alla fordon bedömas efter verkliga parametrar på respektive fordon. Något de inte gör, trafikfordonen optimeras och simuleras enbart som punktmassor med en hastighet. Detta har i det här arbetet inte utvecklats ytterligare, främst för att det är svårt att implementera i CarMaker. Det kan dock vara

relevant för framtida arbete och nämns därför här.

5.3.1 Ytterligare realisering av modellsystemet

Optimeringen behandlar alla vägar som lika långa, oberoende om fordonen svänger vänster, höger eller fortsätter rakt fram. Generellt så innebär en vänstersväng en längre körväg i korsningen än att köra rakt fram eller göra en högersväng. Att köra rakt fram innebär i sin tur en längre körväg i korsningen än att svänga höger. Diskrepansen mellan optimeringen och det verkliga systemet kan innebära problem vid simulering och bör åtgärdas vid framtida arbete utveckling av systemet.

Referenshastigheten sätts som ett konstant värde för alla punkter i korsningen. I en mer verklighetstrogen simulering bör hastigheten vara olika beroende på olika fysiska parametrar. Till exempel bör hastigheten vara lägre under en kraftig sväng.

5.3.2 Utveckling av MPC-algoritmen

Rapporten ämnar till att beskriva valideringen av en specifik algoritm. Vissa aspekter i den behandlade algoritmen kan utvecklas till att skapa både snabbare optimering och ett mer verklighetstroget system.

I den aktuella algoritmen antas att alla fordon kontrolleras inom en given kontrollradie. Generellt är det överflödigt att reglera fordonet centralt, efter att de lämnat den kritiska zonen, då fordonen antas vara autonoma.

En av algoritmens centrala villkor är att maximalt ett fordon får befinna sig i den kritiska zonen vid alla tidpunkter. Krockar i korsningar kan dock endast uppstå då två fordon faktiskt korsar färdväg alternativt befinner sig på samma färdväg. Ett sätt att identifiera vilka färdvägar som korsar varandra kan utveckla systemet och låta fordon som inte korsar varandras färdväg befinna sig i den kritiska zonen samtidigt. Om denna utveckling implementeras bör flödet genom korsningen öka betydligt. Det finns en risk att algoritmen är mer krävande att optimera, vilket kan leda till långsammare optimering.

5.3.3 Utveckling av optimering

Det finns fortfarande problem som inte har addresserats i detta arbete. Exempelvis så skulle optimeringsalgoritmen kunna göras snabbare genom att identifiera vilka matriselement som är konstanta mellan iterationer, och vilka som varierar. Med vetskap om detta så skulle omgenerering av matriser kunna

undvikas och på så vis erhålla en algoritm som är avsevärt snabbare.

Valet att använda Quadprog skulle också kunna undersökas. Det är en av många solvers för kvadratisk optimering. Huruvida det är den bästa ekvationslösaren för denna tillämpning behandlas inte i denna rapport.

5.3.4 Optimering i realtid

För att optimera i realtid krävs att tillstånden för de olika fordonen kan hämtas och optimeras om. Systemet kan optimeras om vid varje iteration av en loop. Detta ställer hårda krav på MPC-algoritmen. Om optimeringen inte körs tillräckligt fort så kommer resultat inte att hinna erhållas tillräckligt fort för att resultatet ska vara användbart. För fortsatta arbeten skulle en undersökning av olika kvadratiska algoritmers gångbarhet därför krävas.

6
Samhälleliga och etiska aspekter

En av de samhälleliga och etiska aspekterna som bör beaktas i detta projekt är säkerheten. Eftersom den MPC-styrda korsningen bygger på att fordon som färdas i den är autonoma är även de sociala och etiska aspekterna för autonoma fordon intressanta att diskutera. Idag är den mänskliga faktorn orsaken till 90-95% av alla olyckor [20]. Autonoma fordon torde reducera antalet olyckor och dödsfall. Bara i USA skulle nästan 30 000 liv per år kunna sparas med hjälp av autonoma fordon [21]. Baksidan av införandet av autonoma fordon är att fordonet ibland kommer ställas inför situationer där det till exempel måste välja mellan att köra över en gångtrafikant eller offra sig själv och sina passagerare [22].

En annan vinning med autonoma fordon är att fler får en större rörelsefrihet. Till exempel rörelsehindrade, äldre eller andra personer med svårighet att köra bil får lättare att förflytta sig, vilket innebär att de får en större möjlighet att delta i samhället och de sociala tillfällena som finns [23]. Det finns även sociala nackdelar med autonoma fordon i samhället. Ett exempel är att det är många lastbils-, buss- och taxichaufförer som blir arbetslösa när fordonen blir autonoma. Men det kommer istället finnas ett större behov av andra typer av arbeten, till exempel tekniskt underhåll [24].

Autonoma fordon håller en jämnare hastighet än manuellt körda fordon [23]. När ett fordon accelererar och retarderar mindre blir bränsleförbrukningen och därmed utsläppen lägre, vilket gynnar miljön. Ett sätt att minska ytterligare på acceleration och retardation på vägarna är att använda den typ av MPC-styrda korsningen som behandlas i denna rapport. I en MPC-styrd korsning anpassas hastigheterna för att få ett så bra flöde som möjligt genom korsningen och till skillnad från i en trafikljusstyrd korsning kommer fordonen inte att behöva stanna. Att fordonen slipper stanna kommer också vara tidsbesparande för resenärerna, vilket även det kan ses som en samhällelig vinning.

Ytterligare säkerhetsaspekter att beakta i en MPC-styrd korsning listas nedan.

- Det kan vara ett problem om optimeringsalgoritmen anger högre hastigheter än vad som är säkert, i till exempel en kurva. Säkerhetsproblemet

kan lösas genom att sätta begränsningar på hastigheten i optimeringsprocessen.

- En annan säkerhetsaspekt att ta hänsyn till i optimeringen är avståndet mellan fordonen. Ju närmare fordonen befinner sig varandra, desto bättre kommer flödet att flyta, men det ökar istället säkerhetsrisken.

- Situationen där ett autonomt fordon ställs inför dilemmat att köra över en fotgängare eller offra sig självt kan även uppkomma i anslutning till en korsning. Därför är det viktigt att tänka över även sådana situationer i utformningen av en MPC-algoritm, och att fordonens interna säkerhetssystem fortfarande fungerar trots att hastigheten styrs globalt av MPC:n.

- Flödet genom korsningen kan bli känsligt för strömavbrott och andra systemfel. Det finns en risk för modellmissanpassning eller att MPC:n inte inte skickar ut någon information alls, vilket kan innebära att flera fordon befinner sig i den kritiska zonen samtidigt och därmed riskerar att krocka.

En annan aspekt att ta hänsyn till är komforten på resan. Autonoma fordon ger möjlighet till tidsbesparing eftersom passageraren kan ägna sig åt annat än att köra bilen under färden [23]. För att kunna slappna av under resan behöver det kännas tryggt och säkert. För ovana passagerare kan det kännas osäkert att sitta i ett fordon som accelerera mot en korsning samtidigt som flera andra fordon närmar sig korsningen från andra håll.

7
Slutsats

En MPC-algoritm som kontrollerar autonoma fordon igenom en korsning har konstruerats. Simuleringar visar att MPC-algoritmen är kapabel att säkert styra fordon igenom en korsning utan några kollisioner. Den simuleringsmiljö som har utvecklats tillåter fordon att svänga igenom korsningar. Simuleringar visar att felkalbering av MPC-algoritmen kan resultera i att fordon kör av vägen, varför fortsatta undersökningar av hur MPC-algoritmen kalibreras behövs.

Litteraturförteckning

[1] Dr. Eun-Ha Choi and U.S. Department of Transportation National Highway Traffic Safety Administration. *Crash Factors in Intersection-Related Crashes: An On-Scene Perspective (NHTSA Technical Report DOT HS 811 366)*. CreateSpace Independent Publishing Platform, 2010. ISBN 1493507133. URL `https://www.amazon.com/Crash-Factors-Intersection-Related-Crashes-Scene/dp/1493507133?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1493507133`.

[2] A Molinero Martinez, E Carter, Claire L. Naing, M.C. Simon, and T Hermitte. Accident causation and pre-accidental driving situations: Part 1. overview and general statistics. 01 2007.

[3] Hiroki Yamamoto, Daichi Yanagisawa, and Katsuhiro Nishinari. Velocity control for improving flow through a bottleneck. *Journal of Statistical Mechanics: Theory and Experiment*, 2017(4):043204, 2017.

[4] Frank P McKenna. The human factor in driving accidents an overview of approaches and problems. *Ergonomics*, 25(10):867–877, 1982.

[5] J. Lee and B. Park. Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):81–90, March 2012. ISSN 1524-9050. doi: 10.1109/TITS.2011.2178836.

[6] N. Murgovski, G. R. de Campos, and J. Sjöberg. Convex modeling of conflict resolution at traffic intersections. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 4708–4713, Dec 2015. doi: 10.1109/CDC.2015.7402953.

[7] L. Riegger, M. Carlander, N. Lidander, N. Murgovski, and J. Sjöberg. Centralized mpc for autonomous intersection crossing. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1372–1377, Nov 2016. doi: 10.1109/ITSC.2016.7795736.

[8] Wilson Edward L. The static condensation algorithm. *International Journal for Numerical Methods in Engineering*, 8(1):198–203. doi: 10.1002/nme.1620080115. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620080115.

[9] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[10] Ton JJ van den Boom and TCPM Backx. Model predictive control. *DISC Course, Lecture Notes*, 16, 2010.

[11] Abhijit S. Badwe, Ravindra D. Gudi, Rohit S. Patwardhan, Sirish L. Shah, and Sachin C. Patwardhan. Detection of model-plant mismatch in mpc applications. *Journal of Process Control*, 19(8):1305 – 1313, 2009. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2009.04.007. URL http://www.sciencedirect.com/science/article/pii/S0959152409000572. Special Section on Hybrid Systems: Modeling, Simulation and Optimization.

[12] Philip Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, 1959. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1909468.

[13] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[14] IPG_CarMaker. Carmaker. https://ipg-automotive.com/products-services/simulation-software/carmaker/, 2018. [Online; accessed 14 maj 2018].

[15] IPG_CarMaker. Quick Start Guide Version 5.1.4, 2017.

[16] Randolph Bank and Craig C. Douglas. Sparse matrix multiplication package (smmp). 1, 05 2001.

[17] IPG_CarMaker. User's guide version 5.1.4, 2017.

[18] IPG_CarMaker. Programmer's guide version 4.5. 2. https://www.scribd.com/document/17842366/IPG-CarMaker-Programmers-Guide#, 2014. [Online; accessed 14 maj 2018].

[19] Fariba Fahroo and Michael A Demetriou. Optimal actuator/sensor location for active noise regulator and tracking control problems. *Journal of Computational and Applied Mathematics*, 114(1):137–158, 2000.

[20] Sonja Forward. *Driving violations: investigating forms of irrational rationality*. PhD thesis, Universitetsbiblioteket, 2008.

[21] Janet Fleetwood. Public health, ethics, and autonomous vehicles. *American journal of public health*, 107(4):532–537, 2017.

[22] Jean-François Bonnefon, Azim Shariff, and Iyad Rahwan. The social dilemma of autonomous vehicles. *Science*, 352(6293):1573–1576, 2016.

[23] Faheem Ahmed Malik. Autonomous vehicles: Safety, sustainability, and fuel efficiency. *Journal on Future Engineering & Technology*, 12(4), 2017.

[24] Erick Guerra. Planning for cars that drive themselves: Metropolitan planning organizations, regional transportation plans, and autonomous vehicles. *Journal of Planning Education and Research*, 36(2):210–224, 2016.

A

Kod för att generera en korsning i CarMaker

```
#INFOFILE1.1 - Do not remove this line!
FileIdent = IPGRoad 5.0
FileCreator = Linnea 2018-03-21 13:40

nLinks = 4
nJunctions = 1
nRoutes = 12
RoadNetworkLength =
BBox = -71.118 430.985 -120.070 221.310 -0.060 25.913

RST = 13.8889 27.7778 -1 8.33333 19.4444 2.777 -1 -1

Junction.0.Knot = 150.0000 150.0000 0
Junction.0.ArmAlpha = 0.0000 90.0000 180.0000 270.0000
Junction.0.ArmRadius = 10 10 10 10
Junction.0.ArmLength = 5.0000 5.0000 5.0000 5.0000
Junction.0.MainArms = 0 2

Link.0.Junctions = 0 0 -1 -1
Link.0.Node0 = 0.0000 0.0000 0.0000 0.0000
Link.0.Seg.0.Type = Straight
Link.0.Seg.0.Param = 150.000 0.0000 0.0000 0.0000 0.0000 0.0000
    0.0000 0.0000
Link.0.RoadMarking.0 = 150 3 0 3 0 0 0.12 0 2 0 0 2 2 1 1 2 ""
Link.0.RoadMarking.1 = 0 2 0 1 0 2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.0.RoadMarking.2 = 0 2 0 1 0 -2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.0.RoadMarking.4 = 13.34 0 13.34 0 0 2 0.6 0 1 1 0 0.6 0.6
   1 1 2 ""
Link.0.RoadMarking.4.PointList:
        0 1.56
        0 -1.5

Link.0.LaneSection.0.Start = 0
Link.0.LaneSection.0.LaneL.0 =  0 3 4 4 0 0 0
Link.0.LaneSection.0.LaneR.0 =  0 3 4 4 0 0 0

Link.1.Junctions = 0 1 -1 -1
```

```
Link.1.Node0 = 150.0000 -150.0000 0.0000 0.0000
Link.1.RST = contryroad
Link.1.Seg.0.Type = Straight
Link.1.Seg.0.Param = 150.0000 0.0000 0.0000 0.0000 0.0000
   0.0000 0.0000 0.0000
Link.1.RoadMarking.0 = 150 3 0 3 0 0 0.12 0 2 0 0 2 2 1 1 2 ""
Link.1.RoadMarking.1 = 0 2 0 1 0 2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.1.RoadMarking.2 = 0 2 0 1 0 -2 0.12 0 1 0 0 5 2 1 1 2 ""

Link.2.Junctions = 0 2 -1 -1
Link.2.Node0 = 0.0000 300.0000 0.0000 0.0000
Link.2.RST = contryroad
Link.2.Seg.0.Type = Straight
Link.2.Seg.0.Param = 150.0000 0.0000 0.0000 0.0000 0.0000
   0.0000 0.0000 0.0000
Link.2.RoadMarking.0 = 150 3 0 3 0 0 0.12 0 2 0 0 2 2 1 1 2 ""
Link.2.RoadMarking.1 = 0 2 0 1 0 2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.2.RoadMarking.2 = 0 2 0 1 0 -2 0.12 0 1 0 0 5 2 1 1 2 ""

Link.3.Junctions = 0 3 -1 -1
Link.3.Node0 = 150.0000 150.0000 0.0000 0.0000
Link.3.RST = contryroad
Link.3.Seg.0.Type = Straight
Link.3.Seg.0.Param =150.0000 0.0000 0.0000 0.0000 0.0000 0.0000
    0.0000 0.0000
Link.3.RoadMarking.0 = 150 3 0 3 0 0 0.12 0 2 0 0 2 2 1 1 2 ""
Link.3.RoadMarking.1 = 0 2 0 1 0 2 0.12 0 1 0 0 5 2 1 1 2 ""
Link.3.RoadMarking.2 = 0 2 0 1 0 -2 0.12 0 1 0 0 5 2 1 1 2 ""

Route.0:
        0
        1

Route.0.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.0.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.1:
        0
        2

Route.1.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.1.Path.1:
        0.0000 -2 0
```

```
        145.0000 -2 0

Route.2:
        0
        3

Route.2.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.2.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.3:
        1
        0

Route.3.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.3.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.4:
        1
        2

Route.4.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.4.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.5:
        1
        3

Route.5.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.5.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.6:
```

```
        2
        0

Route.6.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.6.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.7:
        2
        1

Route.7.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.7.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.8:
        2
        3

Route.8.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.8.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.9:
        3
        0

Route.9.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.9.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.10:
        3
        1
```

```
Route.10.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.10.Path.1:
        0.0000 -2 0
        145.0000 -2 0

Route.11:
        3
        2

Route.11.Path.0:
        0.0000 -2 0
        145.0000 -2 0

Route.11.Path.1:
        0.0000 -2 0
        145.0000 -2 0
```

B

Kod för att köra en simulering med fyra fordon i CarMaker

```
#INFOFILE1.1 - Do not remove this line!
FileIdent = CarMaker-TestRun 5.0
FileCreator = CarMaker 5.1.4 2017-1-20
Description:
Vehicle = Examples/Demo_Audi_R8
Trailer =
Tire.0 =
Tire.1 =
Tire.2 =
Tire.3 =
Snapshot.TimeLimit =
Snapshot.DistLimit =
VehicleLoad.0.mass = 0
VehicleLoad.0.pos = 0 0 0
VehicleLoad.1.mass = 0
VehicleLoad.1.pos = 0 0 0
VehicleLoad.2.mass = 0
VehicleLoad.2.pos = 0 0 0
VehicleLoad.3.mass = 0
VehicleLoad.3.pos = 0 0 0
TrailerLoad.0.mass = 0
TrailerLoad.0.pos = 0 0 0
TrailerLoad.1.mass = 0
TrailerLoad.1.pos = 0 0 0
TrailerLoad.2.mass = 0
TrailerLoad.2.pos = 0 0 0
DrivMan.Init.Velocity = 43.2
DrivMan.Init.GearNo = 3
DrivMan.Init.SteerAng = 0
DrivMan.Init.LaneOffset = 0
DrivMan.Init.OperatorActive = 1
DrivMan.Init.OperatorState = drive
DrivMan.VhclOperator.Kind = IPGOperator 1
DrivMan.nDMan = 1
DrivMan.0.TimeLimit = 50
DrivMan.0.LongDyn = Driver 1 0 43.2
DrivMan.0.LatDyn = Driver 0
```

```
Traffic.N = 4
Traffic.SpeedUnit = ms
Traffic.0.ObjectKind = Movable
Traffic.0.Name = T00
Traffic.0.Info = UNNAMED Object 0
Traffic.0.Movie.Geometry = VW_Beetle_2012_Blue.mobj
Traffic.0.Color = 1.0 1.0 1.0
Traffic.0.Basics.Dimension = 4.1 1.7 1.2
Traffic.0.Basics.Offset = 0.2 0.0
Traffic.0.Init.Orientation = 0.0 0.0 0.0
Traffic.0.FreeMotion = 0
Traffic.0.DetectMask = 0 0
Traffic.0.Route = 7 0
Traffic.0.Init.v = 12.0
Traffic.0.Init.Road = 0 -2
Traffic.0.Man.TreatAtEnd = FreezePos
Traffic.0.Man.N = 0
Traffic.1.ObjectKind = Movable
Traffic.1.Name = T01
Traffic.1.Info = UNNAMED Object 1
Traffic.1.Movie.Geometry = VW_Beetle_2012_Blue.mobj
Traffic.1.Color = 1.0 1.0 1.0
Traffic.1.Basics.Dimension = 4.1 1.7 1.2
Traffic.1.Basics.Offset = 0.2 0.0
Traffic.1.Init.Orientation = 0.0 0.0 0.0
Traffic.1.FreeMotion = 0
Traffic.1.DetectMask = 0 0
Traffic.1.Route = 4 0
Traffic.1.Init.v = 12.0
Traffic.1.Init.Road = 0 -2
Traffic.1.Man.TreatAtEnd = FreezePos
Traffic.1.Man.N = 0
Traffic.2.ObjectKind = Movable
Traffic.2.Name = T02
Traffic.2.Info = UNNAMED Object 2
Traffic.2.Movie.Geometry = VW_Beetle_2012_Blue.mobj
Traffic.2.Color = 1.0 1.0 1.0
Traffic.2.Basics.Dimension = 4.1 1.7 1.2
Traffic.2.Basics.Offset = 0.2 0.0
Traffic.2.Init.Orientation = 0.0 0.0 0.0
Traffic.2.FreeMotion = 0
Traffic.2.DetectMask = 0 0
Traffic.2.Route = 9 0
Traffic.2.Init.v = 12.0
Traffic.2.Init.Road = 0 -2
Traffic.2.Man.TreatAtEnd = FreezePos
Traffic.2.Man.N = 0
Traffic.3.ObjectKind = StatWithName
Traffic.3.Name = T19
Traffic.3.Info = UNNAMED Object 19
Traffic.3.Movie.Geometry = 3DObjects/Vegetation/Bush01.mobj
```

```
Traffic.3.Color = 1.0 1.0 1.0
Traffic.3.Basics.Dimension = 0.001 0.001 0.001
Traffic.3.Basics.Offset = 0.2 0.0
Traffic.3.Init.Orientation = 0.0 0.0 0.0
Traffic.3.FreeMotion = 0
Traffic.3.DetectMask = 0 0
Traffic.3.Route = 0 0
Traffic.3.Init.v = 0.0
Traffic.3.Init.Road = 145.0 -3
DrivMan.OW.Active = 0
DrivMan.OW.Quantities = Time
DrivMan.OW.StartGearNo = 1
DrivMan.OW.StartVelocity =
DrivMan.OW.GasMax = 0.5
DrivMan.OW.Time.Name =
DrivMan.OW.Time.Factor = 1.0
DrivMan.OW.Time.Offset = 0.0
ErrorClass.0.Action = abort
ErrorClass.0.Save = 0
ErrorClass.0.WarningLimit = 3 5
ErrorClass.1.Action = abort
ErrorClass.1.Save = 0
ErrorClass.1.WarningLimit = 3 5
ErrorClass.2.Action = abort
ErrorClass.2.Save = 0
ErrorClass.2.WarningLimit = 3 5
ErrorClass.3.Action = abort
ErrorClass.3.Save = 0
ErrorClass.3.WarningLimit = 3 5
ErrorClass.4.Action = abort
ErrorClass.4.Save = 0
ErrorClass.4.WarningLimit = 3 5
ErrorClass.5.Action = abort
ErrorClass.5.Save = 0
ErrorClass.5.WarningLimit = 3 5
ErrorClass.10.Action = abort
ErrorClass.10.Save = 0
ErrorClass.10.WarningLimit = 3 5
ErrorClass.11.Action = abort
ErrorClass.11.Save = 0
ErrorClass.11.WarningLimit = 3 5
Road.VhclStartPos = 0
Road.Lane = center
Road.DigSmooth = 0.2 0 0
Road.DigOptions =
Road.Country = DEU
Road.RouteId = 1
Road.GCS.RefPos_0 = 0 0 0
Road.GCS.Projection =
Road.2Movie.MStrip = 0.12 3 6
Road.2Movie.SStrip = 0.12 0 0 0.1
```

```
Road.2Movie.GStrip = 1.50 100
Road.2Movie.MaxErrXY = 0.02
Road.2Movie.GeoStepSize = 0.20
Road.2Movie.GenBridges = 0
Road.2Movie.BgGeoFName =
Road.2Movie.BgGeoOptions =
Road.2Movie.TerrainFName =
Road.Definition:
        FileIdent IPGRoad 4.5
        DigRoad   myroad.road
        Origin    0 0 0 0
        Default 3.00 3.00 0.50 0.50 1.0 1.0 - 0 0 - 0 0
Env.StartTime.Year = 2014
Env.StartTime.Month = 1
Env.StartTime.Day = 1
Env.StartTime.Hour = 12
Env.StartTime.Min = 0
Env.StartTime.Sec = 0
Env.StartTime.DeltaUTC = 0.0
Env.GNSS.Active = 0
Env.Temperature = 20.0
Env.AirDensity = 1.205
Env.AirPressure = 1.013
Env.AirHumidity = 60
Env.SolarRadiation = 400.0
Env.Wind.Kind = none
Env.Wind.Velocity = 0.0
Env.Wind.Angle = 0.0
Env.Kind = Generic
Env.Temp.Offset_Elev = -0.0065
Env.Temp.Offset_sRoad.Amplify = 1.0
Env.Temp.Offset_sRoad.On = 0
Env.Temp.Offset_Time.Amplify = 1.0
Env.Temp.Offset_Time.On = 1
Env.Temp.Offset_Time:
        0.0 -2.0
        3.0 -2.5
        6.0 -2.7
        7.5 -2.7
        9.0 -2.5
        10.0 -2.3
        11.0 -1.6
        12.0 0.0
        13.0 1.4
        14.0 2.1
        15.5 2.5
        17.0 2.2
        18.0 1.7
        19.0 1.1
        20.0 0.2
        21.0 -0.6
```

```
        22.0 -1.1
        23.0 -1.6
        24.0 -2.0
Driver.ParamIdent = IPGDriver 5
Driver.Mode = std
Driver.Long.DrivMaxSpeed = 0
Driver.Long.CruisingSpeed = 150
Driver.CornerCutCoef = 0.5
Driver.ConsiderTraffic = 1
Driver.Traffic.TimeGapMin = 1.8
Driver.Traffic.TimeGapMax = 5.0
Driver.Traffic.DistMin = 6
Driver.Traffic.DistMax = 250
Driver.Traffic.EcoCoef = 0.75
Driver.Traffic.Overtake = 0
Driver.Traffic.Overtake_Rate = 1
Driver.Traffic.Overtake_dSpeedMin = 10
Driver.Long.dtAccBrake = 0.5
Driver.Long.axMax = 3.0
Driver.Long.axMin = -4.0
Driver.Long.ayMax = 4.0
Driver.Long.GGExp:
        50 1.0 1.0
Driver.Long.DevMax = 0.0
Driver.Long.tReact = 0.0
Driver.Long.TractionControl = 1
Driver.DecShift.UseBrakePark = 0
Driver.DecShift.tSwitchGear = 1.0
Driver.DecShift.nEngine.Limits:
        1500 4000
Driver.DecShift.nEngine.Shift:
        2000 3000
Driver.Lat.DevMax = 0.0
Driver.Lat.tReact = 0.0
Driver.Knowl.Long.tActionMin = 4
Driver.Knowl.Lat.StWhlAngleMax = 630
Driver.Knowl.Lat.StWhlAngleVelMax = 500
Driver.Knowl.Lat.StWhlAngleAccMax = 3000
Driver.Learn.VehicleLimits.TestRun =
Driver.Learn.VehicleLimits.Date = 0
Driver.Learn.ControllerDyn.TestRun =
Driver.Learn.ControllerDyn.Date = 0
Driver.Learn.MaxSpeed.TestRun =
Driver.Learn.MaxSpeed.Date = 0
Driver.Learn.Remember = 0
Driver.Knowl.Long.tPreviewBra = 0.6
Driver.Knowl.Long.tPreviewAcc = 1.5
Driver.Knowl.Lat.tPreview = 0.8
Driver.Learn.NEng_S = 1
```

C

Kod för optimerare

```
function res=QPsolve(task,astart,vstart,ss)
% QPsolve  solves the quadratic problem defined by task with
    initial values
%astart, vstart and ss
%   res = QPsolve(task,astart,vstart,ss) returns the solution
    object res
%   given a task, and the initial accelerations and velocities
    astart, vstart and the startingposition ss for the
%   vehicles. This is done by computing computing the matrices
    given as
%   inputs to Quadprog corresponding to the task and its
    initial values.

% initialize values for number of vehicles, sample length,
    number of
% vehicles etc. These are precomputed before the generation of
    matrices
% start
V=task.V;
Ns=task.Ns;
Nv=task.Nv;
ds=task.ds;
co=task.crossorderperm; % precomputed crossing order of the
    task

% scaling factors, used to speed up quadprog computation
St=task.St; Sz=task.Sz; Sdz=task.Sdz; Sddz=task.Sddz; Scost=
    task.Scost;

%penalties used in cost function
Wv=task.Wv; Wdv=task.Wdv; Wddv=task.Wddv;
%%
vref=12; % reference speed [m/s]

% anonymous functions for indexing of state vector sample i,
    vehicle j
tind = (i,j) 4*i-3 + 4*Ns*(j-1);zind = (i,j) 4*i-2 + 4*Ns*(j-1);
dzind = (i,j) 4*i-1 + 4*Ns*(j-1);uind = (i,j) 4*i + 4*Ns*(j-1);
```

```
% equality box constraints
Aeq = [];
Aeq = sparse(Aeq);
beq = [];
% inequality box constraints
Aineq = [];
Aineq = sparse(Aineq);
bineq = [];
% upper and lower bounds
ub = zeros(4*Ns*Nv,1);
lb = zeros(4*Ns*Nv,1);


for i = 1:Ns-1
    for j = 1:Nv
        % generate rows of Aeq corresponding to equality box
            constraints
        cond = zeros(4,4*Ns*Nv);
        cond(1,tind(i+1,j)) = 1;
        cond(1,tind(i,j)) = -1;
        cond(1,zind(i,j)) = - ds;

        cond(2,zind(i+1,j)) = 1;
        cond(2,zind(i,j)) = -1;
        cond(2,dzind(i,j)) = - ds;

        cond(3,dzind(i+1,j)) = 1;
        cond(3,dzind(i,j)) = -1;
        cond(3,uind(i,j)) = - Sddz/Sdz*ds;

        cond(4,:) = zeros(1,4*Ns*Nv);
        Aeq = [Aeq; cond];
        beq = [beq; [0 0 0 0]'];

    end
end

% initial values
for i = 1:Nv
    cond = zeros(3,4*Ns*Nv);
    cond(1,tind(1,i)) = 1;
    cond(2,zind(1,i)) = 1;
    cond(3,dzind(1,i)) = 1;
    Aeq = [Aeq;cond];
    beq = [beq; [0 1/vstart(i)/Sz 0]'];
end

for i = 1:Nv-1
    % critical zone constraints
    sample1 = V(co(i)).Nze;
    vehicle1 = co(i);
```

```
    xind1 = tind(sample1,vehicle1);

    sample2 = V(co(i+1)).Nzs;
    vehicle2 = co(i+1);
    xind2=tind(sample2,vehicle2);

    cond = zeros(1,4*Nv*Ns);
    cond(xind1) = 1;
    cond(xind2) = -1;
    Aineq = [Aineq;cond];
    bineq = [bineq; 0];

end

vln = 100000; % very small and large numbers, used for "non-
    constraints"
vsn = - vln;

% lower bounds
for i = 1:Ns
    for j = 1:Nv
        lb(tind(i,j)) = vsn;
        lb(zind(i,j)) = 1/V(j).vxmax/Sz;
        lb(dzind(i,j)) = vsn;
        lb(uind(i,j)) = vsn;
    end
end

% upper bounds
for i = 1:Ns
    for j = 1:Nv
        ub(tind(i,j)) = vln;
        ub(zind(i,j)) = 1/V(j).vxmin/Sz;
        ub(dzind(i,j)) = vln;
        ub(uind(i,j)) = vln;
    end
end

% construct cost function
H = zeros(4*Ns*Nv, 1);
f = zeros(4*Ns*Nv,1);
for i= 1:Ns
    for j = 1:Nv
        % first cost function
        H(zind(i,j)) = Wv*vref.^3;
        f(zind(i,j),1) = -2*1/vref;

        % second cost function
        H(dzind(i,j)) = Wdv*vref^5;
        f(zind(i,j),1) = 0;
```

```
        % third cost function
        H(uind(i,j)) = Wddv*vref^7;
        f(uind(i,j)) = 0;
    end
end
H=diag(H);
H=sparse(H); % use sparse matrix algebra to speed up
    computation

[XhatOptimized,fval,exitflag,output] =quadprog(H,f,Aineq,bineq,
   Aeq,beq,lb,ub);

%%
if exitflag == 1
    res.status='Solved';
    res.cost=fval;
    res.v2=[];
    res.t2=[];
    for i=1:Nv
        for j=1:Ns
            res.v2(j,i)=1./XhatOptimized(zind(j,i),1)/Sz;
            res.t2(j,i)=XhatOptimized(tind(j,i),1)/St;
        end
    end
    res.v=res.v2;
    res.t=res.t2;

    for i=1:Ns-1
        for j=1:Nv
            res.a(j,i)=(res.v(i+1,j)-res.v(i,j))/(res.t(i+1,j)-
                res.t(i,j));
        end
    end
    res.a=res.a';
else
    res.status='Unresolved';
    disp('optimization failed');
end

end

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   Created by Fredrik Johnsson and Carl Svensson 2018-05.
```