Modeling and Control of Hybrid Systems

Assignment report - Group A 37

13/06/2022

Aron Bevelander - 4568125
Sachin Umans - 4720377

Introduction

This report considers modeling problems that are tackled with a hybrid system approach. First of all, a simple hybrid system is modeled and described in order to get acquainted with the idea of hybrid modeling. Next, an Adaptive Cruise Control (ACC) problem is considered. The aim is to analyse the problem and in the end find an optimal control input for which the car follows a given speed profile. This is done by transforming the problem into a Mixed Logical Dynamical (MLD) model and apply a Model Predictive Controller (MPC) on it.

1 Hybrid System example

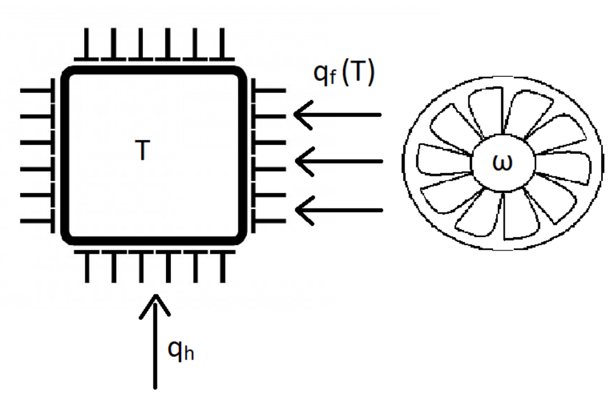


Figure 1: CPU cooler system

First, for a hybrid modeling exercise, a simple CPU cooling system is considered and shown in figure 1. On the left side, the CPU is shown that has a perfectly measurable temperature and a constant heat energy input as an effect of electrical power dissipation. The fan is operational in three modes, e.g. three fan speeds, that induce different cooling rates. The parameters are denoted in table 1. These help understand the system dynamics as given in equations 1 and 2, which describe the CPU temperature evolution and the depending CPU mode respectively. Note that the first equation includes a minus sign in front of the fan contribution, as the heat added is negative.

Parameter	interpretation
T	CPU Temperature
q_h	heat CPU input
$q_f(T)$	heat input due to fan convection
ω	fan speed

Table 1: System Parameters

$$\dot{T} = q_h - q_f \tag{1}$$

$$q_f(T) = \begin{cases} q_0 & \text{if}\, T < T_0 \\ q_{c1} & \text{if}\, T_0 \leq T < T_1 \\ q_{c2} & \text{if}\, T_1 \leq T \end{cases} \tag{2}$$

Next, a hybrid automaton of the system can be constructed. It is given by the 8-tuple defined in equation (3) and elaborated in equation (4). The physical representation is shown in figure 2.

$$H = \{Q, X, f, Init, Inv, E, G, R\} \tag{3}$$

$$
\begin{aligned}
Q &= \{S_1, S_2, S_3\} \\
X &= \mathbb{R} \\
f &: Q \times X \to X \\
&: \{(S_1, T) \to q_h,\ (S_2, T) \to q_h - q_{c1},\ (S_3, T) \to q_h - q_{c2}\} \\
Init &= Q \times X \\
Inv &= \{(S_1, T < T_0),\ (S_2, T \in [T_0, T_1]),\ (S_3, T > T_1)\} \\
E &= \{(S_1, S2),\ (S_2, S1),\ (S_2, S3),\ (S_3, S_2)\} \\
G &= \{((S_1, S2), T > T_0), ((S_2, S1), T \leq T_0), ((S_2, S3), T > T_1), ((S_3, S_2), T \leq T_1)\} \\
R &: E \to P(X \times X) \\
&: E \to (T^- = T^+)
\end{aligned}
\tag{4}
$$

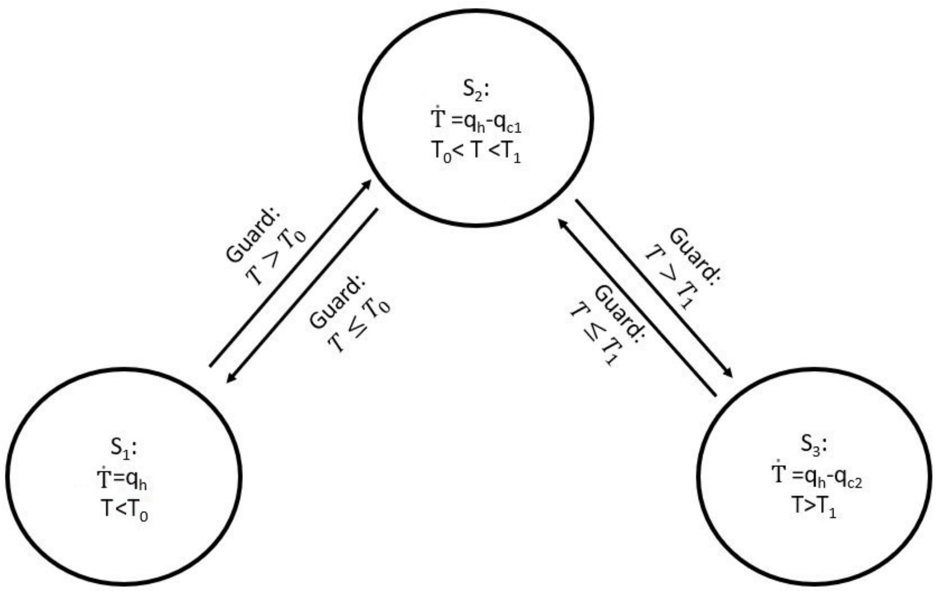


Figure 2: Hybrid Automaton

2 Adaptive cruise control

2.1 Control magnitudes and extreme velocities

Next, the ACC car is considered. First, using equation 5, a number of quantities are demanded.

$$\frac{\mathsf{d}}{\mathsf{d}t}\begin{bmatrix} x(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \frac{1}{m}\left(\frac{bu(t)}{1+\gamma g(t)} - cv^2(t)\right) \end{bmatrix} \tag{5}$$

The maximum velocity is obtained when the maximum drive force equals the air friction force. So that

$$\frac{1}{m}\left(\frac{bu_{max}}{1+\gamma g_{max}} - cv_{max}^2(t)\right) = 0. \tag{6}$$

Which gives

$$v_{max} = \left(\frac{bu_{max}}{c(1+\gamma g_{max})}\right)^{\frac{1}{2}}. \tag{7}$$

If equation 5 is considered again, it can be seen that it consists of a positive and negative part. The maximum acceleration should occur when the positive part is at its largest and the negative part is at its smallest. This implies: $u = u_{max}$ and $v = 0$ in this case and yields:

$$a_{max} = \frac{1}{m}\frac{bu_{max}}{1+\gamma g(t)} \tag{8}$$

The same reasoning can be used for finding the maximal deaccelaration. Now the first term has to be minimal and the second term has to be maximal. This will happen at the maximal velocity when suddenly the maximal brake input is applied. This implies:

$$a_{min} = \frac{1}{m}\left(\frac{bu_{min}}{1+\gamma g(t)} - cv_{max}^2(t)\right) \tag{9}$$

The found values are denoted in table 2. Besides, figure 3 describes a simulation run of the car with all the prescribed dynamics. It can be seen that indeed the arguments with respect to the extremes hold.

Table 2: Model quantities

Entity	Value
Maximum speed	57.72ms^{-1}
Maximum acceleration	3.22ms^{-2}
Maximum deaccelaration	3.33ms^{-2}

2.2 Friction approximation

Now the task is to construct a PWA that approximates the quadratic friction curve using two regions. This can be done by minimizing the integral

$$\int_0^{v_{max}} (V(v) - P(v))^2 \mathsf{d}v \tag{10}$$

The two line segments are given as follows:

- The first one goes through the origin and through point (α, β)
- The second one goes through point (α, β) and through point (v_{max}, cv_{max}^2)

Thus

$$P(v) = P_1(v) + P_2(v) \tag{11a}$$

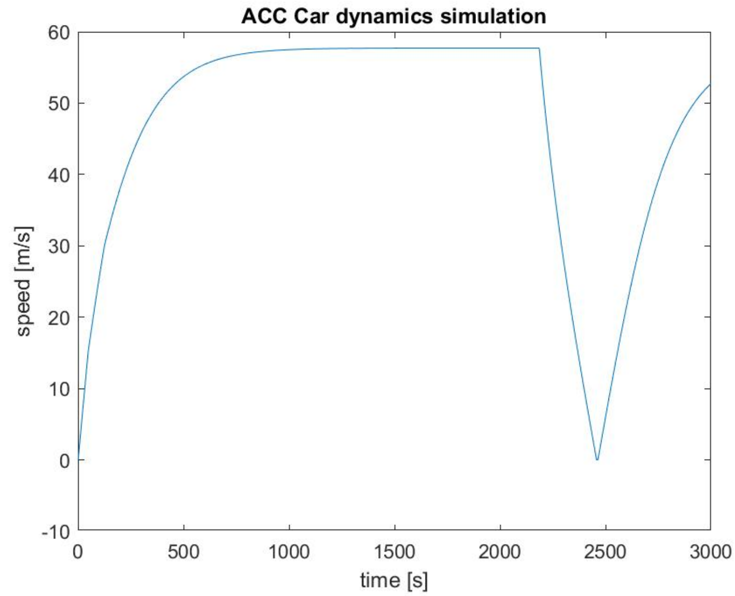


Figure 3: Simulation of the ACC Car

$$P_1(v) \begin{cases} \frac{\beta}{\alpha} v & v < \alpha \\ 0 & v \geq \alpha \end{cases} \tag{11b}$$

$$P_2(v) \begin{cases} 0 & v < \alpha \\ \frac{cv_{max}^2 - \beta}{v_{max} - \alpha} v + \beta - \frac{cv_{max}^2 - \beta}{v_{max} - \alpha} \alpha & v \geq \alpha \end{cases} \tag{11c}$$

The Equation 10 can then be divided into two sections as

$$A = \int_0^{\alpha} (V(v) - P_1(v))^2 \mathrm{d}v + \int_{\alpha}^{v_{max}} (V(v) - P_2(v))^2 \mathrm{d}v. \tag{12}$$

Using the "Symbolic Math Toolbox" provided in `MatLab`, a function $A(\alpha, \beta)$ is determined. To find the extremes of A the system of equations

$$\begin{cases} \frac{\mathrm{d}A}{\mathrm{d}\alpha} = 0 \\ \frac{\mathrm{d}A}{\mathrm{d}\beta} = 0 \end{cases} \tag{13}$$

is solved with `MatLab`'s `solve` function. This yields three pairs of (α, β), of which only one is valid. That is, $\alpha \in [0, v_{max}]$ and $\beta \in [0, cv_{max}^2]$. When evaluating the second derivative of A with respect to both α and β, both turn out to be greater than 0. Thus, this extreme is a minimum. The values are

$$\begin{cases} \alpha \approx 28.8575 \\ \beta \approx 249.8269. \end{cases} \tag{14}$$

2.3 Approximation compared to reality

Right now, there is looked upon the differences between the different friction force models. The friction dynamics correspond with those presented in figure 4. The PWA-model friction dynamics are shown by the red line segments, the true friction force by the blue curve. The velocities at which the different model coincide are called $v_{cross,1}$ and $v_{cross,2}$, which are about $21.5\frac{m}{s}$ and $36.5\frac{m}{s}$ respectively. This shows that the PWA-modelled friction force is in reality a bit lower for all velocities $v < v_{cross,1}$ and $v > v_{cross,2}$ and higher for all velocities $v_{cross,1} < v < v_{cross,2}$.

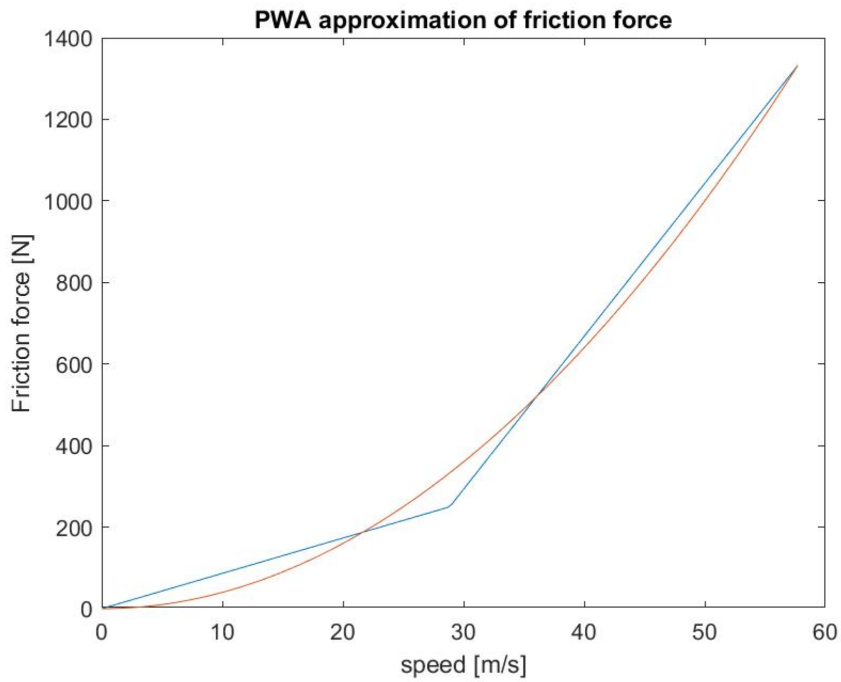


Figure 4: Different friction models

This means that the model will build up an error that correspond with time integral of the difference in the 2 curves. Namely, in the time the velocity is lower than $v_{cross,1}$ and larger than $v_{cross,2}$, the PWA model will introduce an acceleration larger than in reality, in the time the velocity is between $v_{cross,1}$ and $v_{cross,2}$, the PWA model will introduce an acceleration smaller than reality. If a sinusoidal throttle input is applied, as shown in figure 5a, it can be seen that the properties of the input make the error cancel out until some extend, as the time spend in both speed regions is more or less equal as can be seen in figure 5b.

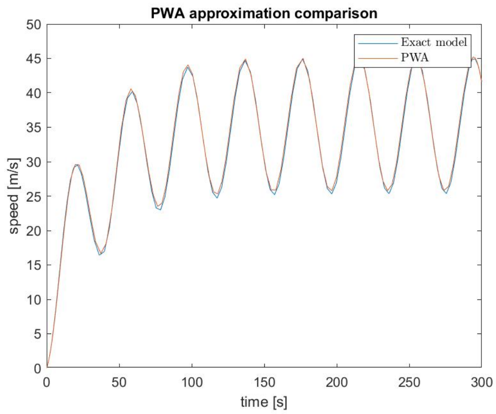
2.4 PWA model

Considering a fully functional gear system again, the piecewise affine (PWA) system model with friction approximation, can be created with the found results. Since there are two regions in the friction approximation, and the gear creates three regions in the driving force, the system will have six regions. These regions are denoted as $\mathbb{X}_i, i \in \{1,2,3,4,5,6\}$ and the corresponding dynamics as f_i. Defining the regions as

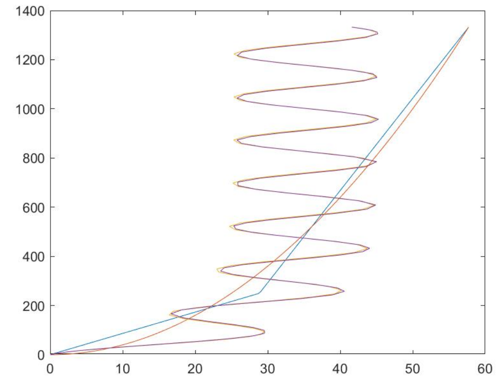
$$\begin{aligned} \mathbb{X}_1 &= \{(x,v) \mid v < v_{12}, v < \alpha\} & \mathbb{X}_4 &= \{(x,v) \mid v < v_{12}, v \geq \alpha\} \\ \mathbb{X}_2 &= \{(x,v) \mid v \geq v_{12}, v < v_{23}, v < \alpha\} & \mathbb{X}_5 &= \{(x,v) \mid v \geq v_{12}, v < v_{23}, v \geq \alpha\} \\ \mathbb{X}_3 &= \{(x,v) \mid v \geq v_{23}, v < \alpha\} & \mathbb{X}_6 &= \{(x,v) \mid v \geq v_{23}, v \geq \alpha\} \end{aligned} \tag{15}$$

creates a division of the whole of $\mathbb{R}^2$. However, since α is found to be between v_{12} and v_{23}, $\mathbb{X}_3$ and $\mathbb{X}_4$ are unreachable, so no dynamics will be defined for these regions. Lastly, three driving force constants are defined as

$$F_{\text{drive},i} = \frac{b}{1+\gamma i} \qquad , i \in \{1,2,3\}. \tag{16}$$



(a) Model input and exact input



(b) Input magnitude analysis

Figure 5: Throttle input properties

Then the PWA model with state $x = (x, v)$ is

$$\frac{\mathsf{d}}{\mathsf{d}t}x = \begin{cases} \begin{bmatrix} x_2 \\ \frac{1}{m}\left(F_{\mathsf{drive},1}u - P_1(x_2)\right) \end{bmatrix} & , x \in \mathbb{X}_1 \\ \begin{bmatrix} x_2 \\ \frac{1}{m}\left(F_{\mathsf{drive},2}u - P_1(x_2)\right) \end{bmatrix} & , x \in \mathbb{X}_2 \\ \begin{bmatrix} x_2 \\ \frac{1}{m}\left(F_{\mathsf{drive},2}u - P_2(x_2)\right) \end{bmatrix} & , x \in \mathbb{X}_5 \\ \begin{bmatrix} x_2 \\ \frac{1}{m}\left(F_{\mathsf{drive},3}u - P_2(x_2)\right) \end{bmatrix} & , x \in \mathbb{X}_6 \end{cases} \tag{17}$$

2.5 Discretization

In this step, the PWA model is discretized using forward euler discretization. The discrete system dynamics then become

$$x(k+1) = x(k) + \dot{x}\Delta t. \tag{18}$$

The results can be shown in the appendix at code section 2.5.

2.6 MLD model

From here on only the second state of the system is considered. The goal of this section is to cast the dynamics into the Mixed Logical Dynamical (MLD) system framework. First, suited expressions for the drag force and driving force will be determined, after which they can be expressed in the MLD format. For the MLD format it is important to remember the ranges in which our state and input live, namely $x \in [0, v_{max}]$ and $u \in [u_{min}, u_{max}]$.

The drag force is expressed as

$$P(x) = \begin{cases} kx & x < \alpha \\ qx + r & x \geq \alpha \end{cases} \tag{19}$$

in the PWA model, where

$$
\begin{aligned}
k &= \frac{\beta}{\alpha} \\
q &= \frac{cv_{max}^2 - \beta}{v_{max} - \alpha} \\
r &= \beta - \frac{cv_{max}^2 - \beta}{v_{max} - \alpha}\alpha.
\end{aligned}
\tag{20}
$$

We introduce a boolean variable δ_P such that $[x \geq \alpha] \iff [-x + \alpha \leq 0] \iff [\delta_P = 1]$. This leads to the constraints for δ_P to be

$$
\begin{aligned}
-x + \alpha &\leq \alpha(1 - \delta_P) \\
-x + \alpha &\geq \epsilon + (-v_{max} + \alpha - \epsilon)\delta_P,
\end{aligned}
\tag{21}
$$

where ϵ is machine precision. The PWA function for the drag force can then be expressed as

$$
\begin{aligned}
P(x) = kx + \delta_P(-kx + qx + r) &= (q - k)\delta_P x + kx + \delta_P r \\
&= (q - k)z_P + kx + \delta_P r,
\end{aligned}
\tag{22}
$$

where $z_P = \delta_P x$, or equivalently

$$
\begin{aligned}
z_P &\leq v_{max}\delta_P \\
z_P &\geq 0 \\
z_P &\leq x \\
z_P &\geq x - v_{max}(1 - \delta_P).
\end{aligned}
\tag{23}
$$

The driving force in the PWA model is expressed as

$$
F_{\text{drive},i}u = \frac{b}{1 + \gamma i}u \qquad , i \in \{1, 2, 3\}.
\tag{24}
$$

To rewrite this expression, two more boolean variables are introduced. The first is δ_2 such that $[x \geq v_{12}] \iff [-x + v_{12} \leq 0] \iff [\delta_2 = 1]$. The second is δ_3 such that $[x \geq v_{23}] \iff [-x + v_{23} \leq 0] \iff [\delta_3 = 1]$. Thus the associated constraints for these variables are

$$
\begin{aligned}
-x + v_{12} &\leq v_{12}(1 - \delta_2) \\
-x + v_{12} &\geq \epsilon + (-v_{max} + v_{12} - \epsilon)\delta_2,
\end{aligned}
\tag{25}
$$

and

$$
\begin{aligned}
-x + v_{23} &\leq v_{23}(1 - \delta_3) \\
-x + v_{23} &\geq \epsilon + (-v_{max} + v_{23} - \epsilon)\delta_3.
\end{aligned}
\tag{26}
$$

It then becomes possible to rewrite the driving force into

$$
F_{\text{drive}}u = F_{\text{drive},1}u + \delta_2 u(F_{\text{drive},2} - F_{\text{drive},1}) + \delta_3 u(F_{\text{drive},3} - F_{\text{drive},2}).
\tag{27}
$$

Again two auxiliary variables, z_2 and z_3, are introduced as $z_2 = \delta_2 u$ and $z_3 = \delta_3 u$ under the constraints

$$
\begin{aligned}
z_2 &\leq u_{max}\delta_2 \\
z_2 &\geq u_{max}\delta_2 \\
z_2 &\leq u - u_{min}(1 - \delta_2) \\
z_2 &\geq u - u_{max}(1 - \delta_2)
\end{aligned}
\tag{28}
$$

and

$$
\begin{aligned}
z_3 &\leq u_{max}\delta_3 \\
z_3 &\geq u_{max}\delta_3 \\
z_3 &\leq u - u_{min}(1 - \delta_3) \\
z_3 &\geq u - u_{max}(1 - \delta_3).
\end{aligned}
\tag{29}
$$

Then

$$F_{\mathsf{drive}}u = F_{\mathsf{drive},1}u + z_2(F_{\mathsf{drive},2} - F_{\mathsf{drive},1}) + z_3(F_{\mathsf{drive},3} - F_{\mathsf{drive},2}). \tag{30}$$

All of the equations are now ready to be written in the MLD format as

$$\begin{aligned} x(k+1) &= Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k) \\ E_1x(k) &+ E_2u(k) + E_3\delta(k) + E_4z(k) \le g_5. \end{aligned} \tag{31}$$

Starting with the dynamics, substituting the new results into Equation 18 yields

$$\begin{aligned} x(k+1) &= x(k) + \dot{x}\Delta t = x(k) + \frac{\Delta t}{m}\left(F_{\mathsf{drive}}u(k) - P(x(k))\right) \\ &= x(k) + \frac{\Delta t}{m}\big(F_{\mathsf{drive},1}u(k) + z_2(k)(F_{\mathsf{drive},2} - F_{\mathsf{drive},1}) + z_3(k)(F_{\mathsf{drive},3} - F_{\mathsf{drive},2}) \\ &\quad -[(q-k)z_P(k) + kx(k) + \delta_P(k)r]\big) \\ &= x(k) + \frac{\Delta t}{m}F_{\mathsf{drive},1}u(k) + \frac{\Delta t}{m}(F_{\mathsf{drive},2} - F_{\mathsf{drive},1})z_2(k) + \frac{\Delta t}{m}(F_{\mathsf{drive},3} - F_{\mathsf{drive},2})z_3(k) \\ &\quad + \frac{\Delta t}{m}[(k-q)z_P(k) - kx(k) - r\delta_P(k)] \\ &= x(k) + \frac{\Delta t}{m}F_{\mathsf{drive},1}u(k) + \frac{\Delta t}{m}(F_{\mathsf{drive},2} - F_{\mathsf{drive},1})z_2(k) + \frac{\Delta t}{m}(F_{\mathsf{drive},3} - F_{\mathsf{drive},2})z_3(k) \\ &\quad + \frac{\Delta t}{m}(k-q)z_P(k) - \frac{\Delta t}{m}kx(k) - \frac{\Delta t}{m}r\delta_P(k) \\ &= \underbrace{(1 - \frac{\Delta t}{m}k)}_{A}x(k) + \underbrace{\frac{\Delta t}{m}F_{\mathsf{drive},1}}_{B_1}u(k) + \underbrace{\begin{pmatrix}0 & 0 & -\frac{\Delta t}{m}r\end{pmatrix}}_{B_2}\begin{bmatrix}\delta_2(k)\\ \delta_3(k)\\ \delta_P(k)\end{bmatrix} \\ &\quad + \underbrace{\frac{\Delta t}{m}\begin{pmatrix}(F_{\mathsf{drive},2} - F_{\mathsf{drive},1}) & (F_{\mathsf{drive},3} - F_{\mathsf{drive},2}) & (k-q)\end{pmatrix}}_{B_3}\begin{bmatrix}z_2(k)\\ z_3(k)\\ z_P(k)\end{bmatrix}. \end{aligned} \tag{32}$$

All of the mentioned constraints can be written as

$$\underbrace{\begin{bmatrix}1\\-1\\0\\0\\-1\\1\\0\\0\\-1\\1\\-1\\1\\0\\0\\0\\0\\-1\\1\\0\\0\\0\\0\end{bmatrix}}_{E_1} x(k) + \underbrace{\begin{bmatrix}0\\0\\1\\-1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\-1\\1\\0\\0\\0\\0\\-1\\1\end{bmatrix}}_{E_2} u(k) + \underbrace{\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&\alpha\\0&0&(-v_{max}+\alpha-\epsilon)\\0&0&-v_{max}\\0&0&0\\0&0&0\\0&0&v_{max}\\v_{12}&0&0\\(-v_{max}+v_{12}-\epsilon)&0&0\\-u_{max}&0&0\\u_{min}&0&0\\-u_{min}&0&0\\u_{max}&0&0\\0&v_{23}&0\\0&(-v_{max}+v_{23}-\epsilon)&0\\0&-u_{max}&0\\0&u_{min}&0\\0&-u_{min}&0\\0&u_{max}&0\end{bmatrix}}_{E_3} \begin{bmatrix}\delta_2(k)\\\delta_3(k)\\\delta_P(k)\end{bmatrix}$$

$$+ \underbrace{\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&1\\0&0&-1\\0&0&1\\0&0&-1\\0&0&0\\0&0&0\\1&0&0\\-1&0&0\\1&0&0\\-1&0&0\\0&0&0\\0&0&0\\0&1&0\\0&-1&0\\0&1&0\\0&-1&0\end{bmatrix}}_{E_4} \begin{bmatrix}z_2(k)\\z_3(k)\\z_P(k)\end{bmatrix} \leq \underbrace{\begin{bmatrix}v_{max}\\0\\u_{max}\\-u_{min}\\0\\\alpha-\epsilon\\0\\0\\0\\v_{max}\\0\\v_{12}-\epsilon\\0\\0\\-u_{min}\\u_{max}\\0\\v_{23}-\epsilon\\0\\0\\-u_{min}\\u_{max}\end{bmatrix}}_{g_5} \tag{33}$$

When the PWA and MLD model are compared for different speeds and inputs, they are largely equal, except for two points. This is demonstrated in Figure 6. At exactly v_{12} and v_{23} the models differ and it is assumed that this occurs because the one model chooses a different gear than the other at these speeds. However, in practise the chances of the state being exactly v_{12} or v_{23} is extremely small, and even then the relative error to the velocities is small.

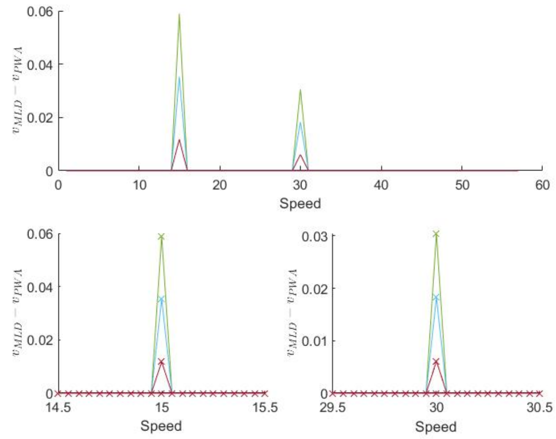


Figure 6: Comparison of a one step state evolution of the PWA and MLD model

2.7 Implicit MPC formulation

This section describes the creation of an MPC controller that optimises speed tracking for the MLD model. The controller considers a prediction horizon N_p and a control horizon $N_c < N_p$ after which the input will remain constant. The state for a time k is defined as

$$x(k) = \begin{bmatrix} v(k) & u(k) & \delta_2(k) & \delta_3(k) & \delta_P(k) & z_2(k) & z_3(k) & z_P(k) \end{bmatrix}^T. \tag{34}$$

The optimisation parameter then becomes a vertical concatenation of $x(k)$ through $x(k+N_p)$.

2.7.1 The cost function

The used optimisation function is the `glpk` function from the "MPT" toolbox that optimises a vector x for cost Cx such that $\Omega x \sim \omega$, where $\sim$ can be one of a number of relations, defined per row of Ω. However in this report only equality and upper bound constraints are considered. The cost function to be optimised is

$$\min_{\mathbf{u}} J(x) = \|\mathbf{v} - \mathbf{v_{ref}}\|_1 + \lambda \|\mathbf{u}\|_1, \tag{35}$$

where $\lambda \in \mathbb{R}^+$. To write this in linear form we introduce two auxiliary variable vectors of length N_p, ρ_x and ρ_u for the state and input norm respectively. An equivalent optimisation problem is then

$$\begin{aligned} \min_{\mathbf{u}} J(x) = & \mathbb{1}^{1\times N_p}\rho_x + \lambda \mathbb{1}^{1\times N_p}\rho_u \\ \text{s.t.} \quad & \mathbf{v} - \mathbf{v_{ref}} \leq \rho_x \\ & \mathbf{v} - \mathbf{v_{ref}} \geq -\rho_x \\ & \mathbf{u} \leq \rho_u \\ & \mathbf{u} \geq -\rho_u. \end{aligned} \tag{36}$$

These constraints are also equivalent to

$$\begin{aligned} & \mathbf{v} - \rho_x \leq \mathbf{v_{ref}} \\ & -\mathbf{v} - \rho_x \leq -\mathbf{v_{ref}} \\ & \mathbf{u} - \rho_u \leq 0 \\ & -\mathbf{u} - \rho_u \leq 0. \end{aligned} \tag{37}$$

The vectors ρ_x and ρ_u are appended to the optimisation parameter, which is now complete. The total optimisation parameter is of size $8N_p + 2N_p = 10N_p$ and is of the form

$$x(k) = \begin{bmatrix} \begin{bmatrix} v(k) & u(k) & \delta_2(k) & \delta_3(k) & \delta_P(k) & z_2(k) & z_3(k) & z_P(k) \end{bmatrix}^T \\ \vdots \\ \begin{bmatrix} v(k+N_p) & u(k+N_p) & \delta_2(k+N_p) & \delta_3(k+N_p) & \delta_P(k+N_p) & z_2(k+N_p) & z_3(k+N_p) & z_P(k+N_p) \end{bmatrix}^T \\ \rho_x \\ \rho_u \end{bmatrix} \tag{38}$$

The cost C then becomes

$$C = \begin{bmatrix} \mathbb{0}^{1\times 8N_p} & \mathbb{1}^{1\times N_p} & \lambda\mathbb{1}^{1\times N_p} \end{bmatrix} \tag{39}$$

2.7.2 Auxiliary optimisation constraints in matrix form

To pass the constraints in Equation 37, they have to be written in matrix form $Mx \leq b_\rho$. To extract $\mathbf{v}$ from x, the matrix X is defined as

$$X = \begin{bmatrix} I_{N_p} \otimes \begin{bmatrix} 1 & \mathbb{0}^{1\times 7} \end{bmatrix} & \mathbb{0}^{N_p\times 2N_p} \end{bmatrix}, \tag{40}$$

where $\otimes$ is the Kronecker product. Likewise to extract $\mathbf{u}$ from x,

$$U = \begin{bmatrix} I_{N_p} \otimes \begin{bmatrix} 0 & 1 & \mathbb{0}^{1\times 6} \end{bmatrix} & \mathbb{0}^{N_p\times 2N_p} \end{bmatrix}. \tag{41}$$

To extract ρ_x and ρ_u, the matrices Γ_x and Γ_u are defined similarly as

$$\Gamma_x = \begin{bmatrix} \mathbb{0}^{N_p\times 8N_p} & I_{N_p} & \mathbb{0}^{N_p\times N_p} \end{bmatrix} \tag{42}$$

and

$$\Gamma_u = \begin{bmatrix} \mathbb{0}^{N_p\times 9N_p} & I_{N_p} \end{bmatrix}. \tag{43}$$

Then the constraints can be written as

$$\underbrace{\begin{bmatrix} X - \Gamma_x \\ -X - \Gamma_x \\ U - \Gamma_u \\ -U - \Gamma_u \end{bmatrix}}_{M} x \leq \underbrace{\begin{bmatrix} \mathbf{v_{ref}} \\ -\mathbf{v_{ref}} \\ \mathbb{0}^{N_p\times 1} \\ \mathbb{0}^{N_p\times 1} \end{bmatrix}}_{b_\rho}. \tag{44}$$

2.7.3 System constraints

The system of course incurs its own constraints as well, these are the dynamics constraint, the logical constraint, and the comfortability constraint. The dynamics constraint is of the form

$$v(k+1) = Hx \iff v(k+1) - Hx = 0, \tag{45}$$

where $H = \begin{bmatrix} A & B_1 & B_2 & B_3 & \mathbb{0}^{1\times 2N_p} \end{bmatrix}$. Let us introduce a notation for off diagonal identity matrices, ${}_nI_m$, such that the nth off diagonal are ones, with towards the upper right being positive, and the matrix is of size $m \times m$. For example

$${}_1I_3 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}. \tag{46}$$

The extraction matrix, Q, of $v(k+1)$ can then concisely be written as

$$Q = {}_1I_{N_p} \otimes \begin{bmatrix} 1 & \mathbb{0}^{1\times 7} \end{bmatrix}. \tag{47}$$

The prediction matrix, T, is expressed as

$$T = I_{N_p} \otimes H. \tag{48}$$

Now subtracting these matrices yields the wanted constraint matrix $R = T - Q$, but the last row is an invalid constraint since there is no $x(k + N_p + 1)$ in the state vector to compare with, so it is removed. The complete dynamics constraint is then expressed as

$$Rx = \mathbb{0}. \tag{49}$$

Since the state at every timestep has to fullfill the state constraints, this constraint is easily expressed as

$$(I_{N_p} \otimes E)x \leq (\mathbb{1}^{N_p \times 1} \otimes g_5), \tag{50}$$

where $E = \begin{bmatrix} E_1 & E_2 & E_3 & E_4 & \mathbb{0}^{N_p \times 2N_p} \end{bmatrix}$.

Lastly, this section also introduces the comfortability constraint on the acceleration. Whereas in Equation 8 the maximum acceleration was determined through $u_{max} \to a_{max}$ in continuous time, now we perform $a_{\text{comf, max}} \to u_{\text{comf, max}}$ in discrete time. This comes down to taking the discrete acceleration and bounding this, thus

$$-a_{\text{comf, max}} \leq \frac{x(k+1) - x(k)}{\Delta t} \leq a_{\text{comf, max}}. \tag{51}$$

Equivalently written using already defined matrices, this constraint becomes

$$\begin{bmatrix} \frac{-1}{\Delta t}(Q - X) \\ \frac{1}{\Delta t}(Q - X) \end{bmatrix} x \leq \mathbb{1} a_{\text{comf, max}}, \tag{52}$$

where the last row of $Q - X$ is again deleted, because it yields an invalid constraint.

2.7.4 MPC constraints

The implementation of MPC induces a number of extra constraints. These consist of the initialisation constraint and the control horizon constraint. The initialisation constraint equates the first state in the optimisation parameter to equal the initial state, v_0, or in matrix notation

$$\begin{bmatrix} 1 & \mathbb{0} \end{bmatrix} x = v_0. \tag{53}$$

The control horizon constraint is formulated as the inputs being equal after N_c timesteps, or equivalently

$$\left[\mathbb{0}^{N_c \times N_p - N_c} \quad (I_{(N_p - N_c)} - {}_1 I_{(N_p - N_c)}) \otimes \begin{bmatrix} 0 & 1 & \mathbb{0}^{1 \times 6} \end{bmatrix})\right] x = \mathbb{0}. \tag{54}$$

It is not necessary to define a terminal set for the MPC controller, since the system is stable by nature, and the space in which $v(k)$ lives is control invariant by the definition of v_{max}. Thus, the controller can not destabilise the system, and thus is the controlled system stable for any input.

2.7.5 The controller

The resultant controller takes an initial state, and predicts the future based on that with the corresponding optimal control. An example of this is depicted in Figure 7.

2.8 Closed loop MPC

The MPC control loop is now closed, and the controller will work in feedback with the plant. Now at each time step the prediction is made with the MLD model and the first input determined with that is given to the plant that evolves with the real dynamics. This introduces differences between the prediction and reality, that continuously needs to be corrected by the controller. An example with a constant reference is shown in Figure 8. The oscillations around the reference occur due to the mentioned modeling error.

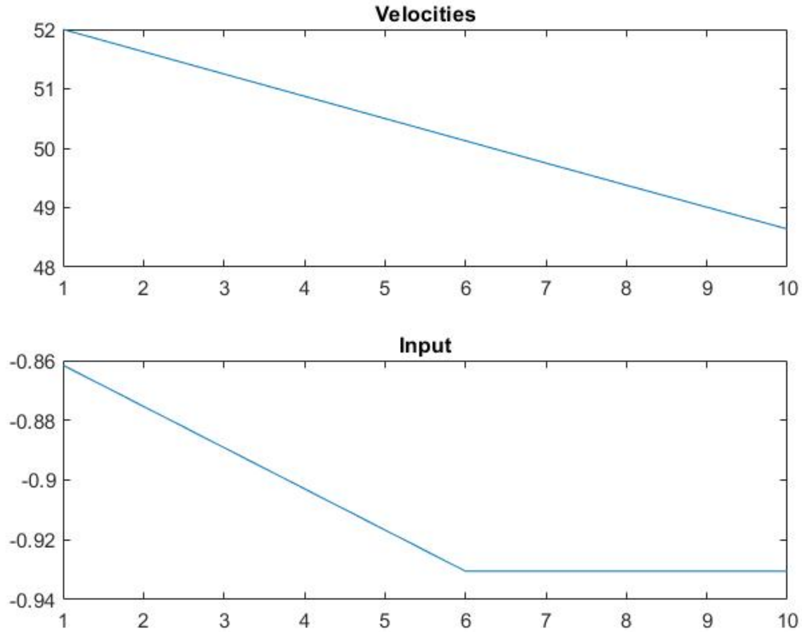


Figure 7: One prediction made by the controller with the expected state and corresponding optimal input; Here $v_0 = 52$m/s, $v_{ref} = 20$m/s, $\lambda = 0.3$, $N_p = 10$ and $N_c = 7$

2.9 Time varying reference

A time varying reference is now considered. This can be interpreted as the leading car communicating its speed and future speed to the following car. Two cases are considered, one with a shorter prediction horizon and a longer one. All of the other parameters will be constant and $\lambda = 0.1$.

2.9.1 Short prediction horizon

A pair $(N_p, N_c) = (5, 4)$ is considered. The phase plot and position evolution are shown in Figure 9. The velocity evolution and acceleration is shown in Figure 10. The velocity residual, input and change in input is shown in Figure 11.

2.9.2 Long prediction horizon

Another pair $(N_p, N_c) = (10, 9)$ is considered. The phase plot and position evolution are shown in Figure 12. The velocity evolution and acceleration is shown in Figure 13. The velocity residual, input and change in input is shown in Figure 14.

2.9.3 Comparison

A qualitative description of the differences between the longer and shorter horizon is that the longer horizon acts more conservative than the shorter one. This is most evident in the peak of the velocity evolutions. Besides from this the differences are small, both controllers track the reference quite good, they both change their inputs a lot, they both show similar residual magnitudes, and they both achieve the comfort criterion.

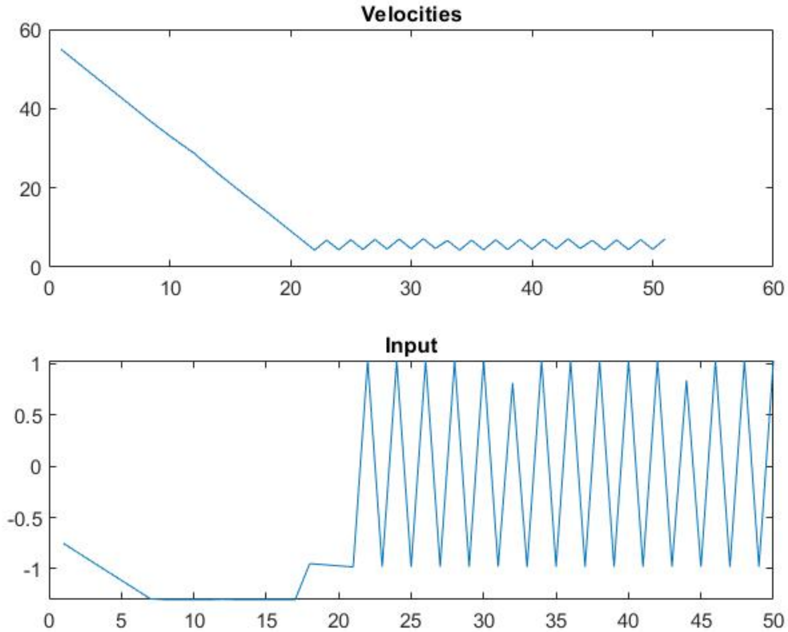


Figure 8: Closed loop control with the state and corresponding optimal input; The horizontal axis is the timestep; Here $v_0 = 55$m/s, $v_{ref} = 5$m/s, $\lambda = 0.3$, $N_p = 10$ and $N_c = 7$

2.10 Explicit MPC

Due to the fact that online implicit MPC has a quite strict bound on computation time to work, the possibility of an offline explicit MPC is investigated. Using the "MPT" Toolbox again, a PWA system is defined according to Equation 17. The comfortability constraint is added and the cost function is defined as Equation 35. Then an implicit MPC is constructed using `MPCController()`, after which this is made explicit by `MPCController.toExplicit()`. This yields an explicit controller with as its inputs the current state and reference, and the output the optimal control.

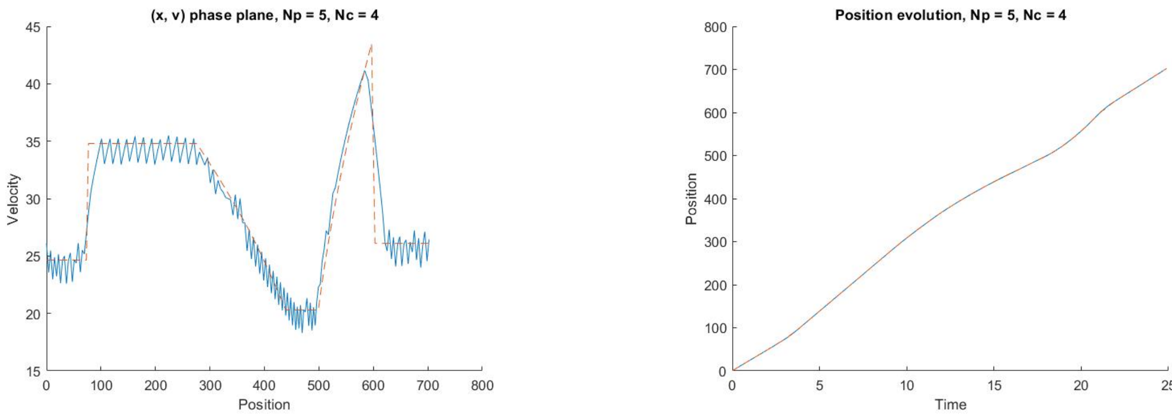
To compare the runtimes of both controllers, the exact same parameters are used. These are $N_c = N_p \in \{2, 3, 4\}$, $\lambda = 0.1$, a simulation time of 25 seconds, the time varying reference speed used before, and the runtime is only measured over the for-loop simulating real time. The average is taken of 30 runs, these are plotted in Figure 15.

It seems that the implicit method has a very slight increase in runtime based on the prediction horizon, whereas the explicit method seems to be increasingly slower. For longer prediction horizons these differences may become even larger.

3 Conclusion

The aim of this section is to evaluate achieved results. In part 1, a simple hybrid system is considered. This model could be extended by including the physical mechanism of warming and cooling a device. If this dynamics are included and the temperature rate is also measured, an optimal MPC controller could be constructed for this problem in order to minimize energy consumption.

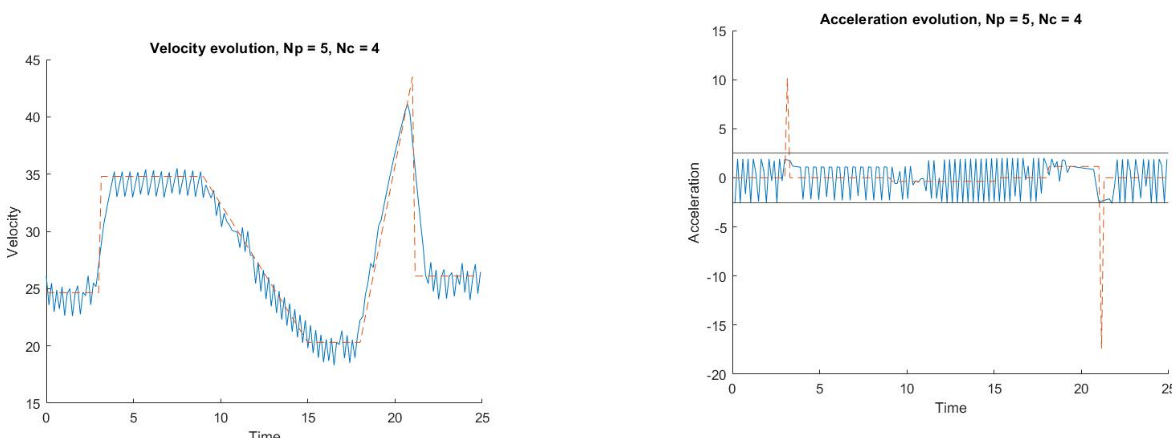
In the beginning of part 2, the friction dynamics are modeled in a simplified manner. Therefore, inevitably, an accumulative error is introduced. This is the case as the speed and friction force are correlated: an error in the friction force magnitude will result in an error in the speed magnitude and so on. It would therefore be interesting to consider the displacement progression of the car, such as visualized in figure 9b, as the displacement is the integral of the car speed and therefore also the integral of the speed error. At times 3 until 9 seconds the



(a) Phase plot

(b) Position evolution

Figure 9: Plots with the closed loop controller in blue, and the reference in striped orange

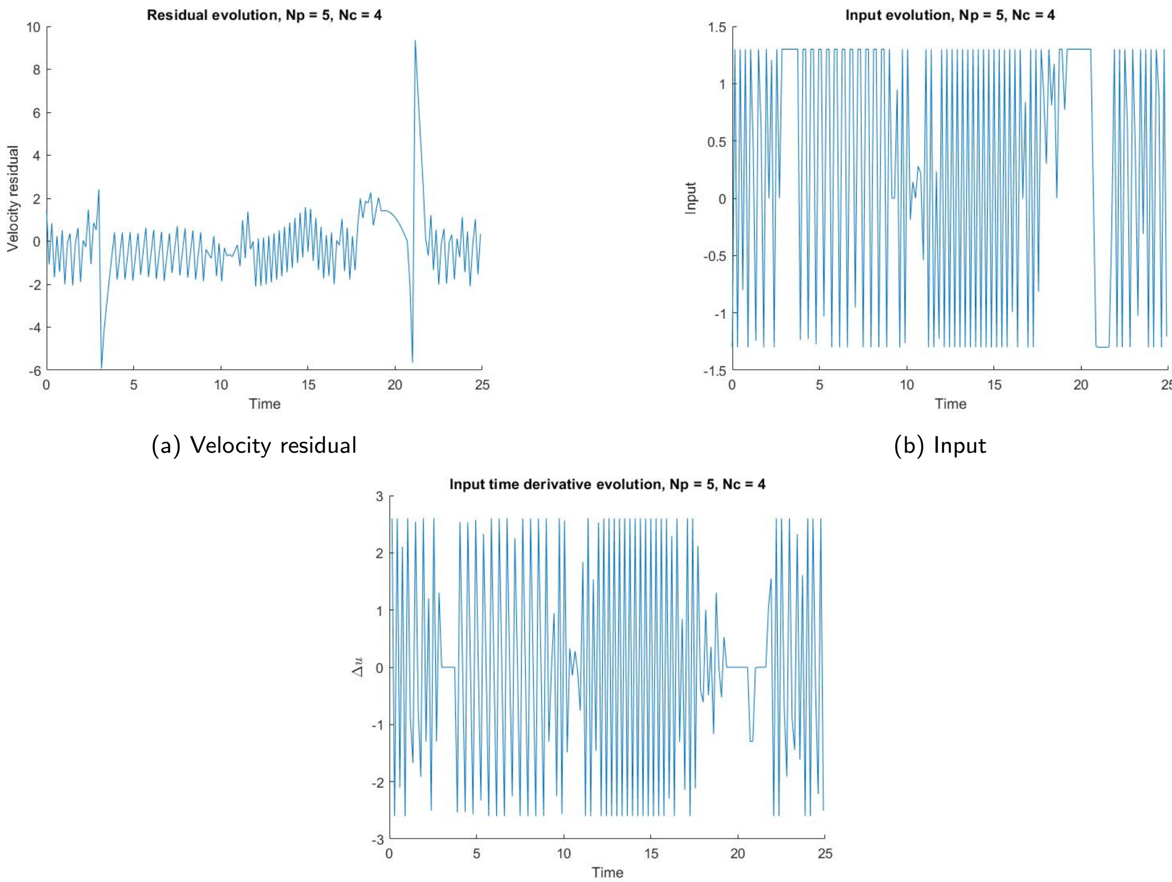


(a) Velocity evolution

(b) Acceleration with the comfortability boundary in black

Figure 10: Plots with the closed loop controller in blue, and the reference in striped orange

prescribed reference speed is constant at $v_{ref} = 1.2\alpha = 34.5\frac{m}{s}$, as is visible in Figure 3 in the exercise description. This could mean a constant error and therefore two diverging curves in the position plot. Nevertheless, the model seems to slightly swing around the reference position. As can be seen in figure 16.



(a) Velocity residual

(b) Input

(c) Change in input

Figure 11: Plots with the closed loop controller

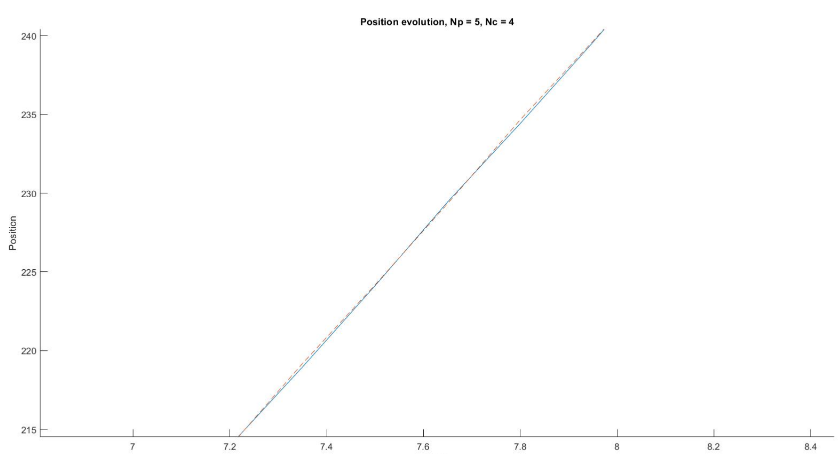
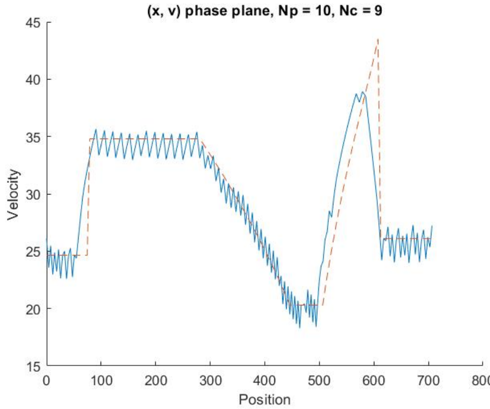
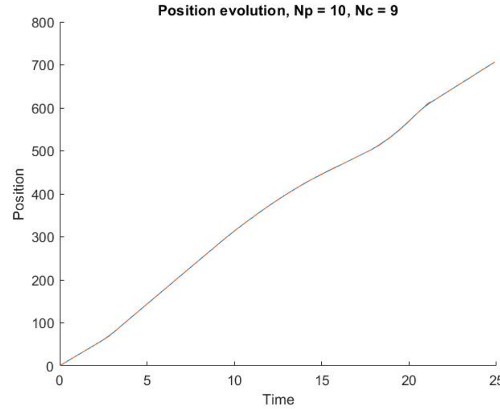


Figure 16: Close up of reference and modeled position values

The explanation for this phenomenon, has to do with the cross speed $v_{cross,2}$ which is very near the constant reference speed during this time interval. What happens, is that first the speed is modeled lower than in reality, as friction force is smaller than in reality according to figure 4, resulting in a positive speed error. The approximated

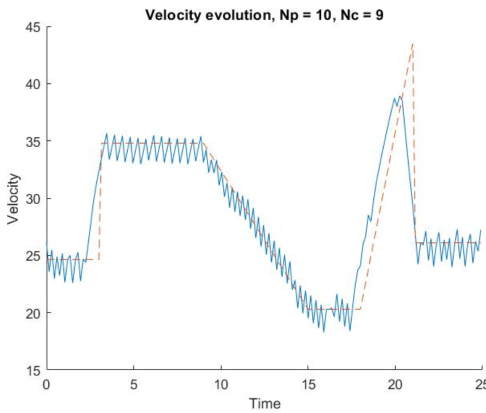


(a) Phase plot

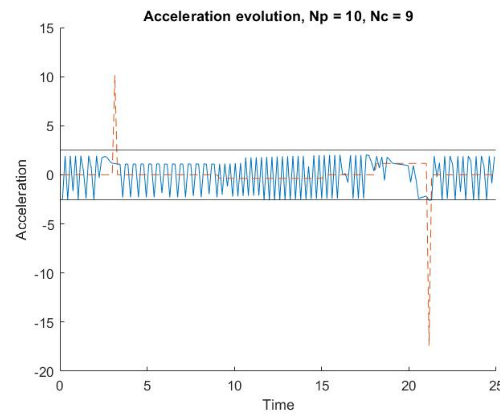


(b) Position evolution

Figure 12: Plots with the closed loop controller in blue, and the reference in striped orange



(a) Velocity evolution

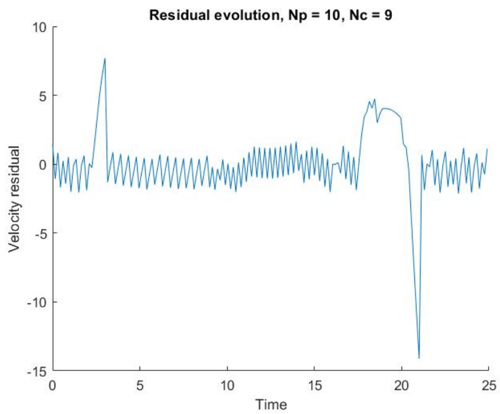


(b) Acceleration with the comfortability boundary in black

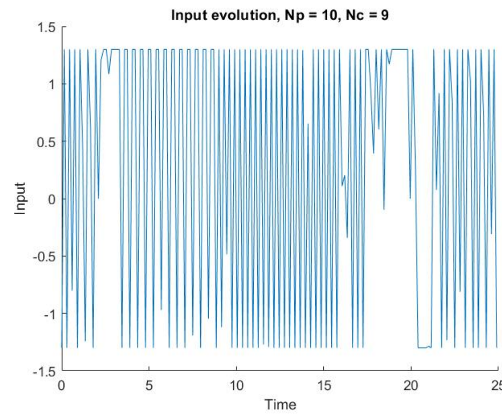
Figure 13: Plots with the closed loop controller in blue, and the reference in striped orange

speed will increase more than in reality. However, as $v_{cross,2}$ will be reached rather quickly, after a short period of time, the approximated speed is larger than in reality, resulting in a negative speed error. This explains the oscillating behaviour of the model curve in figure 16.

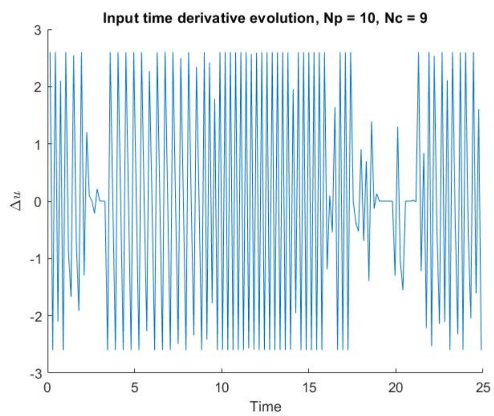
Next the MLD model, which introduces some constraints of some constraints. The relevance of these can be investigated by looking at the constraint activity, possibly, some could be redundant and therefore left out for the sake of computational efficiency. Equation 21 gives the first implied constraints, which are obviously active, as the speed is at times lower and at times higher than α, thus a simplification can not be made here. Next, the speed boundaries described in equation 23 are in this specific case never active, but neither redundant, as for a different control input these boundaries will be achieved. The constraints describing the gear switches are very important and are often active in this case and finally the control boundaries, which related constraints are given in equation 28, are also not redundant, as there is no other constraint limiting the control input and in this case, there is being operated rather close to these limits. Furthermore, as mentioned, the resulting MLD model seems to be accurate, because figure 6 indicates that only around the gear switching velocities, there the MLD model differs from the PWA model. This difference is due to differences in the gear policy in the models at the switching points.



(a) Velocity residual



(b) Input



(c) Change in input

Figure 14: Plots with the closed loop controller

Now when the control loop is closed, the dynamics in figure 8 occur. The figure seems to be linearly decreasing until the equilibrium state in which the speed and input swing around v_{ref} and 0 respectively. By closer inspection, the velocity trend seems linearly decreasing, but this is not the case. The reason why this is less obvious than in the steady state part, is because the error in the predicted velocity is relatively slow with respect to the speed derivative. Therefore the shown velocity only swings slightly around this fictional line segment from the initial speed to the steady state value in the velocity plot.

Considering section 2.9, it strikes that the controller is able to make the model follow the reference speed accurately. A shorter time horizon does a better job yielding less over-/undershoot, comparing 11a and 14a. Besides, comparing figures 9a and 12a, it is clear that the longer time horizons introduce a larger lead with respect to the reference signals. The runtimes of the two models do not differ a lot, the model with the large time horizons is only 2-3% slower than the model that includes time horizons that are twice as short.

The investigation into the explicit MPC yielded little usable results. Only that the explicit method is significantly slower than the implicit method for small horizons. Thus it is concluded that in this case and for these horizons the implicit method is more time efficient than the explicit method.

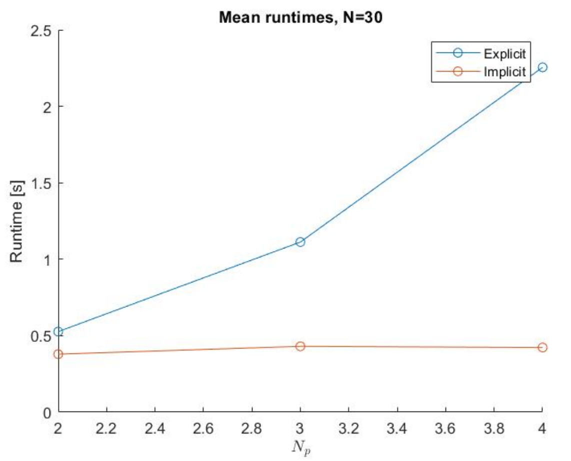


Figure 15: Runtime comparison between implicit and explicit MPC with different horizons

4 Appendix

Code

In this Appendix section, first the main code is presented. It makes use function files, which are listed thereafter.

```
clc; close all; clear; tStart = tic;
variables % Retrieve system parameters
fig = 1; % Figure number token
WorkingOn = "all"; % Token to prevent always plotting everything
%% Dynamics test
u = @(t) u_max + heaviside(t-100)*(-u_max + u_min) + heaviside(t-120)*(-u_min + u_max)...
    + heaviside(t-170)*(-u_max); % Input function
[t_0_1,y_0_1] = ode45(@(t,y) modelExact(t,y,u,vars, "")...
    , [0 t_end*step], [0 0]); % Integration of the exact model

% Plotting
if WorkingOn == "dyntest" || WorkingOn=="all"
figure(fig); fig = fig+1;
plot(t_0_1,y_0_1(:,2))
title 'ACC Car dynamics simulation'
xlabel 'time [s]'
ylabel 'speed [m/s]'
saveas(gcf,'Pics/Plot_2.1_1.jpg')

figure(fig); fig = fig+1;
plot(t_0_1, u(t_0_1))
title 'ACC Car dynamics simulation input'
xlabel 'time [s]'
ylabel 'speed [m/s]'
saveas(gcf,'Pics/Plot_2.1_2.jpg')
end

%% 2.1
v_max = ((b*u_max)/(c*(1+gamma*g(3))))^.5; % Maximum speed
a_max = (b*u_max)/(1+gamma*g(1))/m; % Maximum acceleration
a_min = (b*u_min)/(1+gamma*g(3))/m-c/m*v_max^2; % Minimum acceleration

vars.v_max = v_max; % Make struct to pass to functions later
%% 2.2

[alpha, beta] = optApprox(vars); % Determine optimal values for alpha and beta
vars.alpha = alpha; vars.beta = beta;

```

```
Vsamp = linspace(0, v_max, 200); % Speed samples
P_2 = fricApprox(Vsamp, vars); % Friction approximation

% Plotting
if WorkingOn == "2.2" || WorkingOn=="all"
figure(fig); fig = fig+1;
plot(Vsamp, P_2); hold on
plot(Vsamp, c*Vsamp.^2); hold off
title 'PWA approximation of friction force'
ylabel 'Friction force [N]'
xlabel 'speed [m/s]'
saveas(gcf,'Pics/Plot_2.2.jpg')
end

%% 2.3
x0 = [0;0]; % Initial state
u = @(t) u_max/5 + u_max/2*sin(t/2/pi); % Input sinusoid
[t_3_1,y_3_1] = ode45(@(t,y) modelExact(t,y,u,vars, "gearlock")...
    , [0 t_end*step], x0); % Integrate exact model
[t_3_2,y_3_2] = ode45(@(t,y) modelPWA(t,y,u,vars, "gearlock")...
    , [0 t_end*step], x0); % Integrate PWA model

% Plotting
if WorkingOn == "2.3" || WorkingOn=="all"
    fig = plot2_3(t_3_1,t_3_2, u, y_3_1,y_3_2, Vsamp, P_2, vars, fig);
end

%% 2.5
T = 1:t_end; % Define time vector
U = u(T*dt).*ones(size(T)); % Define input

[t_5_1, y_5_1] = FEuler(x0, u, vars); % Forward Euler integration of
                                              % the PWA model
[t_5_2, y_5_2] = ode45(@(t,y) modelPWA(t,y,u,vars, "")...
    , [0 t_end*dt], x0); % ode45 integration of PWA model

% Plotting
if WorkingOn == "2.5" || WorkingOn=="all"
figure(fig); fig = fig+1;
plot(t_5_2,u(t_5_2)); hold on
plot(T*dt,U, "--");hold off
title 'Integration comparison input'
ylabel 'input'
xlabel 'time [s]'
legend("u(t)", 'Interpreter', 'latex')
saveas(gcf,'Pics/Plot_2.5_1.jpg')

figure(fig); fig = fig+1;
plot(t_5_2,y_5_2(:, 2)); hold on
plot(t_5_1,y_5_1(2, :), "--"); hold off
title 'Integration comparison'
ylabel 'speed [m/s]'
xlabel 'time [s]'
legend("ODE45", "FE", 'Interpreter', 'latex')
saveas(gcf,'Pics/Plot_2.5_2.jpg')
end

%% 2.6
sys = MLD(vars); % Create MLD model struct
% Unpack system
A = sys.A;
B1 = sys.B1; B2 = sys.B2; B3 = sys.B3;
E1 = sys.E1; E2 = sys.E2; E3 = sys.E3; E4 = sys.E4;
g5 = sys.g5;

H = [A B1 B2 B3];
nx = size(A, 1);
nu = size(B1, 2);
```

```
n = size(H, 2);

A = [[1 zeros(1, n-1)];... % v and u are as predefined
     [0 1 zeros(1, n-2)];...
     [E1 E2 E3 E4]]; % state is valid
C = [1 zeros(1, n-1)]; % Minimise the speed
ctype = ['SS', repmat('U', 1,size(E1,1))]; % Two equalities, rest upper bounds
vartype = 'CCBBBCCC'; % State either Continuous or Binary

V = 1:v_max; % Sample speeds
figure(fig); fig = fig+1;
subplot(2,2, 1:2); hold on
for u = u_min:0.2:u_max
    u_ = @(t) u*t/t;
    d = [];
    for v = V
        %MLD model
        b = [v; u; g5];
        [xMLD,~,status,~] = glpk (C,A,b,[],[],ctype,vartype);

        xPmld = H*xMLD;

        xPfe = [0;v] + vars.dt* modelPWA(1, [0;v], u_, vars, "");

        d = [d xPmld - xPfe(2)];
    end
    plot(V, d)
end
xlabel("Speed")
ylabel("$v_{MLD} - v_{PWA}$", 'Interpreter', 'latex')

subplot(2,2, 3); hold on
V = 14.5:0.05:15.5; % Sample speeds
for u = u_min:0.2:u_max
    u_ = @(t) u*t/t;
    d = [];
    for v = V
        %MLD model
        b = [v; u; g5];
        [xMLD,~,status,~] = glpk (C,A,b,[],[],ctype,vartype);

        xPmld = H*xMLD;

        xPfe = [0;v] + vars.dt* modelPWA(1, [0;v], u_, vars, "");

        d = [d xPmld - xPfe(2)];
    end
    plot(V, d, "x-")
end
xlabel("Speed")
ylabel("$v_{MLD} - v_{PWA}$", 'Interpreter', 'latex')

subplot(2,2, 4); hold on
V = 29.5:0.05:30.5; % Sample speeds
for u = u_min:0.2:u_max
    u_ = @(t) u*t/t;
    d = [];
    for v = V
        %MLD model
        b = [v; u; g5];
        [xMLD,~,status,~] = glpk (C,A,b,[],[],ctype,vartype);

        xPmld = H*xMLD;

        xPfe = [0;v] + vars.dt* modelPWA(1, [0;v], u_, vars, "");

        d = [d xPmld - xPfe(2)];
    end
```

```
    plot(V, d, "x-")
end
xlabel("Speed")
ylabel("$v_{MLD} - v_{PWA}$", 'Interpreter', 'latex')



%% 2.7
% From here on the state x only consists of the speed v
x0 = 52; % Initial velocity
Np = 10; % Prediction horizon
Nc = 7; % Control horizon
lambda = 0.3; % Cost function lambda

sys = MLD(vars); % Create MLD model struct
sys.dt = dt;
sys.a_comf = 2.5; % Comfortability threshold
[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % All time invariant
                                %constraints regarding v, u, delta, z

vRef = 20 *ones(Np, 1); % Constant reference speed
[C, M, b2] = costFunc(sys, vRef, Np, lambda); % Get cost function and
                    % additional constraints for the auxilliary states
                    % originating from the norms

K = [[F, zeros(size(F,1),2*Np)];... % Compile constraints matrix
            M];

L = [b1; b2]; % Compile constraints vector

if WorkingOn == "2.7" || WorkingOn=="all"
    % Perform MPC and retrieve optimal input
    [u_2_7, ~] = getOptInput(x0, vRef, sys, K, L, C, Np, Neq, Nleq, "plot");
    fig = fig+1;
end

%% 2.8
T = 50; % Define number of timesteps
x0 = 55; % Inital state
vRef = 5 *ones(T+Np, 1); % Constant reference

Np = 10; % Prediction horizon
Nc = 7; % Control horizon
lambda = 0.3; % Cost function lambda

[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % Retrieve all time
                        % invariant constraints regarding v, u, delta, z

u_2_8 = zeros(T, nu);
x_2_8 = [x0; zeros(T-1, nx)];
for k = 1:T
vRef_k = vRef(k:k+Np-1); % Get reference at time k
[C, M, b2] = costFunc(sys, vRef_k, Np, lambda); % Get cost function

K = [[F, zeros(size(F,1),2*Np)];...
    M]; % Compile constraints matrix

L = [b1; b2]; % Compile constraints vector

[u_2_8(k, :), ~] = getOptInput(x_2_8(k, :), vRef_k, sys, K, L, C...
    , Np, Neq, Nleq, ""); % Get optimal input
x_2_8(k+1, :) = x_2_8(k, :) + modelExact(k*dt, [0;x_2_8(k, :)]...
    , u_2_8(k, :), vars, "SingleState"); % Integrate timestep with input
end

% Plotting
if WorkingOn == "2.8" || WorkingOn=="all"
    figure(fig); fig = fig+1;
```

```
    subplot(2, 1, 1)
    plot(x_2_8);
    title("Velocities")

    subplot(2, 1, 2)
    plot(u_2_8);
    title("Input")
end

%% 2.9
x0 = 0.9*alpha; % Define initial state

Tarr = 0:dt:25; % Create time array
N = max(size(Tarr)); % Number of timesteps

vRef = zeros(N,nx); % Create reference speed array
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

lambda = 0.1; % Prediction horizon
Np = 5; % Prediction horizon
Nc = 4; % Control horizon

[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % All time invariant
                                %constraints regarding v, u, delta, z

u_2_9 = zeros(N, nu);
x_2_9 = [x0; zeros(N-1, nx)];
for k = 1:N % Same loop as in 2.8
vRef_k = vRef(k:k+Np-1);
[C, M, b2] = costFunc(sys, vRef_k, Np, lambda);

K = [[F, zeros(size(F,1),2*Np)];...
    M];

L = [b1; b2];

[u_2_9(k, :), ~] = getOptInput(x_2_9(k, :), vRef_k, sys, K, L, C, Np, Neq, Nleq, "");
x_2_9(k+1, :) = x_2_9(k, :) + modelExact(k*dt, [0;x_2_9(k, :)], u_2_9(k, :)...
        , vars, "SingleState");
end

if WorkingOn == "2.9" || WorkingOn=="all"
    fig = plot2_9(0, x_2_9, u_2_9, vRef(1:N), sys.a_comf, Tarr, dt, fig, Np, Nc);
end

% Second Np, Nc combo, exactly the same as above otherwise
Np = 10; % Prediction horizon
Nc = 9; % Control horizon

vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % All time invariant
                                %constraints regarding v, u, delta, z

u_2_9b = zeros(N, nu);
x_2_9b = [x0; zeros(N-1, nx)];
for k = 1:N
vRef_k = vRef(k:k+Np-1);
[C, M, b2] = costFunc(sys, vRef_k, Np, lambda);

K = [[F, zeros(size(F,1),2*Np)];...
```

```
    M];

L = [b1; b2];

[u_2_9b(k, :), ~] = getOptInput(x_2_9b(k, :), vRef_k, sys, K, L, C, Np, Neq, Nleq, "");
x_2_9b(k+1, :) = x_2_9b(k, :) + modelExact(k*dt, [0;x_2_9b(k, :)], u_2_9b(k, :)...
        , vars, "SingleState");
end

if WorkingOn == "2.9" || WorkingOn=="all"
    fig = plot2_9(0, x_2_9b, u_2_9b, vRef(1:N), sys.a_comf, Tarr, dt, fig, Np, Nc);
end

%% 2.10

% Create explicit formulas
Np = 2;
explMPC2 = getexplMPCfun(vars, lambda, Np);

Np = 3;
explMPC3 = getexplMPCfun(vars, lambda, Np);

Np = 4;
explMPC4 = getexplMPCfun(vars, lambda, Np);

% Runtime counters
toc2 = 0; toc2i = 0; toc3 = 0; toc3i = 0; toc4 = 0; toc4i = 0;

for idx = 1:30
% Simulate with horizon 2
Np = 2;
vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

u_2_10a = zeros(N, nu);
x_2_10a = [x0; zeros(N-1, nx)];
tic
for k = 1:N
    u_2_10a(k) = explMPC2.evaluate(x_2_10a(k, :), 'x.reference', vRef(k));
    x_2_10a(k+1, :) = x_2_10a(k, :) + modelExact(k*dt, [0;x_2_10a(k, :)], u_2_10a(k, :)...
        , vars, "SingleState");
end
toc2 = toc2 + toc;

% Implicit method
Np = 2; % Prediction horizon
Nc = 2; % Control horizon

vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % All time invariant
                                %constraints regarding v, u, delta, z

u_2_10ai = zeros(N, nu);
x_2_10ai = [x0; zeros(N-1, nx)];
tic
for k = 1:N
vRef_k = vRef(k:k+Np-1);
[C, M, b2] = costFunc(sys, vRef_k, Np, lambda);

K = [[F, zeros(size(F,1),2*Np)];...
```

```
    M];

L = [b1; b2];

[u_2_10ai(k, :), ~] = getOptInput(x_2_10ai(k, :), vRef_k, sys, K, L, C, Np, Neq, Nleq, "");
x_2_10ai(k+1, :) = x_2_10ai(k, :) + modelExact(k*dt, [0;x_2_10ai(k, :)], u_2_10ai(k, :)...
        , vars, "SingleState");
end
toc2i = toc2i + toc;


% Simulate with horizon 3
Np = 3;
vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

u_2_10b = zeros(N, nu);
x_2_10b = [x0; zeros(N-1, nx)];
tic
for k = 1:N
    u_2_10b(k) = explMPC3.evaluate(x_2_10b(k, :), 'x.reference', vRef(k));
    x_2_10b(k+1, :) = x_2_10b(k, :) + modelExact(k*dt, [0;x_2_10b(k, :)], u_2_10b(k, :)...
        , vars, "SingleState");
end
toc3 = toc3 + toc;


% Implicit method
Np = 3; % Prediction horizon
Nc = 3; % Control horizon

vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % All time invariant
                                %constraints regarding v, u, delta, z

u_2_10bi = zeros(N, nu);
x_2_10bi = [x0; zeros(N-1, nx)];
tic
for k = 1:N
vRef_k = vRef(k:k+Np-1);
[C, M, b2] = costFunc(sys, vRef_k, Np, lambda);

K = [[F, zeros(size(F,1),2*Np)];...
    M];

L = [b1; b2];

[u_2_10bi(k, :), ~] = getOptInput(x_2_10bi(k, :), vRef_k, sys, K, L, C, Np, Neq, Nleq, "");
x_2_10bi(k+1, :) = x_2_10bi(k, :) + modelExact(k*dt, [0;x_2_10bi(k, :)], u_2_10bi(k, :)...
        , vars, "SingleState");
end
toc3i = toc3i + toc;


% Simulate with horizon 4
Np = 4;
vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
```

```
vRef = [vRef; vRef(end)*ones(Np,nx)];

u_2_10c = zeros(N, nu);
x_2_10c = [x0; zeros(N-1, nx)];
tic
for k = 1:N
    u_2_10c(k) = explMPC4.evaluate(x_2_10c(k, :), 'x.reference', vRef(k));
    x_2_10c(k+1, :) = x_2_10c(k, :) + modelExact(k*dt, [0;x_2_10c(k, :)], u_2_10c(k, :)...
        , vars, "SingleState");
end
toc4 = toc4 + toc;


% Implicit method
Np = 4; % Prediction horizon
Nc = 4; % Control horizon

vRef = zeros(N,nx);
for k = 1:N
    vRef(k) = vref(alpha, Tarr(k));
end
vRef = [vRef; vRef(end)*ones(Np,nx)];

[F, b1, Neq, Nleq] = optContstraint(sys, Np, Nc); % All time invariant
                                %constraints regarding v, u, delta, z

u_2_10ci = zeros(N, nu);
x_2_10ci = [x0; zeros(N-1, nx)];
tic
for k = 1:N
vRef_k = vRef(k:k+Np-1);
[C, M, b2] = costFunc(sys, vRef_k, Np, lambda);

K = [[F, zeros(size(F,1),2*Np)];...
    M];

L = [b1; b2];

[u_2_10ci(k, :), ~] = getOptInput(x_2_10ci(k, :), vRef_k, sys, K, L, C, Np, Neq, Nleq, "");
x_2_10ci(k+1, :) = x_2_10ci(k, :) + modelExact(k*dt, [0;x_2_10ci(k, :)], u_2_10ci(k, :)...
        , vars, "SingleState");
end
toc4i = toc4i + toc;
end

% Average runtimes
toc2 = toc2/idx;
toc2i = toc2i/idx;
toc3 = toc3/idx;
toc3i = toc3i/idx;
toc4 = toc4/idx;
toc4i = toc4i/idx;

if WorkingOn == "2.10" || WorkingOn=="all"
figure(fig); fig = fig+1; hold on
plot([2 3 4], [toc2 toc3 toc4], "o-");
plot([2 3 4], [toc2i toc3i toc4i], "o-");
xlabel("$N_p$", 'Interpreter', 'latex')
ylabel("Runtime [s]")
title("Mean runtimes, N=30")
legend("Explicit", "Implicit")
end

tEnd = toc(tStart)
```

```
% Define all system paramaters as specified in the assignment
m = 800; % mass
c = 0.4; % friction constant
```

```
b = 3700; % driving constant
u_max = 1.3; % Maximum input
u_min = -1.3; % Minimum input
alphamax = 2.5; % Maximum acceleration
gamma = 0.87; % Gear constant
v12 = 15; % gear 2 switch
v23 = 30; % gear 3 switch

g = [1,2,3]; % gears
step=.1; % time step size
t_end = 3*10^3; % simulation time
dt = 0.15; % time step size for integration

% Make struct to pass to functions later
vars.m = m;
vars.c = c;
vars.b = b;
vars.u_max = u_max;
vars.u_min = u_min;
vars.gamma = gamma;
vars.v12 = v12;
vars.v23 = v23;
vars.g = g;
vars.t_end = t_end;
vars.dt = dt;
vars.a_comf = 2.5;
```

```
function [dx] = modelExact(t, y, u, vars, mode)
%MODELEXACT The exact differential equations

%% Determine u (either a function or a double)
if convertCharsToStrings(class(u)) == "function_handle"
    ut = u(t);
else
    ut = u;
end

%% Determine the gear
if mode == "gearlock"
    gear = 1;
elseif y(2)<vars.v12
    gear = 1;
elseif y(2)<vars.v23
    gear=2;
else
    gear=3;
end

%% Determine the derivative
if mode == "SingleState"
    dx = 1/vars.m * vars.b/(1+vars.gamma*gear)*ut - 1/vars.m*vars.c*y(2)^2;
else
    dx = [ y(2);...
        1/vars.m * vars.b/(1+vars.gamma*gear)*ut - 1/vars.m*vars.c*y(2)^2];
end

end
```

```
function [alpha, beta] = optApprox(vars)
syms alph bet % Symbolic variable alpha and beta
syms v % Symbolic variable for the speed
c = vars.c; vmax = vars.v_max;

r1 = (c*v^2 - bet/alph *v)^2; % region 1
r2 = (c*v^2 - (c*vmax^2 - bet)/(vmax - alph) *v - bet + ...
    (c*vmax^2 - bet)/(vmax - alph) *alph)^2; % region 2

A1 = int(r1, v, 0, alph); % Area 1
```

```
A2 = int(r2, v, alph, vmax); % Area 2

A = A1 + A2; % Total area

d_alph = diff(A, alph, 1); % deriv of A wrt to alpha
d_bet = diff(A, bet, 1); % deriv of A wrt to beta

dd_alph = diff(A, alph, 2); % double deriv of A wrt to alpha
dd_bet = diff(A, bet, 2); % double deriv of A wrt to beta

% Find extrema of A
[solAlph, solBet] = solve([d_alph == 0, d_bet == 0], [alph, bet]);
solAlph = double(solAlph);
solBet = double(solBet);

% Identify valid solutions
idx = find(solAlph>0 & solAlph<vmax & solBet>0 & solBet<(c*vmax^2));

alpha = solAlph(idx); % Grab valid values
beta = solBet(idx);

% Possibly plot the values of A dependent on alpha and beta
if 1==0
    fA = matlabFunction(A); % Make a function of alpha, beta
    figure()
    fsurf(fA, [0 vmax 0 c*vmax^2])
end

% Confirm that the second derivative is positive
val_dd_alph = double(subs(dd_alph, [alph, bet], [alpha, beta]));
val_dd_bet = double(subs(dd_bet, [alph, bet], [alpha, beta]));

end
```

```
function [fig] = plot2_3(t_3_1,t_3_2, u, y_3_1,y_3_2, Vsamp, P_2, vars, fig)
%PLOT2_3 Make plots for Q2.3

%% Plot input
figure(fig); fig = fig+1;
plot(t_3_1,u(t_3_1));
title 'PWA approximation comparison input'
ylabel 'input'
xlabel 'time [s]'
% legend("Exact model", "PWA", 'Interpreter', 'latex')
saveas(gcf,'Pics/Plot_2.3_1.jpg')

%% Compare velocities
figure(fig); fig = fig+1;
plot(t_3_1,y_3_1(:, 2)); hold on
plot(t_3_2,y_3_2(:, 2)); hold off
title 'PWA approximation comparison'
ylabel 'speed [m/s]'
xlabel 'time [s]'
legend("Exact model", "PWA", 'Interpreter', 'latex')
saveas(gcf,'Pics/Plot_2.3_2.jpg')

%% Plot friction forces per speed, overlain with time evolution of models
figure(fig); fig = fig+1;
plot(Vsamp, P_2); hold on
plot(Vsamp, vars.c*Vsamp.^2);
plot(y_3_1(:, 2), t_3_1 .* P_2(end)/t_3_1(end));
plot(y_3_2(:, 2), t_3_2 .* P_2(end)/t_3_2(end)); hold off
title 'PWA approximation comparison'
ylabel 'Friction/time'
xlabel 'Speed'
legend("PWA friction", "Exact friction", "PWA evolution"...
    , "Exact evolution", 'Interpreter', 'latex')
saveas(gcf,'Pics/Plot_2.3_3.jpg')
```

```
end
```

```
function [P] = fricApprox(v, vars)
%FRICAPPROX Approximates the friction force
%   Symbol in report is P(v)

P = zeros(size(v));

boundaryIdx = find(v<vars.alpha, 1, 'last' ); % Get switching point between functions
if all(v>vars.alpha) % or if every speed is above alpha
    boundaryIdx = 0;
end


P(1:boundaryIdx) = vars.beta/vars.alpha .* v(1:boundaryIdx); % if v<alpha

P(boundaryIdx+1:end) = (vars.c*vars.v_max^2 - vars.beta)/(vars.v_max - vars.alpha)...
    .* v(boundaryIdx+1:end) + vars.v_max/(vars.v_max-vars.alpha)*vars.beta ...
    - vars.c*vars.v_max^2/(vars.v_max-vars.alpha)*vars.alpha;% if v>alpha

end
```

```
function [T, x] = FEuler(x0, u, vars)
%FEULER Forward Euler integration with initial condition x0 and input u

%% Initialisation
x = zeros(2, vars.t_end);
x(:, 1) = x0;
T = 1:vars.t_end;
T = T*vars.dt;

%% Integration
for t = 2:vars.t_end
    x(:, t) = x(:, t-1) + vars.dt* modelPWA(t*vars.dt, x(:, t-1), u, vars, "");
end

end
```

```
function [sys] = MLD(vars)
%MLD Returns a struct with all of the MLD matrices

%% Unpack variables
dt = vars.dt;
m = vars.m;
a = vars.alpha;
uv = vars.v_max;
uu = vars.u_max;
lu = vars.u_min;
v12 = vars.v12;
v23 = vars.v23;
k = vars.beta/vars.alpha;
q = (vars.c*vars.v_max^2 - vars.beta)/(vars.v_max - vars.alpha);
r = vars.beta - q*vars.alpha;

%% Driving force
Fd = vars.b./(1+vars.gamma.*vars.g);

%% Define matrices
sys.A = 1 - dt*k/m;
sys.B1 = dt*Fd(1)/m;
sys.B2 = [0, 0, -dt*r/m];
sys.B3 = dt/m * [Fd(2)-Fd(1), Fd(3)-Fd(2), k-q];

sys.E1 = zeros(22, 1);
sys.E1([1 6 10 12 18]) = 1;
sys.E1([2 5 9 11 17]) = -1;

```

```
sys.E2 = zeros(22, 1);
sys.E2([3 16 22]) = 1;
sys.E2([4 15 21]) = -1;

sys.E3 = zeros(22, 3);
sys.E3(5:10, 3) = [a; -uv+a-eps; -uv; 0; 0; uv];
sys.E3(11:16, 1) = [v12; -uv+v12-eps; -uu; lu; -lu; uu];
sys.E3(17:22, 2) = [v23; -uv+v23-eps; -uu; lu; -lu; uu];

sys.E4 = zeros(22, 3);
sys.E4(7:10, 3) = [1; -1; 1; -1];
sys.E4(13:16, 1) = [1; -1; 1; -1];
sys.E4(19:22, 2) = [1; -1; 1; -1];

sys.g5 = zeros(22, 1);
sys.g5([1 10]) = uv;
sys.g5([3 16 22]) = uu;
sys.g5([4 15 21]) = -lu;
sys.g5(6) = a - eps;
sys.g5(12) = v12 - eps;
sys.g5(18) = v23 - eps;
end
```

```
function [dx] = modelPWA(t, y, u, vars, mode)
%MODELEXACT The piecewise affine approximation of the differential equations

%% Determine u (either a function or a double)
if convertCharsToStrings(class(u)) == "function_handle"
    ut = u(t);
end

%% Determine the gear
if mode == "gearlock"
    gear = 1;
elseif y(2)<=vars.v12
    gear = 1;
elseif y(2)<=vars.v23
    gear=2;
else
    gear=3;
end

%% Determine the derivative
dx = [ y(2);...
        1/vars.m * vars.b/(1+vars.gamma*gear)*ut...
        - 1/vars.m*fricApprox(y(2), vars)];

end
```

```
function [F, b, Neq, Nleq] = optContstraint(sys, Np, Nc)
%OPTCONTSTRAINT Returns the time invariant constraint matrix F and vector b
%such that the constraints F x ~ b can be passed to the optimiser, where
%the relation ~ is to be specified in glpk.m. Neq is the number of ==
%constraints, and Nleq the number of <= constraints. F will begin with all
%of the == constraints, then the <=, then the >=.

%% Unpack system
A = sys.A;
B1 = sys.B1; B2 = sys.B2; B3 = sys.B3;
E1 = sys.E1; E2 = sys.E2; E3 = sys.E3; E4 = sys.E4;
g5 = sys.g5;

H = [A B1 B2 B3];
nx = size(A, 1);
nu = size(B1, 2);
n = size(H, 2);

aComf = sys.a_comf;
```

```
dt = sys.dt;

%% Create dynamic projection constraint (R x == 0)
T = kron(eye(Np), H); % Block diagonal with H

q = [eye(nx) zeros(nx, n-nx)]; % Extract x(k) for one timestep
Q = kron(diag(ones(Np-1,1),1),q); % Extract x(k+1) for all timesteps

R = T-Q; % x+ - H x = 0
R((end-nx+1):end,:) = []; % Remove zero row

%% Create logical constraints (E x <= G)
e = [E1 E2 E3 E4];
E = kron(eye(Np), e); % Block diagonal with E
G = kron(ones(Np, 1), g5); % Repeat g5 Np times

%% Create constant u constraint after Nc (W x == 0)
w = [zeros(nu, nx) eye(nu) zeros(nx, n-nx-nu)]; % Extract u(k) for one timestep
W = kron(eye(Np)-diag(ones(Np-1,1),-1),w); % u(k) - u(k+1) = 0
W = W(Nc:end, :); % Only from Nc and onwards

%% Guarantee comfortabillity constraint ( P x <= p)
X = kron(eye(Np), q); % Extract all states

dX = Q - X; % x(k+1) - x(k)
dX(end, :) = []; % Last row is invalid
P = [-1/dt*dX;...
      1/dt*dX]; % Double bound

p = aComf * ones(size(P,1), 1); % Constraint vector

%% Compile to F
F = [R; W; E; P];
b = [zeros(size([R;W],1),1); G; p];

%% Determine Neq, Nleq
Neq  = size([R;W], 1); % Number of equality constraints
Nleq = size([E;P], 1); % Number of inequality constraints
end
```

```
function [c, M, b] = costFunc(sys, vRef, Np, lambda)
%COSTFUNC Return the c matric for the minimisation problem min c' x with
%extra constraints M x <= b

%% Unpack system
A = sys.A;
B1 = sys.B1; B2 = sys.B2; B3 = sys.B3;
E1 = sys.E1; E2 = sys.E2; E3 = sys.E3; E4 = sys.E4;
g5 = sys.g5;

H = [A B1 B2 B3];
nx = size(A, 1);
nu = size(B1, 2);
n = size(H, 2);

%% The tracking cost
c_x = [zeros(1,Np*n) ones(1, Np) zeros(1,Np)]; % Sum rho_x

%% The input cost
c_u = lambda*[zeros(1,Np*n) zeros(1,Np) ones(1, Np)]; % Sum rho_u

%% The tracking constraints
x_ = [eye(nx) zeros(nx, n-nx)];
X = [kron(eye(Np), x_), zeros(Np*nx, 2*Np)]; % states

rhoX = [zeros(Np, Np*n) eye(Np) zeros(Np)]; % Extract rho_x from the state

M_x = [X - rhoX;...
```

```
        -X - rhoX]; % Double bounded matrix

b_x = [vRef; -vRef]; % Double bounded vector

%% The input constraints
u_ = [zeros(nu, nx) eye(nu) zeros(nx, n-nx-nu)];
U = [kron(eye(Np), u_), zeros(Np*nx, 2*Np)]; % inputs

rhoU = [zeros(Np, Np*n) zeros(Np) eye(Np)]; % Extract rho_u from the state

M_u = [U - rhoU;...
        -U - rhoU]; % Double bounded matrix

b_u = zeros(size(M_u, 1), 1); % Double bounded vector

%% Compile matrices (M x <= b)
c = c_x + c_u; % Cost function
M = [M_x; M_u]; % Auxilliary constraints matrix
b = [b_x; b_u]; % Auxilliary constraints vector

end
```

```
function [u, x] = getOptInput(x0, vRef, sys, K, L, C, Np, Neq, Nleq, plt)
%GETOPTINPUT Determine through MPC the optimal control input at time k
%% Unpack system
A = sys.A;
B1 = sys.B1; B2 = sys.B2; B3 = sys.B3;
E1 = sys.E1; E2 = sys.E2; E3 = sys.E3; E4 = sys.E4;
g5 = sys.g5;

H = [A B1 B2 B3];
nx = size(A, 1);
nu = size(B1, 2);
n = size(H, 2);

%% Add constraint x0 == x0 (Ox == o)
O = [eye(nx), zeros(nx, size(K,2)-nx)]; % Matrix
o = x0; % Vector

%% Define and constrain terminal set ( |x_Np - xRef| <= TerSet )
TerSet = 58; % This is an inactive constraint by definition, since the
                        % whole of the state space is control invariant
S = [zeros(2*nx,n*(Np-1)) [eye(nx);-eye(nx)] zeros(2*nx,n-nx)...
    zeros(2*nx,2*Np)]; % Grab the last state
s = TerSet * ones(2*nx, 1) + [eye(nx);-eye(nx)]*vRef(end); % Vector, double
                                                            % bounded

%% Perform optimisation
ConstrA = [K;O;S]; % Compile constraint matrices
ConstrB = [L;o;s]; % Compile constraint vectors

ctype = [repmat('S', 1, Neq),... R and W ==
        repmat('U', 1, Nleq+4*Np),... E, P and M <=
        repmat('S', 1, nx),... Initial condition ==
        repmat('U', 1, 2*nx)]; % Terminal set <=
vartype = [repmat('CCBBBCCC', 1, Np), ... Np number of states
           repmat('C', 1, nx*Np), ... nx*Np number of aux variables
           repmat('C', 1, nu*Np)]; %  nu*Np number of aux variables

% Perform optimisation
[xN,~,status,~] = glpk (C,ConstrA,ConstrB,[],[],ctype,vartype);
% status
%% Return first input and state
u = xN(nx+1:nx+nu);
x = xN(n+1:n+nx);

%% Possibly plot the predicted trajectory
if plt == "plot"
```

```
    figure();
    subplot(2, 1, 1)
    x_ = [eye(nx) zeros(nx, n-nx)];
    X = [kron(eye(Np), x_), zeros(Np*nx, 2*Np)]; % states
    plot(X*xN);
    title("Velocities")

    subplot(2, 1, 2)
    u_ = [zeros(nu, nx) eye(nu) zeros(nx, n-nx-nu)];
    U = [kron(eye(Np), u_), zeros(Np*nx, 2*Np)]; % inputs
    plot(U*xN);
    title("Input")
end
end
```

```
function [fig] = plot2_9(x0, v, u, vRef, aComf, t, dt, fig, Np, Nc)
%PLOT2_9 Plot all of the plots asked for in Q2.9

N = max(size(t));
%% Integrate v to obtain x
% x+ = x + dt*v
dx = dt.*v;
dxRef = dt.*vRef;
x = [x0; zeros(N-1, 1)];
xRef = [x0; zeros(N-1, 1)];
for k = 1:N
    x(k+1) = x(k) + dx(k);
    xRef(k+1) = x(k) + dxRef(k);
end
x = x(1:end-1);
xRef = xRef(1:end-1);
v = v(1:end-1);

%% Plot phase plane
figure(fig); fig = fig+1; hold on
plot(x, v)
plot(xRef, vRef, "--")
xlabel("Position")
ylabel("Velocity")
title("(x, v) phase plane, Np = " + Np + ", Nc = " + Nc)
hold off

%% Plot x
figure(fig); fig = fig+1; hold on
plot(t, x)
plot(t, xRef, "--")
xlabel("Time")
ylabel("Position")
title("Position evolution, Np = " + Np + ", Nc = " + Nc)
hold off

%% Plot v
figure(fig); fig = fig+1; hold on
plot(t, v)
plot(t, vRef, "--")
xlabel("Time")
ylabel("Velocity")
title("Velocity evolution, Np = " + Np + ", Nc = " + Nc)
hold off

%% Plot acceleration
a = diff(v);
aRef = diff(vRef);
figure(fig); fig = fig+1; hold on
plot(t(2:end), a)
plot(t(2:end), aRef, "--")
yline(aComf, "k")
yline(-aComf, "k")
```

```
xlabel("Time")
ylabel("Acceleration")
title("Acceleration evolution, Np = " + Np + ", Nc = " + Nc)
hold off

%% Plot reference residual
figure(fig); fig = fig+1; hold on
plot(t, v-vRef)
xlabel("Time")
ylabel("Velocity residual")
title("Residual evolution, Np = " + Np + ", Nc = " + Nc)
hold off

%% Plot input
figure(fig); fig = fig+1; hold on
plot(t, u)
xlabel("Time")
ylabel("Input")
title("Input evolution, Np = " + Np + ", Nc = " + Nc)
hold off

%% Plot diff u
figure(fig); fig = fig+1; hold on
plot(t(2:end), diff(u))
xlabel("Time")
ylabel("$\Delta u$", Interpreter = 'latex')
title("Input time derivative evolution, Np = " + Np + ", Nc = " + Nc)
hold off
end
```

```
function [explMPC] = getexplMPCfun(vars, lambda, Np)
%% Unpack variables
alpha = vars.alpha;
beta = vars.beta;
b = vars.b;
c = vars.c;
gam = vars.gamma;
v_max = vars.v_max;
g = vars.g;
m = vars.m;
dt = vars.dt;
acomf = vars.a_comf;
v12 = vars.v12;
v23 = vars.v23;

%% Driving force
Fd = b./(1+gam.*g);

%% Friction approximation
k = beta/alpha;
q = (c*v_max^2 - beta)/(v_max - alpha);
r = beta - (c*v_max^2 - beta)/(v_max - alpha)*alpha;

%% Dynamics and systems
A1 = 1-dt*k/m; % v < alpha
A2 = 1-dt*q/m; % v >= alpha

f1 = 0;
f2 = -dt*r/m; % v >= alpha, constant

B = dt./m.*Fd; % Gears

C = 1;
D = 0;
g = 0;

sys1 = LTISystem('A', A1, 'B', B(1), 'C', C, 'D', D, 'f', f1, 'g', g, 'Ts', dt);
sys2 = LTISystem('A', A1, 'B', B(2), 'C', C, 'D', D, 'f', f1, 'g', g, 'Ts', dt);
```

```matlab
sys5 = LTISystem('A', A2, 'B', B(2), 'C', C, 'D', D, 'f', f2, 'g', g, 'Ts', dt);
sys6 = LTISystem('A', A2, 'B', B(3), 'C', C, 'D', D, 'f', f2, 'g', g, 'Ts', dt);

%% Regions
X1 = Polyhedron([1;1;-1], [alpha-eps;v12-eps;0]);
X2 = Polyhedron([1;1;-1], [alpha-eps;v23-eps;-v12]);
X5 = Polyhedron([-1;1;-1], [-alpha;v23-eps;-v12]);
X6 = Polyhedron([-1;-1;1], [-alpha;-v23;v_max]);

% plot(X1, 'color', 'r', X2, 'color', 'b', X5, 'color', 'r', X6, 'color', 'b')

%% Link dynamics to regions
sys1.setDomain('x', X1);
sys2.setDomain('x', X2);
sys5.setDomain('x', X5);
sys6.setDomain('x', X6);

%% Construct system
sys = PWASystem([sys1, sys2, sys5, sys6]);

%% Add constraints
sys.x.min = 0;
sys.x.max = v_max;
sys.u.min = vars.u_min;
sys.u.max = vars.u_max;
sys.x.with('deltaMin');
sys.x.with('deltaMax');
sys.x.deltaMin = -acomf; % Comfortability constraint
sys.x.deltaMax = acomf;

sys.x.with('reference'); % The state is a tracking problem
sys.x.reference = 'free';

%% Determine cost
sys.u.penalty = OneNormFunction( lambda );
sys.x.penalty = OneNormFunction( 1 );

%% Make Explicit controller
implMPC = MPCController(sys, Np);

explMPC = implMPC.toExplicit(); % Make explicit
end
```