Assignment

Group A16
Jiaxuan Zhang (5258162)
Yiting Li (5281873)

June 13, 2022

Course:	SC4160 Modelling and Control of Hybrid Systems
Professor:	Prof.dr.ir. Bart De Schutter
Date:	June 13, 2022

Contents

1 Part 1 | Hybrid System Example

1.1 Step 1.1 System Description

The thermal management system inside a personal computer can be considered a hybrid system. The thermal management system aims at reducing overall temperature and providing a tolerant noise by dynamically regulating the rotation speed of the electric fans and the Thermal Design Power Setting and Scenario Design Power (we will call it TDP&SDP) in the computer.

The thermal management system can work in several different modes, for example, muted mode, normal mode, and boosting mode. During each mode, it will have different TDP&SDP settings and rotation speed settings. Each group of TDP&SDP setting specifies the largest power the CPU and GPU can achieve. The setting of TDP&SDP is always not continuous, it will have several sub-modes, for example, low power setting, normal power setting, and high power setting (it is not a technical name, but from the power consumption each sub-modes allows we can use these simplified names). If we assume the largest power of a CPU is P, then we can model different TDP&SDP sub-model by $\alpha_i P$, which presents the maximum power a TDP&SDP setting can provide with α_i are parameters in $(0,1]$. If we observe the electrical fans in each mode, the rotation speed always fluctuates around different fixed values and the speed changes so rapidly when modes change that we can partly regard it as an abrupt change. Figure 1 shows an example TDP&SDP table and figure 2 shows the electric fans example of a laptop.

PSU 12V2 Capability Recommendations – 10^{th} & 11^{th} gen

Processor TDP	Continuous Current	Peak Current	Test Result
165W	37.5A	40A	
125W	26A	34A	
65W	23A	30A	
35W	13A	16.5A	

PSU 12V2 Capability Recommendations – ALD-S

Processor TDP	Continuous Current	Peak Current	Test Result
165W	37.5A	45A	
125W	26A	39A	
65W	23A	38.5A	
35W	11A	20.5A	

FCPOWERUP

Figure 1: An Example TDP&SDP Table

Figure 2: An Example Laptop Back

For example, a user can set the computer to the muted mode during a meeting to reduce the noise. In the muted mode, the rotation speed of the fans will fluctuate under 100 rpm and the TDP&SDP will be set to $0.3P$ to make sure the temperature of the CPU and GPU are acceptable. If the user then changes the mode to the normal mode, the rotation speed will rapidly change to 3500 rpm and fluctuate in the interval of $[2500, 4500]$ rpm according to the dynamic workload. In some computers, the rotation speed may also change to around 8000 rpm, if the system detects that the user is doing some heavy workload computation task under the normal mode.

A real computer may have more than 3 thermal modes and different types of computers may have different mode settings. Here we will use a simplified example to show the details of the hybrid architecture and hybrid automaton of the thermal management system of the computer.

We assume a thermal computer system with two modes: muted mode and normal mode. In the muted mode, the rotation speed will fluctuate under 300 rpm and the TDP&SDP will be set to $0.3P$. In the normal mode, the TDP&SDP will be set to $0.8P$. The rotation speed will fluctuate between $[2500, 4500]$ rpm if the current core temperature is lower than $75°C$ and will fluctuate between $[7000-9000]$ rpm if the current core temperature is higher than $75°C$.

We define state variables as $x(t) = [T(t), p(t), v(t), m(t)]$, where $T(t)$ is the CPU temperature at time t, $p(t)$ is the CPU power, $v(t)$ is the fans' rotation speed and $m(t)$ is the system mode (0 for the muted mode and 1 for the normal mode). The input of the system is $u(t) = [w(t), s(t)]$, where $w(t)$ is the user workload at time t and $s(t)$ is the user's setting of the thermal management system mode (0 for the muted mode and 1 for the normal mode). The state-space model of $m(t)$ is $m(t) = s(t)$. The function of CPU temperature, CPU power, fan's rotation speed is unknown for us (but known for the manufacturer), so we will use f to represent the functions. With different fan's rotation interval, the f will be different. In other words, system model can be presented by:

$$\dot{x}(t) = f_i(x(t), u(t)) \quad i = 1, 2, 3; \tag{1}$$

1.2 Step 1.2 Hybrid Automaton Description of the System

Based on the simplified thermal management system scenarios mentioned above, we can express the thermal management system as a hybrid automaton model as shown in Figure 3. Because it is hard to include the latex format in the figure, we use the raw latex code in the graph.

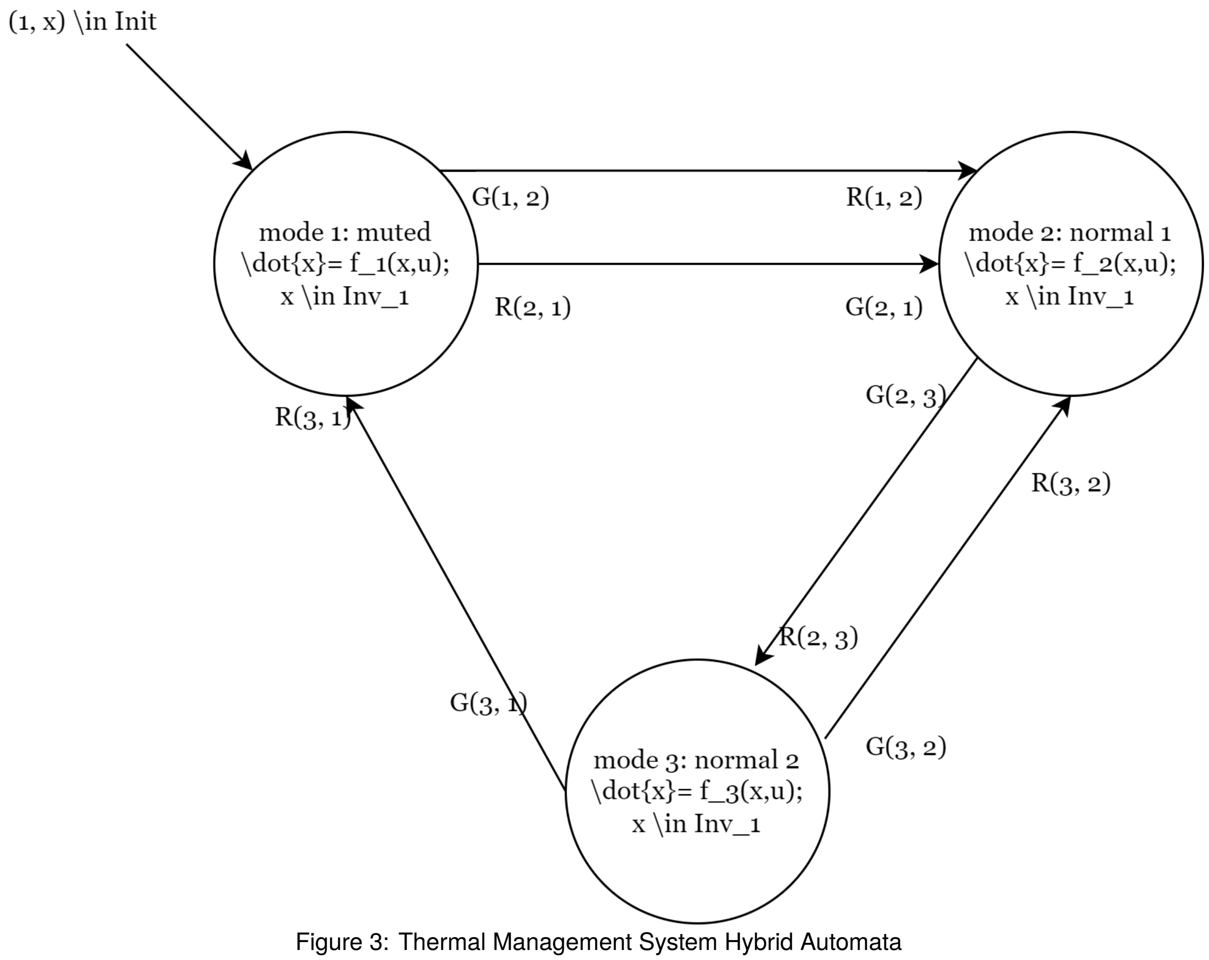

Figure 3: Thermal Management System Hybrid Automata

The automata has three modes 1,2,3 — 1 for model 1, the muted mode, 2 for mode 2, the normal mode, condition 1, 3 for mode 3 the normal mode, condition 2. The details of the initial states set, invariants. guard condition, reset map in the Figure are shown as follows.

$$
\begin{aligned}
&\text{Init} := (1, x_0) \quad \text{with } x_0 \in \left\{(T, p, v, m) | T < 50^\circ C, p < 0.3P, v = 0, m = 0\right\};\\
&\text{Inv}_1 := \left\{(T, p, v, m) | T < 70^\circ C, p < 0.3P, 0 \leq v \leq 300, m = 0\right\}\\
&\text{Inv}_2 := \left\{(T, p, v, m) | T <= 75^\circ C, p < 0.8P, 2500 \leq v \leq 4500, m = 1\right\}\\
&\text{Inv}_3 := \left\{(T, p, v, m) | T >= 75^\circ C, p < 0.8P, 7000 \leq v \leq 9000, m = 1\right\}\\
&G(1,2) := (m = 1) \quad G(2,1) := (m = 0) \quad G(3,1) := (m = 0)\\
&G(2,3) := (T \geq 75^\circ C) \quad G(3,2) := (T \leq 75^\circ C)\\
&R(1,2) := (T, p, 3000, 1) \quad R(2,1) := (T, p, 0, 0)\\
&R(2,3) := (T, p, 8000, 1) \quad R(3,2) := (T, p, 3000, 0)\\
&R(3,1) := (T, p, 0, 0)
\end{aligned} \tag{2}
$$

2 Part 2 | Adaptive Cruise Control

2.1 Step 2.1

The car will achieve the maximum speed when the the dynamic driving force is equal to the largest driving force, i.e. $F_{drive}(t) = F_{friction}(t)$. The largest driving force meets when $u(t) = u_{max}$. The analytical solution for this point is $v_{max} = \sqrt{\frac{1}{c}\frac{b}{1+\gamma g(t)}u_{max}}$. We calculated the solution for all the three possible $g(t)$ values. The result is shown in Table 1. From the table, if we assume $v < v_{12}$ ($g(t) = 1$) or $v < v_{23}$ ($g(t) = 2$), the v_{max} will violate the assumed speed limitation. The violations mean when the car start at a speed under the limitation, the car will finally exceed the limitation if it keeps the largest input u_{max}, then the gear rate assumption becomes invalid. We can specify that the max speed the car can achieve is $v_{max} = 57.7150 m/s$.

Table 1: Max Speed Calculation

	v	g	v_{max} (m/s)
1	$v < v_{12}$	1	80.1903
2	$v < v_{23}$	2	66.2472
3	$v > v_{23}$	3	57.7150

From the state-space model, the $\frac{dv(t)}{dt}$ contains two parts: $\frac{1}{m}\frac{b}{1+\gamma g(t)}u(t)$ and $-\frac{1}{m}cv^2(t)$. The first part is monotonically increase of $u(t)$ if we fix $g(t)$ and monotonically decrease of $g(t)$ if we fix $u(t)$. The second part is monotonically decrease of $v(t)$. Because the range of $u(t)$ is independent to the value of $g(t)$, the maximum value of the first part achieves when $u = u_{max} = 1.3$ and $g = g_{min} = 1$. The maximum value of the second part will achieve when $v = v_{min} = 0$. Then the maximum acceleration will be:

$$a_{acc,max} = \frac{1}{m}\frac{b}{1+\gamma\cdot 1}u_{max} - \frac{1}{m}c\cdot 0^2 = 3.2152 m/s^2 \tag{3}$$

The calculation of the maximum deceleration is more complex. Because of the piecewise affine $g(t)$, we calculated the maximum deceleration separately. The car should achieve the maximum deceleration in each region when u is u_{min} and the v reaches the maximum speed of each region — v_{12}, v_{23} and v_{max} separately. The calculation process is shown in the following equations. the maximum deceleration is $-3.3310 m/s^2$.

$$\begin{aligned}
a_{dec,max,1} &= \frac{1}{m}\frac{b}{1+g(1)\gamma}u_{min} - \frac{1}{m}cv_{12}^2 = -3.3277 m/s^2;\\
a_{dec,max,2} &= \frac{1}{m}\frac{b}{1+g(2)\gamma}u_{min} - \frac{1}{m}cv_{23}^2 = -2.6443 m/s^2;\\
a_{dec,max,3} &= \frac{1}{m}\frac{b}{1+g(3)\gamma}u_{min} - \frac{1}{m}cv_{max}^2 = -3.3310 m/s^2;
\end{aligned} \tag{4}$$

2.2 Step 2.2

The expression of piecewise affine (PWA) approximation $P(v)$ and true friction model $V(v)$ is shown as follow:

$$P(v) = \begin{cases} P1 = \frac{\beta(v-\alpha)}{\alpha}, 0 < v \leq \alpha \\ P2 = \frac{(cv_{max}^2-\beta)(v-v_{max})}{v_{max}-\alpha} + cv_{max}, \alpha < v < v_{max} \end{cases} \tag{5}$$
$$V(v) = cv^2$$

The selection of parameter α and β in the PWA approximation model $P(v)$ would base on following objective:

$$min \int_0^{v_{max}} (V(x) - P(v))^2 dv \tag{6}$$

We first design each expression in the `MAPLE` by the following commands:

```
p_1_val := subs(c = 0.4, vmax = 57.7150, p_1(x))
p_2_val := subs(c = 0.4, vmax = 57.7150, p_2(x))
V := x -> c*x^2
V_val := subs(c = 0.4, vmax = 57.7150, V(x))
err_1 := (p_1_val - V_val)^2
err_2 := (p_2_val - V_val)^2
```

The `int` function in the `MAPLE` software can help us derive an explicit expression of the error.

```
target := int(err_1, x = 0 .. alpha) + int(err_2, x = alpha .. 57.7150);
```

$$\begin{aligned} target = {} & 2.0492 \times 10^7 - 0.2000\beta\,\alpha^3 + 0.3333\beta^2\alpha - \frac{0.2000\left(1332.4085 - 1.0\beta\right)\left(1.1096 \times 10^7 - 1.0\alpha^4\right)}{57.7150 - 1.0\alpha} \\ & + 0.3333\left(\frac{46.1720\left(1332.4085 - 1.0\beta\right)}{57.7150 - 1.0\alpha} - 1065.9268 + \frac{\left(1332.4085 - 1.0\beta\right)^2}{(57.7150 - 1.0\alpha)^2}\right)\left(1.9225 \times 10^5 - 1.0\alpha^3\right) \\ & + \frac{\left(-\frac{57.7150(1332.4085-1.0\beta)}{57.7150-1.0\alpha} + 1332.4085\right)\left(1332.4085 - 1.0\beta\right)\left(3331.0212 - 1.0\alpha^2\right)}{57.7150 - 1.0\alpha} \\ & + \left(-\frac{57.7150\left(1332.4085 - 1.0\beta\right)}{57.7150 - 1.0\alpha} + 1332.4085\right)^2 (57.7150 - 1.0\alpha) \end{aligned} \tag{7}$$

Then we use the `diff` function in the `MAPLE` to generate the explicit derivative of the error at α and the β.

```
dtda := diff(target, alpha)
dtdb := diff(target, beta)
```

$$
\begin{aligned}
dtda =\ & -0.6000\beta\,\alpha^2 + 0.3333\beta^2 - \frac{0.2000\left(1332.4085 - 1.0\beta\right)\left(1.1096\times 10^7 - 1.0\alpha^4\right)}{(57.7150 - 1.0\alpha)^2} \\
& + \frac{0.8000\left(1332.4085 - 1.0\beta\right)\alpha^3}{57.7150 - 1.0\alpha} \\
& + 0.3333\left(\frac{46.1720\left(1332.4085 - 1.0\beta\right)}{(57.7150 - 1.0\alpha)^2} + \frac{2.0\left(1332.4085 - 1.0\beta\right)^2}{(57.7150 - 1.0\alpha)^3}\right)\left(1.9225\times 10^5 - 1.0\alpha^3\right) \\
& - 1.0\left(\frac{46.1720\left(1332.4085 - 1.0\beta\right)}{57.7150 - 1.0\alpha} - 1065.9268 + \frac{\left(1332.4085 - 1.0\beta\right)^2}{(57.7150 - 1.0\alpha)^2}\right)\alpha^2 \\
& - \frac{57.7150\left(1332.4085 - 1.0\beta\right)^2\left(3331.0212 - 1.0\alpha^2\right)}{(57.7150 - 1.0\alpha)^3} \\
& + \frac{1.0\left(-\frac{57.7150(1332.4085 - 1.0\beta)}{57.7150 - 1.0\alpha} + 1332.4085\right)\left(1332.4085 - 1.0\beta\right)\left(3331.0212 - 1.0\alpha^2\right)}{(57.7150 - 1.0\alpha)^2} \\
& - \frac{2.0\left(-\frac{57.7150(1332.4085 - 1.0\beta)}{57.7150 - 1.0\alpha} + 1332.4085\right)\left(1332.4085 - 1.0\beta\right)\alpha}{57.7150 - 1.0\alpha} \\
& - \frac{115.4300\left(-\frac{57.7150(1332.4085 - 1.0\beta)}{57.7150 - 1.0\alpha} + 1332.4085\right)\left(1332.4085 - 1.0\beta\right)}{57.7150 - 1.0\alpha} \\
& - 1.0\left(-\frac{57.7150\left(1332.4085 - 1.0\beta\right)}{57.7150 - 1.0\alpha} + 1332.4085\right)^2 \\
dtdb =\ & -0.2000\alpha^3 + 0.6667\beta\alpha + \frac{0.2000\left(1.1096\times 10^7 - 1.0\alpha^4\right)}{57.7150 - 1.0\alpha} \\
& + 0.3333\left(-\frac{46.1720}{57.7150 - 1.0\alpha} - \frac{2.0\left(1332.4085 - 1.0\beta\right)}{(57.7150 - 1.0\alpha)^2}\right)\left(1.9225\times 10^5 - 1.0\alpha^3\right) \\
& + \frac{57.7150\left(1332.4085 - 1.0\beta\right)\left(3331.0212 - 1.0\alpha^2\right)}{(57.7150 - 1.0\alpha)^2} \\
& - \frac{1.0\left(-\frac{57.7150(1332.4085 - 1.0\beta)}{57.7150 - 1.0\alpha} + 1332.4085\right)\left(3331.0212 - 1.0\alpha^2\right)}{57.7150 - 1.0\alpha} \\
& - \frac{6662.0424\left(1332.4085 - 1.0\beta\right)}{57.7150 - 1.0\alpha}
\end{aligned}
\tag{8}
$$

Finally, we call `solve` function to find the (α, β) combinations at which both `dtda` and `dtdb` are all zeros. `MAPLE` gave us 3 solution that located inside the v_{max} as shown in the Table 2. We choose $\alpha = 28.8575, \beta = 249.8266$ because it has the smallest approximation error.

```
solve({dtda = 0, dtdb = 0})
```

Table 2: α, β results

	α	β	error
1	28.8575	249.8266	8.0049×10^4
2	57.7055	998.9756	1.2536×10^6
3	57.7150	1332.4085	3.4154×10^6

We can get the optimal values of parameters are $\alpha = 28.8575$ and $\beta = 249.8266$, the approximation results are shown in the figure 4.

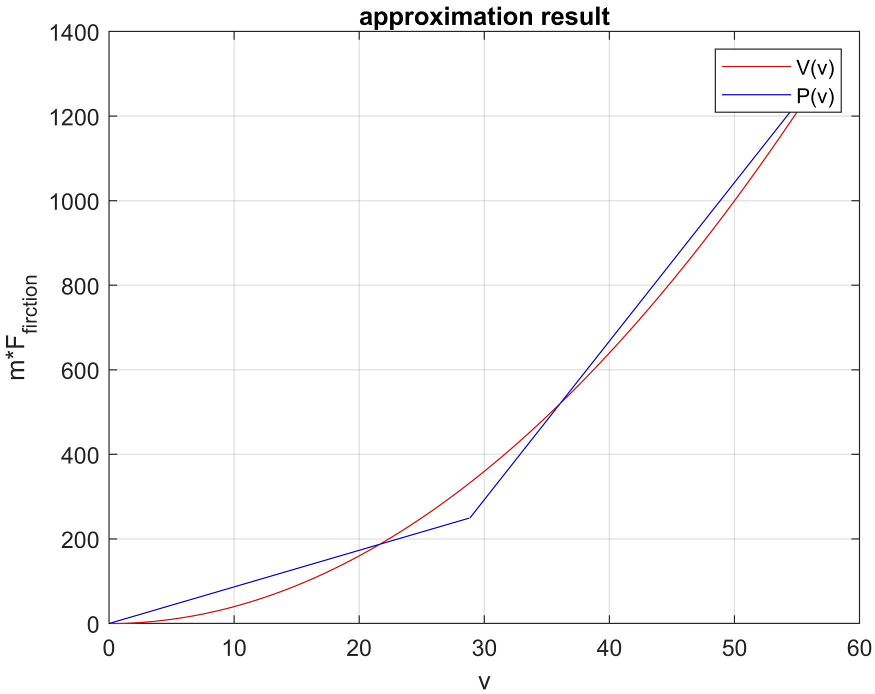

Figure 4: approximation result

2.3 Step 2.3

In this step, we assume the gear is constant, then we test different initial points — (0, 22), (0, 30), (0, 37), (0, 44), with the help of MATLAB command `ode45`, we can get the results, which are shown in the figure 9. The sinusoidal throttle input has a unit amplitude, the simulation time is 5 seconds.

The results show that when the approximation is close enough to the original system, around $v = 37m/s$ and around $v = 22m/s$, the error between the two models will be negligible. Figure 4 presents that around $v = 22m/s$ and $v = 37m/s$, there are intersections between the approximation line and the true line, which means the approximation error is small in the neighborhoods and will introduce a better simulation result.

For speed larger than $v = 37m/s$ or smaller than $v = 22m/s$, the approximation friction is larger than the true friction, while in other regions the approximation value is smaller. From the simulation, if we start at $v_0 = 44m/s$, the approximated speed is always smaller than the continuous one. If we start from $v_0 = 30m/s$ (at this speed, both regions of the PWA model are visited), the approximated speed is larger than the true speed.

2.4 Step 2.4

By the help of MAPLE, we can substitute each component and get the expression of state derivative:

$$\dot{v}(t) := f = \begin{cases} \frac{bu}{m(1+\gamma)} - \frac{\beta v}{m\alpha}, \ 0 \leq v < v_{12} \\ \frac{bu}{m(1+2\gamma)} - \frac{\beta v}{m\alpha}, \ v_{12} \leq v < \alpha \\ \frac{bu}{m(1+2\gamma)} - \frac{1}{m}\frac{(cv_{max}^2-\beta)(v-v_{max})}{v_{max}-\alpha} + cv_{max}, \ \alpha \leq v < v_{23} \\ \frac{bu}{m(1+3\gamma)} - \frac{1}{m}\frac{(cv_{max}^2-\beta)(v-v_{max})}{v_{max}-\alpha} + cv_{max}, \ v_{23} \leq v < v_{max} \end{cases} \tag{9}$$

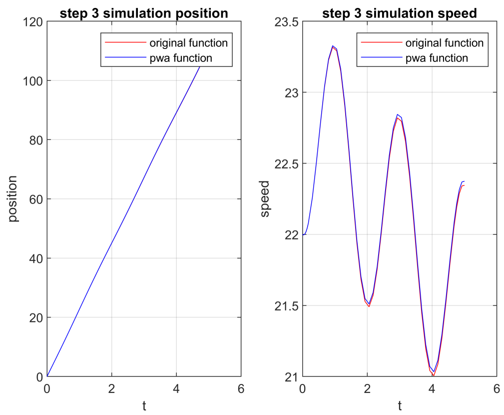

Figure 5: Evolution of the System with $v_0 = 22$

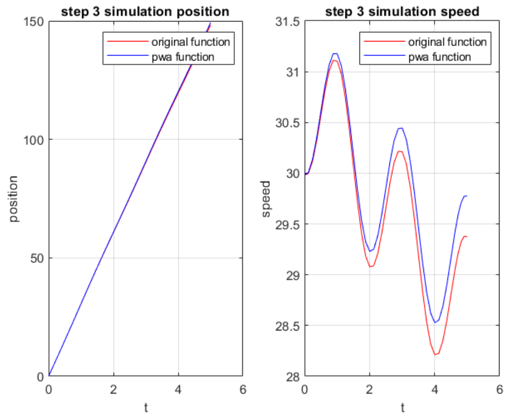

Figure 6: Evolution of the System with $v_0 = 30$

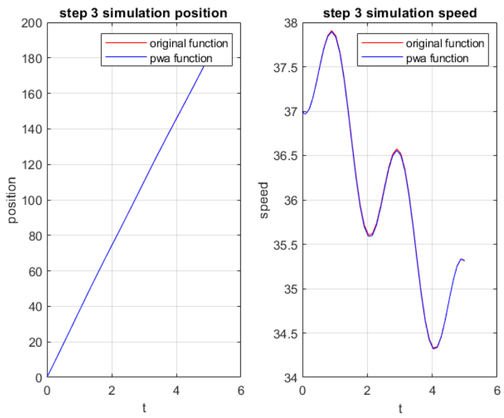

Figure 7: Evolution of the System with $v_0 = 37$

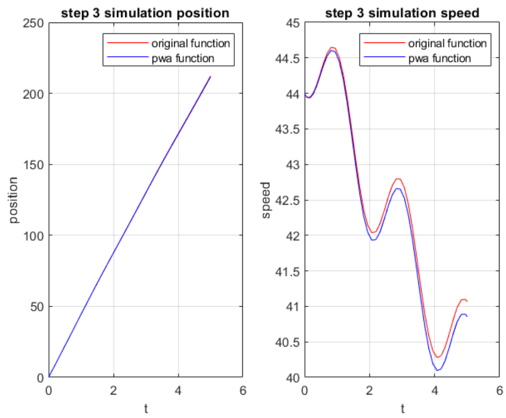

Figure 8: Evolution of the System with $v_0 = 44$

Figure 9: Evolution of the System with Constant Gear under Different Initial State

With $m = 800, c = 0.4, b = 3700, \gamma = 0.87, v_{12} = 15, v_{23} = 30, v_{max} = 57.7150, \alpha = 28.8575, \beta = 249.8266$, we can get the expression:

$$\dot{v}(t) := f = \begin{cases} 2.4733u - 0.0108v,\ 0 \le v < 15 \\ 1.6880u - 0.0108v,\ 15 \le v < 28.8575 \\ 1.6880u - 0.0469v + 1.0409,\ 28.8575 \le v < 30 \\ 1.2812u - 0.0469v + 1.0409,\ 30 \le v < 57.7150 \end{cases} \tag{10}$$

The visualization of the derivative is shown in the figure 10

2.5 Step 2.5

After we applied a forward Euler rule to the continuous time model, the discretized model is as follow:

$$v(k+1) := f = v(k) + 0.15 \times \begin{cases} 2.4733u - 0.0108v,\ 0 \le v < 15 \\ 1.6880u - 0.0108v,\ 15 \le v < 28.8575 \\ 1.6880u - 0.0469v + 1.0409,\ 28.8575 \le v < 30 \\ 1.2812u - 0.0469v + 1.0409,\ 30 \le v < 57.7150 \end{cases} \tag{11}$$

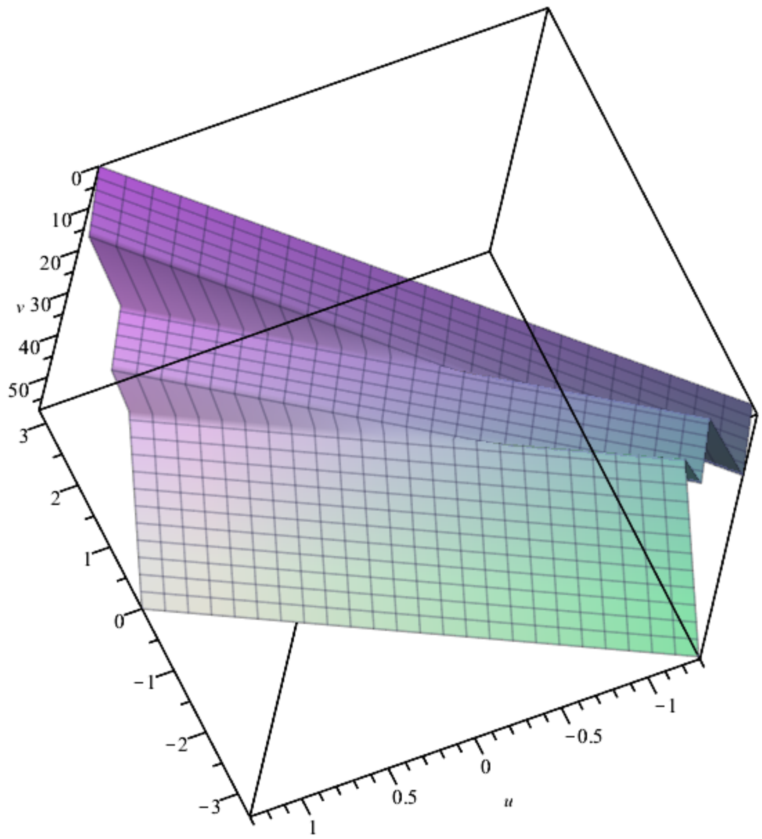

Figure 10: visualization of the derivative

2.6 Step 2.6

According to the discretized PWA model, we have four different states transition and corresponding speed condition, by adding logical variable $\delta_i \in \{0,1\}, i = 1,2,3$, we can transfer it into an MLD model as follows.

We use the three logical variables to represent the speed regions:

$$\begin{aligned} [\delta_1(k) = 1] &\Longleftrightarrow v(k) - 15 \leq 0 \\ [\delta_2(k) = 1] &\Longleftrightarrow v(k) - 28.8575 \leq 0 \\ [\delta_3(k) = 1] &\Longleftrightarrow v(k) - 30 \leq 0 \end{aligned} \tag{12}$$

We first define the boolean auxiliary variables δ_i to replace the speed condition in the states transition. The definitions are as follows, where the speed v can't be a negative number.

$$\Delta = (\delta_1, \delta_2, \delta_3) = \begin{cases} (1,1,1) \Leftarrow 0 \leq v \leq 15 \\ (0,1,1) \Leftarrow 15 < v \leq 28.8575 \\ (0,0,1) \Leftarrow 28.8575 < v \leq 30 \\ (0,0,0) \Leftarrow 30 < v \leq 57.7150 \end{cases} \tag{13}$$

Then we donate $f1, f2, f3, f4$ as four state transition under different condition:

$$
\begin{aligned}
f1 &= T \times (2.4733 \times u - 0.0108 \times v) + v \\
f2 &= T \times (1.6880 \times u - 0.0108 \times v) + v \\
f3 &= T \times (1.6880 \times u - 0.0469 \times v + 1.0409) + v \\
f4 &= T \times (1.2812 \times u - 0.0469 \times v + 1.0409) + v
\end{aligned} \tag{14}
$$

Then we can rewrite the system state transition expression combining with boolean auxiliary variables δ_i as below:

$$
v(k+1) := f = \delta_1(f1 - f2) + \delta_2(f2 - f3) + \delta_3(f3 - f4) + f4 \tag{15}
$$

The justification of rewritten expression is shown below:

$$
v(k+1) := f = \begin{cases} f1, \text{ iff } (\delta_1, \delta_2, \delta_3) = (1,1,1) \Leftarrow 0 \le v \le 15 \\ f2, \text{ iff } (\delta_1, \delta_2, \delta_3) = (0,1,1) \Leftarrow 15 < v \le 28.8575 \\ f3, \text{ iff } (\delta_1, \delta_2, \delta_3) = (0,0,1) \Leftarrow 28.8575 < v \le 30 \\ f4, \text{ iff } (\delta_1, \delta_2, \delta_3) = (0,0,0) \Leftarrow 30 < v \le 57.7150 \end{cases} \tag{16}
$$

We can introduce new variable $z1$, $z2$ and $z3$ into the 15, then we can get the state transition expression:

$$
v(k+1) := f = \frac{9609}{50000}u - \frac{31227}{200000}\delta_2 + \frac{198593}{200000}v + \frac{23559}{200000}z_1 + \frac{1083}{200000}z_2 + \frac{3051}{50000}z_3 + \frac{31227}{200000} \tag{17}
$$

With the definition of new variable as, $z_1 = \delta_1 \times u, z_2 = \delta_2 \times v, z_3 = \delta_3 \times u$.

Then we can further convert above expression into a standard MLD model:

$$
\begin{gathered}
v(k+1) := f = Av(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k) + \text{const} \\
A = \frac{198593}{200000}, B_1 = \frac{9609}{50000}, B_2 = \begin{bmatrix} 0 & -\frac{31227}{200000} & 0 \end{bmatrix}, B_3 = \begin{bmatrix} \frac{23559}{200000} & \frac{1083}{200000} & \frac{3051}{50000} \end{bmatrix}, \text{const} = \frac{31227}{200000}
\end{gathered} \tag{18}
$$

For the constraints, the first part is the δ variables constraints, with the speed boundary $M_v = 57.7150, m_v = 0$

$$
\begin{aligned}
&\delta_1 : \begin{cases} (v - v_1) \le (M_v - 15) \times (1 - \delta_1) \\ (v - v_1) \ge \epsilon + (m_v - 15 - \epsilon) \times \delta_1 \end{cases}, v_1 = 15 \\
&\delta_2 : \begin{cases} (v - v_2) \le (M_v - 28.8575) \times (1 - \delta_2) \\ (v - v_2) \ge \epsilon + (m_v - 28.8575 - \epsilon) \times \delta_2 \end{cases}, v_2 = 28.8575 \\
&\delta_3 : \begin{cases} (v - v_3) \le (M_v - 30) \times (1 - \delta_3) \\ (v - v_3) \ge \epsilon + (m_v - 30 - \epsilon) \times \delta_3 \end{cases}, v_3 = 30
\end{aligned} \tag{19}
$$

The second part of the constraints comes from the new variable $z1$, $z2$ and $z3$, with the input boundary $M_u = 1.3, m_u = -1.3$:

$$z_1: \begin{cases} z_1 \leq M_u \times \delta_1 \\ z_1 \geq m_u \times \delta_1 \\ z_1 \leq u - m_u \times (1-\delta_1) \\ z_1 \geq u - M_u \times (1-\delta_1) \end{cases}, z_2: \begin{cases} z_2 \leq M_v \times \delta_2 \\ z_2 \geq m_v \times \delta_2 \\ z_2 \leq v - m_v \times (1-\delta_2) \\ z_2 \geq v - M_v \times (1-\delta_2) \end{cases}, z_3: \begin{cases} z_3 \leq M_u \times \delta_3 \\ z_3 \geq m_u \times \delta_3 \\ z_3 \leq u - m_u \times (1-\delta_3) \\ z_3 \geq u - M_u \times (1-\delta_3) \end{cases} \tag{20}$$

We can further present constraints from δ and z to the matrix format:

$$E_1 x(k) + E_2 u(k) + E_3 \vec{\delta}(k) + E_4 \vec{z}(k) \leqslant g_5, \tag{21}$$

where

$$E_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad E_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} \quad E_3 = \begin{bmatrix} 42.7150 & 0 & 0 \\ -15.0000 & 0 & 0 \\ 0 & 28.8575 & 0 \\ 0 & -38.4767 & 0 \\ 0 & 0 & 27.7150 \\ 0 & 0 & -30.0000 \\ -1.3000 & 0 & 0 \\ -1.3000 & 0 & 0 \\ 1.3000 & 0 & 0 \\ 1.3000 & 0 & 0 \\ 0 & -57.7150 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 57.7150 & 0 \\ 0 & 0 & -1.3000 \\ 0 & 0 & -1.3000 \\ 0 & 0 & 1.3000 \\ 0 & 0 & 1.3000 \end{bmatrix} \quad E_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \tag{22}$$

and

$$\vec{\delta}(k) = [\delta_1(k), \delta_2(k), \delta_3(k)]^T \qquad \vec{z}(k) = [z_1(k), z_2(k), z_3(k)]^T \tag{23}$$

With state transition equation 18, delta variables constraints 19 and z variables constraints 20, we can express the MLD model of the given system.

2.7 Step 2.7

Based on the model in Step 2.6, we can expand the model to the future N_p steps.

For our group, in the target function, we should use $J_{\text{track}}(k) = \|\tilde{v}(k) - \tilde{v}_{\text{ref}}(k)\|_1$ and $J_{\text{input}}(k) = \|\Delta\tilde{u}(k)\|_\infty$. So we used the method in the assignment instruction to transform the optimization target to a linear programming problem. So except for the variables related to v, u, δ and z, we need to introduce variables $\rho \geq 0$ and $\tau \geq 0$.

Using τ and ρ will transform our target function to:

$$\min \sum_{i=1}^{Np} \rho_i + \lambda\tau \tag{24}$$

And at the same time, this transformation will introduce the following constraints:

$$\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix} + \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{N_p} \end{bmatrix} \geq \begin{bmatrix} x_{ref}(k+1) \\ \vdots \\ x_{ref}\left(k+N_p\right) \end{bmatrix} \quad \begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix} - \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{N_p} \end{bmatrix} \leq \begin{bmatrix} x_{ref}(k+1) \\ \vdots \\ x_{ref}\left(k+N_p\right) \end{bmatrix} \tag{25}$$

$$\begin{aligned} &\begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix} - \begin{bmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{bmatrix} \begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix} \leq \tau \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} u(k-1) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &\begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix} - \begin{bmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{bmatrix} \begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix} \geq -\tau \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} u(k-1) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned} \tag{26}$$

For the state-transition we can write the constraints as:

$$\begin{aligned} &\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix} - \begin{bmatrix} 0 & & & 0 \\ A_1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & A_1 & 0 \end{bmatrix} \begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix} - \begin{bmatrix} B_1 & & 0 \\ & \ddots & \\ 0 & & B_1 \end{bmatrix} \begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix} \\ &- \begin{bmatrix} 0 & & & 0 \\ B_2 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & B_2 & 0 \end{bmatrix} \begin{bmatrix} \vec{\delta}(k+1) \\ \vdots \\ \vec{\delta}\left(k+N_p\right) \end{bmatrix} - \begin{bmatrix} 0 & & & 0 \\ B_3 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & B_3 & 0 \end{bmatrix} \begin{bmatrix} \vec{z}(k+1) \\ \vdots \\ \vec{z}\left(k+N_p\right) \end{bmatrix} \\ &= \begin{bmatrix} Av(k) + B_2\vec{\delta}(k) + B_3\vec{z}(k) + const \\ const \\ \vdots \\ const \end{bmatrix} \end{aligned} \tag{27}$$

The inequality part of the MLD system can be written as constraints:

$$
\begin{bmatrix} E_1 & & 0 \\ & \ddots & \\ 0 & & E_1 \end{bmatrix}
\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix}
+
\begin{bmatrix} 0 & E_2 & & 0 \\ & \ddots & \ddots & \\ & & 0 & E_2 \\ 0 & & & E_2 \end{bmatrix}
\begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix}
+
\begin{bmatrix} E_3 & & 0 \\ & \ddots & \\ 0 & & E_3 \end{bmatrix}
\begin{bmatrix} \vec{\delta}(k+1) \\ \vdots \\ \vec{\delta}\left(k+N_p\right) \end{bmatrix}
+
\begin{bmatrix} E_4 & & 0 \\ & \ddots & \\ 0 & & E_4 \end{bmatrix}
\begin{bmatrix} \vec{z}(k+1) \\ \vdots \\ \vec{z}\left(k+N_p\right) \end{bmatrix}
\leq
\begin{bmatrix} \vec{g}_5 \\ \vdots \\ \vec{g}_5 \end{bmatrix}
\tag{28}
$$

The comfortable acceleration constraints can be represented by:

$$
\begin{aligned}
&\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix}
-
\begin{bmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{bmatrix}
\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix}
\leq a_{comf,max} T_s +
\begin{bmatrix} v(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
&\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix}
-
\begin{bmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{bmatrix}
\begin{bmatrix} v(k+1) \\ \vdots \\ v\left(k+N_p\right) \end{bmatrix}
\geq -a_{comf,max} T_s +
\begin{bmatrix} v(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\end{aligned}
\tag{29}
$$

Besides, we still need constraints to make $u_{N_c+1} = u_{N_c+2} = \cdots = u_{N_p-1}$. This constraints can be represented as follows:

$$
\begin{aligned}
&\begin{bmatrix}
0 & \cdots & \cdots & & & & 0 \\
\vdots & \ddots & & & & & \vdots \\
(N_C): 0 & \cdots & 0 & 0 & \cdots & & 0 \\
0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
\vdots & & & \ddots & \ddots & & \vdots \\
\vdots & & & & 0 & \ddots & 0 \\
0 & \cdots & & & & 0 & 1
\end{bmatrix}
\begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix} \\
&-
\begin{bmatrix}
0 & \cdots & \cdots & 0 & \cdots & & 0 \\
\vdots & \ddots & & \vdots & & & \vdots \\
(N_C): 0 & \cdots & 0 & 0 & \cdots & & 0 \\
0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
\vdots & & & \vdots & \vdots & & \vdots \\
0 & \cdots & 0 & 1 & 0 & \cdots & 0
\end{bmatrix}
\begin{bmatrix} u(k) \\ \vdots \\ u\left(k+N_p-1\right) \end{bmatrix}
=
\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}
\end{aligned}
\tag{30}
$$

To be concluded, in this part, we generate a linear programming problem with 7 groups of inequality constraints, 2 groups of equality constraints and the constraints $\rho \geq 0, \tau \geq 0$. Besides, x, z, u, ρ, τ are real-value variables while δ are binary variables.

We defined decision variables is defined as follows and we can then easily use the `glpk()` function to solve the problem.

$$
\vec{x} = \left[v(k+1), \cdots, v(k+N_p), u(k), \cdots u(k+N_p-1), \vec{\delta}(k+1), \cdots, \vec{\delta}(k+N_p), \vec{z}(k+1), \cdots, \vec{z}(k+N_p), \rho_1, \cdots, \rho_{Np}, \tau \right]
\tag{31}
$$

2.8 Step 2.8

Based on the model deducted in the section 2.7, we programmed our Model Predictive Control Simulator in the `Simulator_2_8.m` file. We use the `glpk()` function to solve the MILP problem.

At the end of the simulation period, the left state reference may be shorter than N_p. We then copy the last element in the state reference to make the length of the state reference equal to N_p.

2.9 Step 2.9

We run simulations with two different parameters, $(N_p, N_c) = (9, 8)$ and $(N_p, N_c) = (5, 4)$, the evolution of the system variables are shown in the figure 17.

The simulation results show that the system has a good capability to track the reference when the reference is constant, as in part c of figure 17. When the derivative of the reference speed signal is large, e.g. at $t = [3, 9]$, the derivative is infinite and at $t = [18, 21]$, the derivative is large, the controller won't force the state value to strictly track the reference signal. While the derivative of the reference speed signal is large, e.g. at $t = [9, 15]$, the state value will follow the reference signal well.

The reason for the tracking limitation at $t = [3, 9]$ and $t = [18, 21]$ could relate to the acceleration constraints and input constraints, e.g. at $t = [6, 9]$, the input signal had already reached its upper boundary, as in part e of the figure 17.

Moreover, the difference of simulation two-parameter setup $(N_p, N_c) = (9, 8)$ and $(N_p, N_c) = (5, 4)$, is not significant. One of the findings is that with larger N_p and N_c, the controller can "foresee" the speed reference, so when the state value has deviated from the reference value, the input value will be larger than the case with smaller N_p and N_c, the time instant when the input is applied is earlier as well.

The reason for the small simulation difference between the above two parameter settings could also relate to the constraints of the input signal magnitude. Since the input signal had reached its upper bound under this setting $(N_p, N_c) = (5, 4)$, increasing the control horizon and prediction horizon to $(N_p, N_c) = (9, 8)$ won't contribute much to the performance improvement.

2.10 Step 2.10

Explicit MPC is an offline computation process, which pre-compute control law as a function of state x. We need to construct the original MLD MPC problem into a parametric programming problem and feed it into the solver. The solver mostly will try to turn the problem into a pLCP problem and solve it with some algorithms. We prepare to use the `Opt()` function in the `MPT3` toolbox, so we need a parametric programming problem with the following format, where þ is the parameter and u is the controller output.

$$\begin{aligned} J(th) = \min 0.5 \times U' \times H \times u + (pF \times th + f)' \times u + t' \times Y \times th + C \times th + c \\ \text{s.t. } A \times U \le b + pB \times th \\ A_e \times U = b_e + pE \times th \\ \text{lb } \le \text{u} \le \text{ub} \end{aligned} \tag{32}$$

We tried to transform the original MLD MPC problem to a parametric programming problem. Firstly, we define current speed $[u_{k-1}; x_k; d_k; z_k; |x_{ref,k+1}, \cdots, x_{ref,k+N_p}]$ as the parameter th of the controller and define $[x_{k+1},\cdots,x_{k+Np},|u_k,\cdots,u_{k+Np-1},|\vec{d}_{k+1},\cdots,\vec{d}_{k+Np},|\vec{z}_{k+1},\cdots,\vec{z}_{k+Np},|\rho_1,...,\rho_{Np},|\tau]$ as the output u of the controller.

The original optimization target keeps unchanged, and the constraints in section 2.7 can still be used. If we want to put them into `Opt` function, the structure needs to be changed slightly because now we have different decision variables. The changes can be seen in the code shown in the appendix.

Unfortunately, we are unable to solve the above optimization problem using the MPT3 toolbox. The `Opt.solve()` function with solver `ENUMPLCP` needs more than 1 hour to generate explicit controller for $N_p = N_c = 2$. So we turn to those predefined MPC controller functions provided by the MPT3 toolbox.

We used the system model in the previous step, once the model is ready, we add the constraints using the following command.

For the boundaries of input and system states, we define the constraints as below:

```
    sys.u.min = umin;
    sys.u.max = umax;
    sys.v.min = vmin;
    sys.v.max = vmax;
```

We also need to add the acceleration constraints, we add a new attribute `deltaMin` and `deltaMax` to the system model object, then assign the acceleration constraints.

```
    sys.v.with('ΔMin');
    sys.v.with('ΔMax');
    sys.v.ΔMin = -Ts * a_comfort;
    sys.v.ΔMax = Ts * a_comfort;
```

For the objective function, we add a penalty to the difference between speed and its reference, using the keyword `free` to indicate a varying reference is applying. And the penalty function is the one-norm function of the difference.

```
    sys.v.with('reference');
    sys.v.reference = 'free';
    sys.v.penalty = OneNormFunction(1);
```

We also need to add penalty to the input. We apply infinite norm function to the difference between the current input and the previous input.

```
    sys.u.with('ΔPenalty');
    sys.u.ΔPenalty = InfNormFunction(lambda);
```

In the end, we specify the control horizon using the following command.

```
    sys.u.with( 'block' );
    sys.u.block.from = Nc;
    sys.u.block.to = Np;
```

After the constraint setup, we call the solver, and return the `MPCcontoller` abject for further simulation.

```
ctrl = MPCController(sys, Np);
explicit_ctrl = ctrl.toExplicit();
figure
explicit_ctrl.partition.plot();
```

Below command we can run the simulation, the state value and input value are stored in `data.X` and `data.U`.

```
loop = ClosedLoop(ctrl, sys);
data = loop.simulate(x0, Nsim, 'x.reference', xref, 'u.previous', u_0);
```

Or we can use the following manual simulation, whose computation time is close to the previous simulation method.

```
t = tic;
for i = 1:length(v_ref)

    u = explicit_ctrl_4_4.evaluate(x_prev, 'x.reference', v_ref(i), ...
        'u.previous', u_prev);
    [temp_t, temp_v] = ode45(@(t,y) dydt_step8(t, y, m, gamma, b, c, g), ...
        [0, Ts], [10; x_prev; u]);
    x = temp_v(end, 2);
    X = [X x];
    U = [U u];
    u_prev = u;
    x_prev = x;
end
T = toc(t);
```

Based on the above scripts, we run several experiments and got the following comparison results, as shown in the table 3.

Table 3: computation times of different approach

	$\mathrm{N}_p = N_c = 2$	$\mathrm{N}_p = N_c = 3$	$\mathrm{N}_p = N_c = 4$
implicit	0.6077 s	0.6211 s	0.6683 s
explicit	0.5759 s	1.2171 s	2.5533 s

The experiments' results show that the computation time of the implicit approach is less than the explicit approach, which is quite count-intuitive since normally solving an online optimization problem would be slower than searching an offline look-up table. Moreover, as the prediction horizon increases, the computation time of the implicit approach doesn't change a lot, while the explicit method consumes much more time.

Then we check the implicit method (`glpk` function), and we found that between every sampling time, it would take around 25 steps for `glpk` to solve the optimization, regardless of the prediction horizon. That would be the reason why the computation time of the implicit method doesn't affect by the number of the horizon.

Then we check the partition of the explicit method, and we found for the case with $N_c = N_p = 2$, the partition consists of a 3D domain with 105 polyhedra, for the case with $N_c = N_p = 2$, the number of polyhedra increases to 1748. The number of polyhedra would dramatically increase if the number of horizons raises. The look-up table method would be low-efficient if the size of the table is large, leading the time of locating one point is longer than solving an optimization problem. We also found that some literature mentioned that using a neural-network-based auto-encoder may be a good solution for this problem [1].

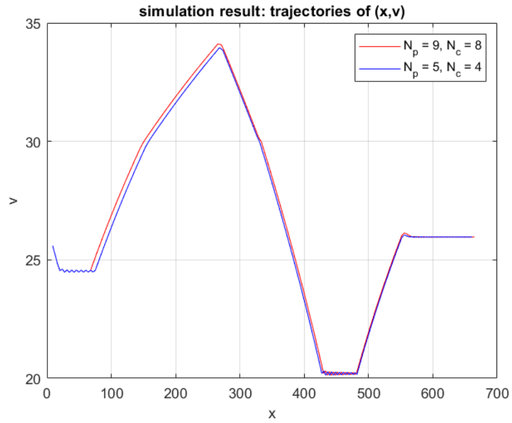

Figure 11: Evolution of (x, v) over Time

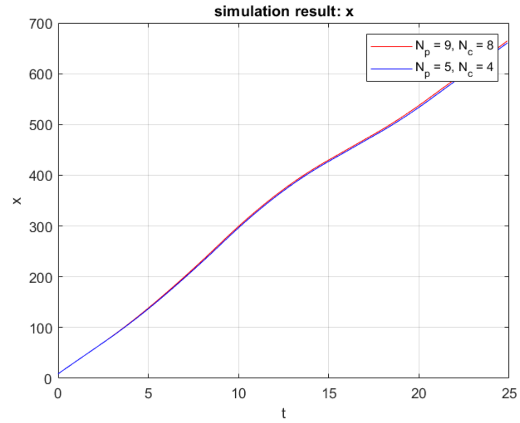

Figure 12: Evolution of x over Time

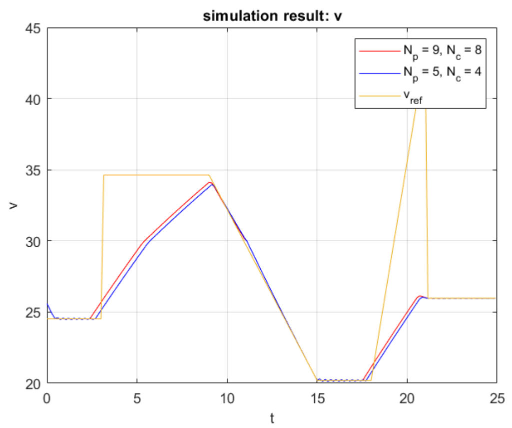

Figure 13: Evolution of v over Time

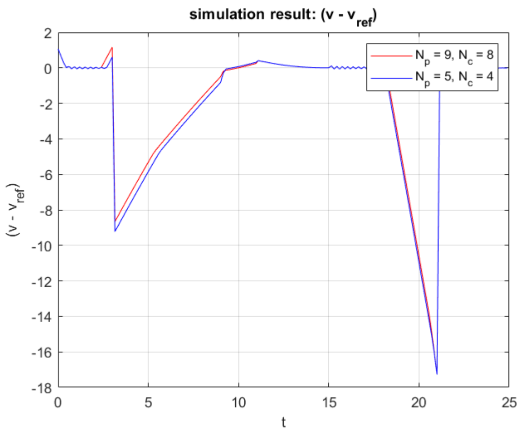

Figure 14: Evolution of $(v - v_{ref})$ over Time

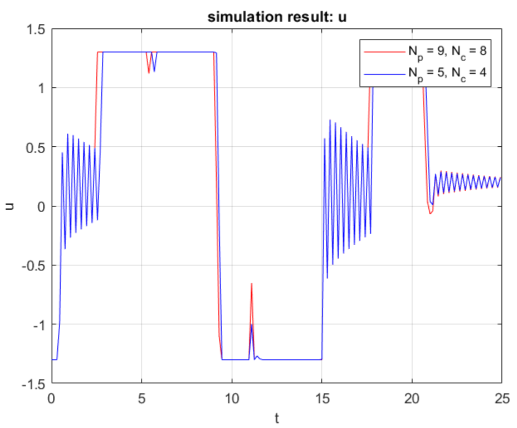

Figure 15: Evolution of u over Time

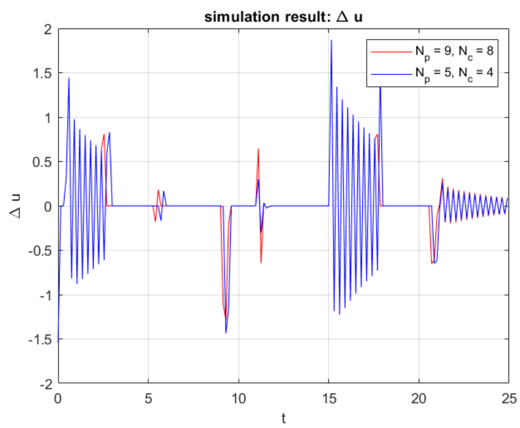

Figure 16: Evolution of Δu over Time

Figure 17: Evolution of the Controlled Closed-loop System

3 Part 3| Evaluation and Conclusion

The first insight we have obtained is the strong modeling power of the hybrid system. A lot of real-life systems can be represented by a hybrid system model. Our previous knowledge only allows us to contain a single continuous system or model them with a single discrete-time/discrete-state system but ignores the whole upper structure of the combinations of discrete part and continuous part. With the conception of the hybrid system and the tools the hybrid system research provides, we can easily model a system from the top level, research the stability and design a controller by considering both discrete part and the continuous part.

The second insight we obtained is from the Part 2 of the assignment. Part 2 shows us the whole process to analyze and design a controller for a complex real-life system.

1. The model of a real-life system may have some difficult parts (like the nonlinear part), which is too complex to deal with directly. Methods like function approximation are good methods to simplify the model.
2. After obtaining the simplified model, we can use simulation methods to test whether our simplification is acceptable and analyze the potential effect of the simplification.
3. If the simplified model is satisfying, we can then start to design a controller based on the simplified model. When designing the controller at the beginning, limitations like real-time computation requirements are not significant. We should allow our first controller to be imperfect but could work.
4. After we have a first-step available controller, we can then test requirements like real-time computation requirements to improve our controller. For improvement, general improvement patterns may not work very well in our specific scenario. Sometimes we need a specific solution, but at least we should try some general methods.

Besides, the assignment shows us how to design and implement an MPC controller in a hybrid system. During the assignment, we first tested the online MPC approach where we convert the optimization problem to a MILP problem and solved it using `glpk` solver and we noticed that the implicit MPC does has some limitations in the real-time property. To solve that, solutions like explicit MPC may work in general, because the solution of explicit MPC is a piecewise-affine function of state value and this mapping is stored offline as a look-up table. As a result, the controller doesn't have to solve the optimization problem at each step but find the specific control input based on current state values.

But the explicit MPC doesn't always outperform implicit MPC, the performance of explicit MPC might degrade when dealing with problems, where the number of partitions would grow exponentially when the horizon increases. During the experiment, we didn't get a smaller computation time after applying the explicit MPC method because of the large number of partitions, the time spent on finding the optimal value in a large loop-up table would be longer than solving an optimization problem. This also enlightens us that before applying a potential method from literature, we should always test its availability.

A Reference

[1] E. T. Maddalena, M. W. Specq, V. L. Wisniewski, and C. N. Jones, “Embedded pwm predictive control of dc-dc power converters via piecewise-affine neural networks,” *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 199–206, 2021.

B MATLAB Code

The code of this project can be found on Github Repo: SC42075-MCHS

B.1 Main File

The `main.m` file contains the main file of our program.

```
    %%%%%
%%% SC42075 Modelling and Control of Hybrid Systems
%%% Assignment
%%% Author: Jiaxuan Zhang, Yiting Li
%%%%%

clear
close all
clc

%% add path
addpath('./src')

%% Global Parameters
m = 800;
c = 0.4;
b = 3700;
umax = 1.3;
umin = -1.3;
vmin = 0;
a_comf_max = 2.5;
gamma = 0.87;
v12 = 15;
v23 = 30;

g = [1,2,3];

%% Step 2.1
% maximum speed: 57.7150 m/s;
vmax_1 = sqrt(1/c * b / (1 + g(1) * gamma) * umax);
vmax_2 = sqrt(1/c * b / (1 + g(2) * gamma) * umax);
vmax_3 = sqrt(1/c * b / (1 + g(3) * gamma) * umax);
vmax = min([vmax_1, vmax_2, vmax_3])

% maximum accelerationg 3.2152 m/s^2
a_acc_max = 1/m * b/(1 + g(1) * gamma) * umax - 0;
% maximum deacceleration
% for state 1 (-3.3277 m/s^2), however, it is a limit number and cannot ...
    actually be achieved
a_dec_max_1 =  1/m * b/(1 + g(1) * gamma) * umin - 1/m * c * v12^2;
% for state 2 (-2.6443 m/s^2), however, it is a limit number and cannot ...
    actually be achieved
a_dec_max_2 =  1/m * b/(1 + g(2) * gamma) * umin - 1/m * c * v23^2;
% for state 3 (-3.3310 m/s^2), however, it is a limit number and cannot ...
    actually be achieved
a_dec_max_3 =  1/m * b/(1 + g(3) * gamma) * umin - 1/m * c * vmax^2;
% maixmum deacceleartion is -3.3310 m/s^2
a_dec_max = min([a_dec_max_1, a_dec_max_2, a_dec_max_3]);

% clear a_dec_max_1 a_dec_max_2 a_dec_max_3
%% step 2.2
```

```
% model with two-point format
% by using maple, optimal alpha, beta are: alpha=28.8575, beta=249.8266
alpha = 28.8575;
beta = 249.8266;

Script_2_2

%% step 2.3

test_t = 5;
step_3.y0 = [0;44];

Script_2_3

%% step 2.4

% See in .mw file

% for driving force part
% f1 = @(v,u) 1/m * b/(1 + gamma) * u;
% f2 = @(v,u) 1/m * b/(1 + 2 * gamma) * u;
% f3 = @(v,u) 1/m * b/(1 + 3 * gamma) * u;
%
% % for friction part
% g1 = @(v) beta/alpha * v;
% g2 = @(v) 1/m * (c * vmax^2 - beta)/(vmax - alpha) * (v - vmax) + c * vmax^2;
%
% f11 = f1 + 0.15 * f1

%% step 2.6

model = MLD_Model_3Δ();

%% step 2.7
lambda = 0.1;
Np = 2;
Nc = 2;
x_0 = 5;
v_0 = [0];
u_0 = 0;
Ts = 0.15;
v_ref = [10; 10];

[flag, v, u, xc, uc] = Solution_2_7(Np, Nc, lambda, umax, umin, vmax, vmin, ...
    a_comf_max,...
                 v_0, u_0, model, Ts, v_ref);

%% step 2.8
T_0 = 0;
T_end = 25;
v_ref = 5 * ones(length(T_0: Ts: T_end), 1);

[v, u, Result_constant_ref] = Simulator_2_8(Np, Nc, lambda, [umin, umax], ...
    [vmin, vmax], a_comf_max,...
                 x_0, v_0, u_0, v_ref, Ts, [T_0, T_end], model, @(t,y) ...
                     dydt_step8(t, y, m, gamma, b, c, g));

%% step 2.9

Script_2_9

%% step 2.10

```

```
Script_2_10
```

B.2 Step 2.2

Step 2.2 contains one script, the `Script_2_2.m`

```
figure;

v = [0: 0.1: vmax];
v1 = [0: 0.1: alpha];
v2 = [alpha: 0.1: vmax];
plot(v, c*v.^2, 'r');
hold on
plot(v1, beta/alpha*v1, 'b')
hold on
plot(v2, ((c*vmax^2 - beta)/(vmax-alpha)*(v2 - vmax) + c*vmax^2), 'b')
grid on
legend('V(v)', 'P(v)');
xlabel('v');
ylabel('m*F_{firction}');
title("approximation result")

clear v v1 v2
```

B.3 Step 2.3

Step 2.3 contains two files, the file `dydt_step3.m` which contains original model and the PWA model, the `Script_2_3.m` is the script of step 2.3.

`dydt_step3.m`

```
    function dydt = dydt_step3(t,y,model,alpha,beta,m,gamma,b,c,vmax)
%DYDT_STEP3 the function is used to continuous-time simulation of step 3
%
% input:
%   t,y: parameter for ode function
%   model: whether use PWA (1) or Original Model (0)
%   alpha, beta: parameter in the PWA
%   m,gamma,b,c,vmax: system parameters

    gear = 1;
    dydt = zeros(2,1);

    % generate a sinusoidal throttle input
    u = sin(pi*t);

    % calculate dydt
    if model == 0
    % if we want to use the Original Model

        dydt(1) = y(2);
        dydt(2) = 1/m * b/(1 + gamma * gear) * u - 1/m * c * y(2)^2;

    elseif model ==1
    % if we want to use the PWA Model
```

```

        if y(2) ≤ alpha
        % if lies in the first part

            dydt(1) = y(2);
            dydt(2) = 1/m * b/(1 + gamma * gear) * u - 1/m * beta/alpha * y(2);

        else
        % if lies in the second part

            dydt(1) = y(2);
            dydt(2) = 1/m * b/(1 + gamma * gear) * u -...
                1/m * ((c * vmax^2 - beta)/(vmax - alpha) * (y(2) - vmax) + c ...
                    * vmax^2);

        end
    end
end
```

Script_2_3.m

```
    test_t = 5;
    step_3.y0 = [0;22];

    % original function simulation
    [temp_t, temp_y] = ode45(@(t,y) ...
        dydt_step3(t,y,0,alpha,beta,m,gamma,b,c,vmax),...
         [0,test_t],step_3.y0);
    step_3.original_simulation.t = temp_t;
    step_3.original_simulation.y = temp_y;

    % PWA function simulation
    [temp_t, temp_y] = ode45(@(t,y) ...
        dydt_step3(t,y,1,alpha,beta,m,gamma,b,c,vmax),...
         [0,test_t],step_3.y0);
    step_3.pwa_simulation.t = temp_t;
    step_3.pwa_simulation.y = temp_y;

    % plot the result
    figure
    % position result
    subplot(1, 2, 1)
    plot(step_3.original_simulation.t, step_3.original_simulation.y(:,1), 'r');
    hold on
    plot(step_3.pwa_simulation.t, step_3.pwa_simulation.y(:,1), 'b');
    grid on
    legend('original function', 'pwa function')
    xlabel('t')
    ylabel('position')
    title("step 3 simulation position")

    % speed result
    subplot(1, 2, 2)
    plot(step_3.original_simulation.t, step_3.original_simulation.y(:,2), 'r');
    hold on
    plot(step_3.pwa_simulation.t, step_3.pwa_simulation.y(:,2), 'b');
    grid on
    legend('original function', 'pwa function')
    xlabel('t')
    ylabel('speed')
    title("step 3 simulation speed")
```

B.4 Step 2.6

Step 2.6 contains one MATLAB file, the file `MLD_Model_3delta.m`.

```
function model = MLD_Model_3Δ()
    %MLD_Model_3Δ Output an MLD model with 3 Δ variables

    %% Define Variables
    %  basic variables
    syms v u

    % Discrete Time Step
    Ts = 0.15;

    % basics functions
    f1 = Ts * (2.4733 * u - 0.0108 * v) + v;
    f2 = Ts * (1.6880 * u - 0.0108 * v) + v;
    f3 = Ts * (1.6880 * u - 0.0469 * v + 1.0409) + v;
    f4 = Ts * (1.2812 * u - 0.0469 * v + 1.0409) + v;

    % some known parameters
    vmax = 57.7150;
    vmin = 0;
    umax = 1.3;
    umin = -1.3;
    v1 = 15;
    v2 = 28.8575;
    v3 = 30;

    m = vmin;
    M = vmax;

    % binary auxiliary variables
    % d1 -> v ≤ v1
    % d2 -> v ≤ v2
    % d3 -> v ≤ v3
    syms d1 d2 d3

    % auxiliary real-value variables
    syms z1 z2 z3

    % machine precision value
%     epsilon = 0.0001;
    epsilon = 0;

    % dimensions
    nv = 1;
    nu = 1;
    nz = 3;
    nd = 3;

    n = nv + nu + nz * 4 + nd * 2;

    %% Construct Target Function
    % target function
    f_original = d1 * (f1 - f2) + d2 * (f2 - f3) + d3 * (f3 - f4) + f4;
    f_original = expand(f_original);

    % f_original =  (9609*u)/50000 - (31227*d2)/200000 + (198593*v)/200000
    %                + (23559*d1*u)/200000 + (3051*d3*u)/50000 + ...
        (1083*d2*v)/200000 + 31227/200000
```

```

% after expand you can find there are only multiply:
% d1*u, d3*u, d2*v

% subs binary variables * real-value variables
% d1*u -> z1
% d2*v -> z2
% d3*u -> z3
f = subs(f_original, [d1 * u, d2 * v, d3 * u], [z1, z2, z3]);
pretty(f);

% 9609 u   31227 d2   198593 v   23559 z1   1083 z2   3051 z3    31227
% ------ - -------- + -------- + -------- + ------- + ------- + ------
%  50000    200000     200000     200000     200000    50000    200000

% Convert to standard MLD
MLD.A1 = 198593/200000;
MLD.B1 = 9609/50000;
MLD.B2 = [0, -31227/200000, 0];
MLD.B3 = [23559/200000, 1083/200000, 3051/50000];
MLD.constant = 31227/200000;

%% construct constraints

constraints = [];

% Δ variables constraints
% d1 -> v - v1 ≤ 0
g = [];
temp_g = (v - v1) ≤ (M - v1) * (1 - d1);
g = [g; temp_g];
temp_g = (v - v1) ≥ epsilon + (m - v1 - epsilon) * d1;
g = [g; temp_g];
constraints = [constraints; g];
fprintf("d1 -> v - v1 ≤ 0\n");
pretty(g);

% d2 -> v - v2 ≤ 0
g = [];
temp_g = (v - v2) ≤ (M -v2) * (1 - d2);
g = [g; temp_g];
temp_g = (v - v2) ≥ epsilon + (m - v2 - epsilon) * d2;
g = [g; temp_g];
constraints = [constraints; g];
fprintf("d2 -> v - v2 ≤ 0\n");
pretty(g);

% d3 -> v - v3 ≤ 0
g = [];
temp_g = (v - v3) ≤ (M - v3) * (1 - d3);
g = [g; temp_g];
temp_g = (v - v3) ≥ epsilon + (m -v3 - epsilon) * d3;
g = [g; temp_g];
constraints = [constraints; g];
fprintf("d3 -> v - v3 ≤ 0\n");
pretty(g);

% z variables constraints
% d1*u -> z1
g = [];
temp_g = z1 ≤ umax * d1;
g = [g; temp_g];
temp_g = z1 ≥ umin * d1;
```

```
    g = [g; temp_g];
    temp_g = z1 ≤ u - umin * (1 - d1);
    g = [g; temp_g];
    temp_g = z1 ≥ u - umax * (1 - d1);
    g = [g; temp_g];
    constraints = [constraints; g];
    fprintf("d1*u -> z1\n");
    pretty(g);

    % d2*v -> z2
    g = [];
    temp_g = z2 ≤ M * d2;
    g = [g; temp_g];
    temp_g = z2 ≥ m * d2;
    g = [g; temp_g];
    temp_g = z2 ≤ v - m * (1 - d2);
    g = [g; temp_g];
    temp_g = z2 ≥ v - M * (1 - d2);
    g = [g; temp_g];
    constraints = [constraints; g];
    fprintf("d2*v -> z2\n");
    pretty(g);

    % d3*u -> z3
    g = [];
    temp_g = z3 ≤ umax * d3;
    g = [g; temp_g];
    temp_g = z3 ≥ umin * d3;
    g = [g; temp_g];
    temp_g = z3 ≤ u - umin * (1 - d3);
    g = [g; temp_g];
    temp_g = z3 ≥ u - umax * (1 - d3);
    g = [g; temp_g];
    constraints = [constraints; g];
    fprintf("d3*u -> z3\n");
    pretty(g);

    % Other Constraints
    % speed constraint
    g = [];
    temp_g = vmin ≤ v;
    g = [g; temp_g];
    temp_g = v ≤ vmax;
    g = [g; temp_g];
    constraints = [constraints; g];
    fprintf("Other Constraints\n");
    pretty(g);

    clear g temp_g

    % comfort constraint
    % cannot be modified in a single step

    % There are total n = 20 constraint

    %% change to standard MLD constraints

    ng = nd * 2 + nz * 4;

    % E1*v + E2*u + E3*d + E4*z ≤ g5
    MLD.E1 = zeros(ng, nv);
    MLD.E2 = zeros(ng, nu);
    MLD.E3 = zeros(ng, nd);
```

```
    MLD.E4 = zeros(ng, nz);
    MLD.g5 = zeros(ng, 1);

    % define E1, v
    MLD.E1(1) = 1;
    MLD.E1(2) = -1;
    MLD.E1(3) = 1;
    MLD.E1(4) = -1;
    MLD.E1(5) = 1;
    MLD.E1(6) = -1;
    MLD.E1(13) = -1;
    MLD.E1(14) = 1;

    % define E2, u
    MLD.E2(9) = -1;
    MLD.E2(10) = 1;
    MLD.E2(17) = -1;
    MLD.E2(18) = 1;

    % define E3, [d1 d2 d3]: epsilon = 0
    % d1
    MLD.E3(1, 1) = 8543/200;
    MLD.E3(2, 1) = -15;
    % d2
    MLD.E3(3, 2) = 11543/400;
    MLD.E3(4, 2) = -11543/300;
    % d3
    MLD.E3(5, 3) = 5543/200;
    MLD.E3(6, 3) = -30;
    % d1*u -> z1
    MLD.E3(7, 1) = -13/10;
    MLD.E3(8, 1) = -13/10;
    MLD.E3(9, 1) = 13/10;
    MLD.E3(10, 1) = 13/10;
    % d2*v -> z2
    MLD.E3(11, 2) = -11543/200;
    MLD.E3(14, 2) = 11543/200;
    % d3*u -> z3
    MLD.E3(15, 3) = -13/10;
    MLD.E3(16, 3) = -13/10;
    MLD.E3(17, 3) = 13/10;
    MLD.E3(18, 3) = 13/10;

    % define E4, [z1 z2 z3]
    % d1*u -> z1
    MLD.E4(7, 1) = 1;
    MLD.E4(8, 1) = -1;
    MLD.E4(9, 1) = 1;
    MLD.E4(10, 1) = -1;
    % d2*v -> z2
    MLD.E4(11, 2) = 1;
    MLD.E4(12, 2) = -1;
    MLD.E4(13, 2) = 1;
    MLD.E4(14, 2) = -1;
    % d3*u -> z3
    MLD.E4(15, 3) = 1;
    MLD.E4(16, 3) = -1;
    MLD.E4(17, 3) = 1;
    MLD.E4(18, 3) = -1;

    % % define g5, constant: epsilon = 0
    MLD.g5(1) = 15 + 8543/200;
    MLD.g5(2) = - 15;
```

```
    % d2
    MLD.g5(3) = 11543/400 + 11543/400;
    MLD.g5(4) = - 11543/400;
    % d3
    MLD.g5(5) = 30 + 5543/200;
    MLD.g5(6) = - 30;
    % d1*u -> z1
    MLD.g5(9) = 13/10;
    MLD.g5(10) = 13/10;
    % d2*v -> z2
    MLD.g5(14) = 11543/200;
    % d3*u -> z3
    MLD.g5(17) = 13/10;
    MLD.g5(18) = 13/10;

    model = MLD;

end
```

B.5 Step 2.7

Step 2.7 contains one MATLAB file, the file `Solution_2_7.m`

```
function [flag, v, u, vc, uc] = Solution_2_7(Np, Nc, lambda, umax, umin, ...
    vmax, vmin, a_comfort, v_0, u_0, model, Ts, v_ref)
%Solution_2_7 Solution file for question 2.7
%   Detailed explanation goes here
% Input:
%   Np: prediction horizon
%   Nc: control horizon
%   lambda: relative weight of Jinput in the objective function
%   umax, umin: min and max input
%   vmax, vmin: min and max speed
%   a_comfort: comfortable acceleration limitation
%   v_0: initial v
%   u_0: previous input
%   model: MLD model
%   Ts: sampling time
%   v_ref: reference v
% Output:
%   flag: 1 for feasible optimal solution
%   v: the result of the decision variables
%   u: the optimal u for Np
%   vc: current state
%   uc: the optimal u for current time


%% define parameters
nx = 1; % dimension of x
nu = 1; % dimension of u
nd = 3; % dimension of d
nz = 3; % dimension of z
ng = size(model.g5, 1); % how many inequality constraints

v1 = 15;
v2 = 28.8575;
v3 = 30;

%% judge d_0 and z_0

```

```
if v_0 ≤ v1

    d_0 = [1; 1; 1];

elseif v_0 ≤ v2

    d_0 = [0; 1; 1];

elseif v_0 ≤v3

    d_0 = [0; 0; 1];

else

    d_0 = [0; 0; 0];

end

z_0 = d_0 .* [u_0; v_0; u_0];

%% prepare target function

c1 = zeros(1, nx*Np); % for x
c2 = zeros(1, nx*Np); % for u
c3 = zeros(1, nd*Np); % for Δ
c4 = zeros(1, nz*Np); % for z
c5 = ones(1, Np); % for rho
c6 = lambda; % for tau

c = [c1, c2, c3, c4, c5, c6];

%% prepare constraints
% decision variable format:
% [ x_{k+1}, ..., x_{k+Np}, | u_{k}, ..., u_{k+Np-1}, | d_{k+1}, ..., ...
    d_{k+Np}, |
%   z_{k+1}, ..., z_{k+Np}, | rho_{1}, ..., rho_{Np}, | tau ]

% state transition
aux_matrix = diag(ones(1,Np-1),-1);
A11 = -kron(aux_matrix, model.A1);
A11 = eye(Np) + A11;

aux_matrix = diag(ones(1, Np), 0);
A12 = -kron(aux_matrix, model.B1);

aux_matrix = diag(ones(1,Np-1),-1);
A13 = -kron(aux_matrix, model.B2);

A14 = -kron(aux_matrix, model.B3);
A15 = zeros(Np,Np);
A16 = zeros(Np,1);

A1 = [A11, A12, A13, A14, A15, A16];
b1 = ones(Np,1) * model.constant;
b1(1) = b1(1) + model.A1 * v_0 + model.B2 * d_0 + model.B3 * z_0;

% original inequalities
aux_matrix = diag(ones(1, Np), 0);
A21 = kron(aux_matrix, model.E1);

aux_matrix = diag(ones(1, Np-1), 1);
aux_matrix(end,end) = 1;
A22 = kron(aux_matrix, model.E2);
```

```

aux_matrix = diag(ones(1, Np), 0);
A23 = kron(aux_matrix, model.E3);
A24 = kron(aux_matrix, model.E4);

A25 = zeros(ng*Np,Np);
A26 = zeros(ng*Np,1);

A2 = [A21, A22, A23, A24, A25, A26];
b2 = ones(Np,1);
b2 = kron(b2, model.g5);

% optimization  construction
% for rho
A31 = eye(Np);
A32 = zeros(Np, Np);
A33 = zeros(Np, Np*nd);
A34 = zeros(Np, Np*nz);
A35 = -eye(Np, Np);
A36 = zeros(Np, 1);

A3 = [A31, A32, A33, A34, A35, A36];
b3 = v_ref;

A41 = eye(Np);
A42 = zeros(Np, Np);
A43 = zeros(Np, nd*Np);
A44 = zeros(Np, nz*Np);
A45 = eye(Np, Np);
A46 = zeros(Np, 1);

A4 = [A41, A42, A43, A44, A45, A46];
b4 = v_ref;

% for tau
A51 = zeros(Np, Np);

aux_matrix = - diag(ones(1,Np-1),-1);
A52 = eye(Np) + aux_matrix;

A53 = zeros(Np, nd*Np);
A54 = zeros(Np, nz*Np);
A55 = zeros(Np, Np);
A56 = -ones(Np, 1);

A5 = [A51, A52, A53, A54, A55, A56];
b5 = zeros(Np, 1);
b5(1) = u_0;

A61 = zeros(Np, Np);

aux_matrix = - diag(ones(1,Np-1),-1);
A62 = eye(Np) + aux_matrix;

A63 = zeros(Np, nd*Np);
A64 = zeros(Np, nz*Np);
A65 = zeros(Np, Np);
A66 = ones(Np, 1);

A6 = [A61, A62, A63, A64, A65, A66];
b6 = zeros(Np, 1);
b6(1) = u_0;

```

```
% comfortable acceleration
aux_matrix = - diag(ones(1,Np-1),-1);
A71 = eye(Np) + aux_matrix;
A72 = zeros(Np, Np);
A73 = zeros(Np, nd*Np);
A74 = zeros(Np, nz*Np);
A75 = zeros(Np, Np);
A76 = zeros(Np, 1);

A7 = [A71, A72, A73, A74, A75, A76];
b7 = zeros(Np, 1);
b7(1) =v_0;
b7 = b7 + a_comfort * Ts * ones(Np, 1);

aux_matrix = - diag(ones(1,Np-1),-1);
A81 = eye(Np) + aux_matrix;
A82 = zeros(Np, Np);
A83 = zeros(Np, nd*Np);
A84 = zeros(Np, nz*Np);
A85 = zeros(Np, Np);
A86 = zeros(Np, 1);

A8 = [A81, A82, A83, A84, A85, A86];
b8 = zeros(Np, 1);
b8(1) =v_0;
b8 = b8 - a_comfort * Ts * ones(Np, 1);

% prediction horizon vs control horizon

A91 = zeros(Np, Np);
A92 = zeros(Np, Np);
aux_matrix = zeros(Np, Np);
if (Nc < Np)
    aux_matrix([Nc+1: 1: Np], Nc) = 1;
    A92([Nc+1: 1: Np], [Nc+1: 1: Np]) = 1;
    flag = 0;
elseif (Nc > Np)
    fprintf("illegal Nc and Np");
    flag = -1;
else
    flag = 1;
end
A92 = A92 - aux_matrix;
A93 = zeros(Np, nd*Np);
A94 = zeros(Np, nz*Np);
A95 = zeros(Np, Np);
A96 = zeros(Np, 1);

A9 = [A91, A92, A93, A94, A95, A96];
b9 = zeros(Np, 1);


flag = 0 ;

% combined them together
A = [A1; A2; A3; A4; A5; A6; A7; A8; A9];
b = [b1; b2; b3; b4; b5; b6; b7; b8; b9];

% A = [A1; A3; A4; A5; A6; A7; A8; A9];
% b = [b1; b3; b4; b5; b6; b7; b8; b9];


%% prepare lb and ub
```

```

% state lb
lb1 = ones(Np, 1) * vmin;
% input lb
lb2 = ones(Np, 1) * umin;
% d lb
lb3 = 0 * ones(Np*nd, 1);
% z lb
aux_matrix = ones(Np,1);
lb4 = kron(aux_matrix, [0; 0; 0]);
% rho lb
lb5 = 0 * ones(Np ,1);
% tau lb
lb6 = 0 * ones(1);

% state ub
ub1 = ones(Np, 1) * vmax;
% input ub
ub2 = ones(Np, 1) * umax;
% d ub
ub3 = 1 * ones(Np*nd, 1);
% z ub
aux_matrix = ones(Np,1);
ub4 = kron(aux_matrix, [umax; vmax; umax]);
% rho lb
ub5 = +Inf * ones(Np, 1);
% tau lb
ub6 = +Inf * ones(1);

% combine the ub and lb
lb = [lb1; lb2; lb3; lb4; lb5; lb6];
ub = [ub1; ub2; ub3; ub4; ub5; ub6];



%% prepare solver parameter

% constraints characters
ctype =repelem(['S','U', 'U','L','U','L','U','L','S'],...
               [nx*Np, ng*Np, nx*Np, nx*Np, nu*Np, nu*Np, nx*Np, nx*Np, nu*Np]);

% ctype =repelem(['S','U','L','U','L','U','L','S'],...
%                 [nx*Np, nx*Np, nx*Np, nu*Np, nu*Np, nx*Np, nx*Np, nu*Np]);


% types of the variables
vartype = repelem(['C','C','B','C','C','C'], ...
                 [nx*Np, nu*Np, nd*Np, nz*Np, Np, 1]);


% this is a minize question
sense = 1;

% solver options
param.msglev = 3;
% param.lpsolver = 2;

%% call solver

if flag == -1

else

```

```
    [xopt, fopt, status, extra] = glpk (c, A, b, lb, ub, ctype, vartype, ...
        sense, param);

    if (status == 2)

        fprintf("feasible solution exists \n");

        fprintf("optimal objective function: d% \n", fopt);

        v = xopt;
        u = xopt([Np+1: 1: Np+Np]);
        vc = v_0;
        uc = u(1);
        flag = 1;

    elseif (status == 5)

        fprintf("optimal solution exists \n");

        fprintf("optimal objective function: d% \n", fopt);

        v = xopt;
        u = xopt([Np+1: 1: Np+Np]);
        vc = v_0;
        uc = u(1);
        flag = 1;

    else

        fprintf("no feasible solution exists \n");
        v = Inf;
        u = Inf;
        vc = Inf;
        uc = Inf;

    end

end

end
```

B.6 Step 2.8

Step 2.8 contains two MATLAB files, the file `Simulation_2_8.m` for simulate the MPC system, the file `dydt8.m` for the original system model.

`Simulation_2_8.m`

```
function Result = Simulator_2_8(Np, Nc, lambda, u_range, v_range, a_comfort, ...
    x_0, v_0, u_0, v_ref, Ts, Tspand, model_mld, modelc)
%Simulator_2_8:  Simulator to simulate the closed-loop behavior of the system
% Input:
%   Np: prediction horizon
%   Nc: control horizon
%   lambda: relative weight of Jinput in the objective function
%   urange: [umin, umax] min and max input
%   v_range: [vmin, vmax] min and max speed
%   a_comfort: comfortable acceleration limitation
%   v_0: initial v
```

```
%   model: MLD model
%   Ts: sampling time
%   v_ref: reference v
%   Tspand: [T_0, T_end]: starting time and stop time
%   model_mld: MLD model
%   modelc: continuous time model
% Output:
%  Result: a structure records the result of the simulation
%       Result.u_diff
%       Result.u_history
%       Result.v_diff
%       Result.v_history
%       Result.x_history
%       Result.v_ref

%% some parameters

v1 = 15;
v2 = 28.8575;
v3 = 30;

% limitation of u and limitation of v
umin = u_range(1);
umax = u_range(2);
vmin = v_range(1);
vmax = v_range(2);

% starting time and stop time
T_0 = Tspand(1);
T_end = Tspand(2);

%% judge d_0 and z_0

if v_0 ≤ v1

    d_0 = [1; 1; 1];

elseif v_0 ≤ v2

    d_0 = [0; 1; 1];

elseif v_0 ≤v3

    d_0 = [0; 0; 1];

else

    d_0 = [0; 0; 0];

end

z_0 = d_0 .* [u_0; v_0; u_0];

%% start simulation

x_history = zeros(1, length(T_0: Ts: T_end));
v_history = zeros(1, length(T_0: Ts: T_end));
u_history = zeros(1, length(T_0: Ts: T_end));

i = 1;
x_history(i) = x_0;
v_history(i) = v_0;
u_history(i) = u_0;
```

```
for t = T_0: Ts: T_end

    if (t + Np * Ts > T_end)
    % if Np future > T_end,
    % then extend reference with the last element in x_ref

        temp_v_ref = v_ref([i: 1:  end]);
        temp_v_ref = [temp_v_ref; v_ref(end) * ones(Np - length(temp_v_ref), 1)];

    else

        temp_v_ref = v_ref([i: 1: i + Np - 1]);

    end


    [flag, v, u, vc, uc] = Solution_2_7(Np, Nc, lambda, umax, umin, vmax, ...
        vmin, a_comfort,...
                v_0, u_0, model_mld, Ts, temp_v_ref);

    if flag == 1

        % update state with continuous model
        [temp_t, temp_v] = ode45(modelc, [0, Ts], [10; vc; uc]);

    else

        fprintf("no feasible optimal solution at time d% \n", t);
        break;

    end

    % update state and input
    v_0 = temp_v(end, 2);
    x_0 = x_0 + v_0 * Ts;
    u_0 = u(1);

    % store state and input
    x_history(i) = x_0;
    v_history(i) = v_0;
    u_history(i) = u_0;
    i = i + 1;


end

%% store the result

if flag == 1
% if feasible simulation
    v_diff = v_history' - v_ref;
    temp = circshift(u_history, 1);
    u_diff = u_history - temp;

    Result.u_diff = u_diff;
    Result.u_history = u_history;
    Result.v_diff = v_diff;
    Result.v_history = v_history;
    Result.x_history = x_history;
    Result.v_ref = v_ref;
end

```

```
end
```

dydt8.m

```
function dydt = dydt_step8(t, y, m, gamma, b, c, g)
%DYDT_STEP8 the function is used to continuous-time simulation of step 8
%
% input:
%    t,y: parameter for ode function
%    m,gamma,b,c: system parameters
%    g: possible gear settings


%% some parameters

v1 = 15;
v2 = 28.8575;
v3 = 30;


%% dydt generate

if y < v1

    gear = g(1);

elseif y<v3

    gear = g(2);

else

    gear = g(3);

end


dydt = zeros(3,1);

dydt(1) = y(2);
dydt(2) = 1/m * b/(1 + gamma * gear) * y(3) - 1/m * c * y(2)^2;
dydt(3) = 0;  % represent u

end
```

B.7 Step 2.9

Step 2.9 contains two MATLAB file, the file `GenerateXRef_2_9.m` for generating the reference signal, the file `Script_2_9.m` for some scripts use in the 2.9.

GenerateXRef_2_9.m

```
function v_ref = GenerateXRef_2_9(Ts, alpha)
%GenerateXRef_2_8 Generate the Corresponding x_ref Signal
% Input:
%    Ts: sampling time
%    alpha: parameter of PWA model
% Output:
```

```
%   v_ref: reference v

% constant
T_0 = 0; T_end = 25;
T1 = 3; T2 = 9; T3 = 15; T4 = 18; T5 = 21;

v_ref = 5 * ones(length(T_0: Ts: T_end), 1);

% 0 ≤ t ≤ 3
temp = 0.85 * alpha * ones(length(T_0: Ts: T1), 1);
v_ref = temp;

% 3 < t ≤ 9
temp = 1.2 * alpha * ones((length(T1: Ts: T2)-1), 1);
v_ref = [v_ref; temp];

% 9 < t ≤ 15
t = T2 + Ts;
for i = 1 : (length(T2: Ts: T3)-1)
    temp = 1.2 * alpha - 1/12 * alpha * (t - 9);
    t = t + Ts;
    v_ref = [v_ref; temp];
end

% 15 < t ≤ 18
temp = 0.7 * alpha * ones((length(T3: Ts: T4)-1), 1);
v_ref = [v_ref; temp];

% 18 < t ≤ 21
t = T4 + Ts;
for i = 1 : (length(T4: Ts: T5)-1)
    temp = 0.7 * alpha + 4/15 * alpha * (t - 18);
    t = t + Ts;
    v_ref = [v_ref; temp];
end

% 21 < t ≤ 25
temp = 0.9 * alpha * ones((length(T5: Ts: T_end)-1), 1);
v_ref = [v_ref; temp];

end
```

Script_2_9.m

```
lambda = 0.1;
Np = 5;
Nc = 4;
x_0 = 5;
v_0 = 0.9*alpha;
u_0 = 0;
Ts = 0.15;
T_0 = 0;
T_end = 25;
v_ref = GenerateXRef_2_9(Ts, alpha);

[v, u, Results_varying_ref_5_4] = Simulator_2_8(Np, Nc, lambda, [umin, umax], ...
    [vmin, vmax], a_comf_max,...
                    x_0, v_0, u_0, v_ref, Ts, [T_0, T_end], model, @(t,y) ...
                        dydt_step8(t, y, m, gamma, b, c, g));

lambda = 0.1;
Np = 9;
```

```
Nc = 8;
x_0 = 5;
v_0 = 0.9*alpha;
u_0 = 0;
Ts = 0.15;
T_0 = 0;
T_end = 25;
v_ref = GenerateXRef_2_9(Ts, alpha);

[v, u, Results_varying_ref_9_8] = Simulator_2_8(Np, Nc, lambda, [umin, umax], ...
    [vmin, vmax], a_comf_max,...
                  x_0, v_0, u_0, v_ref, Ts, [T_0, T_end], model, @(t,y) ...
                      dydt_step8(t, y, m, gamma, b, c, g));


figure
plot(Results_varying_ref_9_8.x_history, Results_varying_ref_9_8.v_history, 'r');
hold on;
plot(Results_varying_ref_5_4.x_history, Results_varying_ref_5_4.v_history, 'b');
grid on;
legend('N_p = 9, N_c = 8', 'N_p = 5, N_c = 4');
xlabel('x');
ylabel('v');
title("simulation result: trajectories of (x,v)")

figure
plot([T_0: Ts: T_end], Results_varying_ref_9_8.x_history, 'r');
hold on;
plot([T_0: Ts: T_end], Results_varying_ref_5_4.x_history, 'b');
grid on;
legend('N_p = 9, N_c = 8', 'N_p = 5, N_c = 4');
xlabel('t');
ylabel('x');
title("simulation result: x")

figure
plot([T_0: Ts: T_end], Results_varying_ref_9_8.v_history, 'r');
hold on;
plot([T_0: Ts: T_end], Results_varying_ref_5_4.v_history, 'b');
hold on;
plot([T_0: Ts: T_end], v_ref);
grid on;
legend('N_p = 9, N_c = 8', 'N_p = 5, N_c = 4', 'v_{ref}');
xlabel('t');
ylabel('v');
title("simulation result: v")


figure
plot([T_0: Ts: T_end], Results_varying_ref_9_8.v_diff, 'r');
hold on;
plot([T_0: Ts: T_end], Results_varying_ref_5_4.v_diff, 'b');
grid on;
legend('N_p = 9, N_c = 8', 'N_p = 5, N_c = 4');
xlabel('t');
ylabel('(v - v_{ref})');
title("simulation result: (v - v_{ref})")

figure
plot([T_0: Ts: T_end], Results_varying_ref_9_8.u_history, 'r');
hold on;
plot([T_0: Ts: T_end], Results_varying_ref_5_4.u_history, 'b');
grid on;
```

```
legend('N_p = 9, N_c = 8', 'N_p = 5, N_c = 4');
xlabel('t');
ylabel('u');
title("simulation result: u")
figure
plot([T_0: Ts: T_end], Results_varying_ref_9_8.u_diff, 'r');
hold on
plot([T_0: Ts: T_end], Results_varying_ref_5_4.u_diff, 'b');
grid on;
legend('N_p = 9, N_c = 8', 'N_p = 5, N_c = 4');
xlabel('t');
ylabel('\Delta u');
title("simulation result: \Delta u")
```

B.8 Step 2.10

Step 2.10 contains two files: `Solution_2_10.m` for generating controller, `Script_2_10.m` for some script used in this step.

`Solution_2_10.m`

```

function [ctrl, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, vmin, ...
    a_comfort, model, Ts, mode)
%Solution_2_10 Solution file for question 2.7
%   Detailed explanation goes here
% Input:
%   Np: prediction horizon
%   Nc: control horizon
%   lambda: relative weight of Jinput in the objective function
%   umax, umin: min and max input
%   vmax, vmin: min and max speed
%   a_comfort: comfortable acceleration limitation
%   model: MLD model
%   Ts: sampling time
%   mode: 1 for implicit by MPT3, 0 for explicit, 2 for explicit manually(failed)
% Output:
%   ctrl: output controller
%

%% define parameters
nv = 1; % dimension of x
nu = 1; % dimension of u
nd = 3; % dimension of d
nz = 3; % dimension of z
ng = size(model.g5, 1); % how many inequality constraints

v1 = 15;
v2 = 28.8575;
v3 = 30;


%% variables classification

if mode == 2
    % explicit manually

    %% define target function
    % output variable (u) format:
```

```
% [ v_{k+1}, ..., v_{k+Np}, | u_{k}, ..., u_{k+Np-1}, | d_{k+1}, ..., ...
    d_{k+Np}, |
%   z_{k+1}, ..., z_{k+Np}, | rho_{1}, ..., rho_{Np}, | tau ]

f1 = zeros(1, nv*Np); % for x
f2 = zeros(1, nv*Np); % for u
f3 = zeros(1, nd*Np); % for Δ
f4 = zeros(1, nz*Np); % for z
f5 = ones(1, Np); % for rho
f6 = lambda; % for tau

f = [f1, f2, f3, f4, f5, f6]';


%% define equality constraints
% input variable (th) format:
% [u_{k-1}; v_{k}; d_{k}; z_{k}; v_ref(k), ..., c_ref{k+Np}]

% state transition
aux_matrix = diag(ones(1,Np-1),-1);
Ae11 = -kron(aux_matrix, model.A1);
Ae11 = eye(Np) + Ae11;

aux_matrix = diag(ones(1, Np), 0);
Ae12 = -kron(aux_matrix, model.B1);

aux_matrix = diag(ones(1,Np-1),-1);
Ae13 = -kron(aux_matrix, model.B2);

Ae14 = -kron(aux_matrix, model.B3);
Ae15 = zeros(Np,Np);
Ae16 = zeros(Np,1);

Ae1 = [Ae11, Ae12, Ae13, Ae14, Ae15, Ae16];
be1 = ones(Np,1) * model.constant;
pE1 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
pE1(1,[nu+1: 1: nu+nv+nd+nz]) = [model.A1, model.B2, model.B3];

% prediction horizon vs control horizon
Ae21 = zeros(Np, Np);
Ae22 = zeros(Np, Np);
aux_matrix = zeros(Np, Np);
if (Nc < Np)
    aux_matrix([Nc+1: 1: Np], Nc) = 1;
    Ae22([Nc+1: 1: Np], [Nc+1: 1: Np]) = eye(Np-Nc);
    flag = 0;
elseif (Nc > Np)
    fprintf("illegal Nc and Np");
    flag = -1;
else
    flag = 1;
end
Ae22 = Ae22 - aux_matrix;
Ae23 = zeros(Np, nd*Np);
Ae24 = zeros(Np, nz*Np);
Ae25 = zeros(Np, Np);
Ae26 = zeros(Np, 1);

Ae2 = [Ae21, Ae22, Ae23, Ae24, Ae25, Ae26];
be2 = zeros(Np, 1);
pE2 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
pE2(all(Ae2==0, 2),:) = [];
be2(all(Ae2==0, 2),:) = [];
```

```
    Ae2(all(Ae2==0, 2),:) = [];

    Ae = [Ae1; Ae2];
    be = [be1; be2];
    pE = [pE1; pE2];


    %% define inequality constraints
    % input variable (th) format:
    % [u_{k-1}; v_{k}; d_{k}; z_{k}; v_ref(k), ..., v_ref{k+Np}]

    % original inequalities
    aux_matrix = diag(ones(1, Np), 0);
    A11 = kron(aux_matrix, model.E1);

    aux_matrix = diag(ones(1, Np-1), 1);
    aux_matrix(end,end) = 1;
    A12 = kron(aux_matrix, model.E2);

    aux_matrix = diag(ones(1, Np), 0);
    A13 = kron(aux_matrix, model.E3);
    A14 = kron(aux_matrix, model.E4);

    A15 = zeros(ng*Np,Np);
    A16 = zeros(ng*Np,1);

    A1 = [A11, A12, A13, A14, A15, A16];
    b1 = ones(Np,1);
    b1 = kron(b1, model.g5);
    pB1 = ones(Np*ng, nu + nv + nd + nz + nv * Np) * 0;

    % optimization  construction
    % for rho
    A21 = eye(Np);
    A22 = zeros(Np, Np);
    A23 = zeros(Np, Np*nd);
    A24 = zeros(Np, Np*nz);
    A25 = -eye(Np, Np);
    A26 = zeros(Np, 1);

    A2 = [A21, A22, A23, A24, A25, A26];
    b2 = ones(Np, 1) * 0;
    pB2 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
    pB2([1: 1: Np], [nu + nv + nd + nz + 1: 1: nu + nv + nd + nz + nv * Np]) ...
        = eye(Np);

    A31 = eye(Np);
    A32 = zeros(Np, Np);
    A33 = zeros(Np, nd*Np);
    A34 = zeros(Np, nz*Np);
    A35 = eye(Np, Np);
    A36 = zeros(Np, 1);

    A3 = [A31, A32, A33, A34, A35, A36];
    b3 = ones(Np, 1) * 0;
    pB3 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
    pB3([1: 1: Np], [nu + nv + nd + nz + 1: 1: nu + nv + nd + nz + nv * Np]) ...
        = eye(Np);

    % for tau
    A41 = zeros(Np, Np);

    aux_matrix = - diag(ones(1,Np-1),-1);
```

```
    A42 = eye(Np) + aux_matrix;

    A43 = zeros(Np, nd*Np);
    A44 = zeros(Np, nz*Np);
    A45 = zeros(Np, Np);
    A46 = -ones(Np, 1);

    A4 = [A41, A42, A43, A44, A45, A46];
    b4 = zeros(Np, 1);
    pB4 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
    pB4(1,[1: 1: nu]) = 1;

    A51 = zeros(Np, Np);

    aux_matrix = - diag(ones(1,Np-1),-1);
    A52 = eye(Np) + aux_matrix;

    A53 = zeros(Np, nd*Np);
    A54 = zeros(Np, nz*Np);
    A55 = zeros(Np, Np);
    A56 = ones(Np, 1);

    A5 = [A51, A52, A53, A54, A55, A56];
    b5 = zeros(Np, 1);
    pB5 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
    pB5(1,[1: 1: nu]) = 1;

    % comfortable acceleration
    aux_matrix = - diag(ones(1,Np-1),-1);
    A61 = eye(Np) + aux_matrix;
    A62 = zeros(Np, Np);
    A63 = zeros(Np, nd*Np);
    A64 = zeros(Np, nz*Np);
    A65 = zeros(Np, Np);
    A66 = zeros(Np, 1);

    A6 = [A61, A62, A63, A64, A65, A66];
    b6 = zeros(Np, 1);
    pB6 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
    pB6(1, [nu+1: 1: nu+nv]) = 1;

    aux_matrix = - diag(ones(1,Np-1),-1);
    A71 = eye(Np) + aux_matrix;
    A72 = zeros(Np, Np);
    A73 = zeros(Np, nd*Np);
    A74 = zeros(Np, nz*Np);
    A75 = zeros(Np, Np);
    A76 = zeros(Np, 1);

    A7 = [A71, A72, A73, A74, A75, A76];
    b7 = zeros(Np, 1);
    pB7 = ones(Np, nu + nv + nd + nz + nv * Np) * 0;
    pB7(1, [nu+1: 1: nu+nv]) = 1;

    A = [A1; A2; -A3; A4; -A5; A6; -A7];
    b = [b1; b2; -b3; b4; -b5; b6; -b7];
    pB = [pB1 ;pB2; -pB3; pB4; -pB5; pB6; -pB7];

    %% prepare lb and ub
    % output variable (u) format:
    % [ v_{k+1}, ..., v_{k+Np}, | u_{k}, ..., u_{k+Np-1}, | d_{k+1}, ..., ...
        d_{k+Np}, |
    %   z_{k+1}, ..., z_{k+Np}, | rho_{1}, ..., rho_{Np}, | tau ]
```

```

    % state lb
    lb1 = ones(Np, 1) * vmin;
    % input lb
    lb2 = ones(Np, 1) * umin;
    % d lb
    lb3 = 0 * ones(Np*nd, 1);
    % z lb
    aux_matrix = ones(Np,1);
    lb4 = kron(aux_matrix, [0; 0; 0]);
    % rho lb
    lb5 = 0 * ones(Np ,1);
    % tau lb
    lb6 = 0 * ones(1);

    % state ub
    ub1 = ones(Np, 1) * vmax;
    % input ub
    ub2 = ones(Np, 1) * umax;
    % d ub
    ub3 = 1 * ones(Np*nd, 1);
    % z ub
    aux_matrix = ones(Np,1);
    ub4 = kron(aux_matrix, [umax; vmax; umax]);
    % rho lb
    ub5 = 60 * ones(Np, 1);
    % tau lb
    ub6 = 2.6 * ones(1);

    % combine the ub and lb
    lb = [lb1; lb2; lb3; lb4; lb5; lb6];
    ub = [ub1; ub2; ub3; ub4; ub5; ub6];


    %% variable type definition
    vartype = repelem(['C','C','B','C','C','C'], ...
                      [nv*Np, nu*Np, nd*Np, nz*Np, Np, 1]);

    %% call solver

    opt = Opt('f', f, 'A', A, 'b', b, 'pB', pB, 'Ae', Ae, 'be', be, 'pE', pE, ...
            'lb', lb, 'ub', ub, 'vartype', vartype);

    opt.display()
    % opt.minHRep();
    opt.display()
    opt.qp2lcp();
    opt.solver = 'ENUMPLCP';
    ctrl = opt.solve();
    sys = [];


else


    %% define system model

    % % regard [x, d, z] as the whole state variables
    sys = Model_generator(vmin, v1, v2, v3, vmax, Ts);

    %% define MPC model
    % define penalty: the target function

```

```
    sys.u.min = umin;
    sys.u.max = umax;
    sys.x.min = vmin;
    sys.x.max = vmax;
    % sys.y.min = vmin;
    % sys.y.max = vmax;

    sys.u.with('ΔPenalty');
    sys.u.ΔPenalty = InfNormFunction(lambda);

    sys.x.with('reference');
    sys.x.reference = 'free';
    sys.x.penalty = OneNormFunction(1);

    sys.x.with('ΔMin');
    sys.x.with('ΔMax');
    sys.x.ΔMin = -Ts * a_comfort;
    sys.x.ΔMax = Ts * a_comfort;

    % define Np and Nc horizon
    sys.u.with( 'block' );
    sys.u.block.from = Nc;
    sys.u.block.to = Np;

    %% call solver
    ctrl = MPCController(sys, Np);

    if mode == 1
        ctrl;
    elseif mode == 0
        explicit_ctrl = ctrl.toExplicit();
        figure
        explicit_ctrl.partition.plot();
    end

end

end
```

Script_2_10.m

```

x0 = 0.9*alpha;
v_ref = GenerateXRef_2_9(Ts, alpha)';
Nsim = length(v_ref);
temp = [];
for i = 1:Nsim
    temp = [temp, repmat(v_ref(i), 1, 1)];
end
xref = temp;

clear temp

%% Failed Attemps: Manually Explicit Controller: Generation Too Slow

% lambda = 0.1;
% Np = 2;
% Nc = 2;
% x_0 = 5;
% v_0 = [0];
% u_0 = 0;
% Ts = 0.15;
```

```
%
% [explicit_ctrl, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, ...
    vmin, a_comf_max,...
%                          model, Ts, 2);


%% Generate Controllers

Np = 2;
Nc = 2;

[ctrl_2_2, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, vmin, ...
    a_comf_max,...
                 model, Ts, 1);
[explicit_ctrl_2_2, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, ...
    vmin, a_comf_max,...
                 model, Ts, 0);

Np = 3;
Nc = 3;
[ctrl_3_3, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, vmin, ...
    a_comf_max,...
                 model, Ts, 1);
[explicit_ctrl_3_3, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, ...
    vmin, a_comf_max,...
                 model, Ts, 0);

Np = 4;
Nc = 4;
[ctrl_4_4, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, vmin, ...
    a_comf_max,...
                 model, Ts, 1);
[explicit_ctrl_4_4, sys] = Solution_2_10(Np, Nc, lambda, umax, umin, vmax, ...
    vmin, a_comf_max,...
                 model, Ts, 0);

%% simulation: implicit controller
t1 = tic;
% loop_im_2_2 = ClosedLoop(ctrl_2_2, sys);
% data_im_2_2 = loop_im_2_2.simulate(x0, Nsim, 'x.reference', xref, ...
    'u.previous', u_0);
v_ref = GenerateXRef_2_9(Ts, alpha);
[v, u, Results_2_2] = Simulator_2_8(2, 2, lambda, [umin, umax], [vmin, vmax], ...
    a_comf_max,...
                 x_0, v_0, u_0, v_ref, Ts, [T_0, T_end], model, @(t,y) ...
                     dydt_step8(t, y, m, gamma, b, c, g));

T1 = toc(t1);
t2 = tic ;
% loop_im_3_3 = ClosedLoop(ctrl_3_3, sys);
% data_im_3_3 = loop_im_3_3.simulate(x0, Nsim, 'x.reference', xref, ...
    'u.previous', u_0);
[v, u, Results_3_3] = Simulator_2_8(3, 3, lambda, [umin, umax], [vmin, vmax], ...
    a_comf_max,...
                 x_0, v_0, u_0, v_ref, Ts, [T_0, T_end], model, @(t,y) ...
                     dydt_step8(t, y, m, gamma, b, c, g));

T2 = toc(t2);
t3 = tic;
% loop_im_4_4 = ClosedLoop(ctrl_4_4, sys);
% data_im_4_4 = loop_im_4_4.simulate(x0, Nsim, 'x.reference', xref, ...
    'u.previous', u_0);
[v, u, Results_4_4] = Simulator_2_8(4, 4, lambda, [umin, umax], [vmin, vmax], ...
```

```
    a_comf_max,...
                x_0, v_0, u_0, v_ref, Ts, [T_0, T_end], model, @(t,y) ...
                    dydt_step8(t, y, m, gamma, b, c, g));

T3 = toc(t3);

%% simulation: explicit controller

% x0 = 0.9*alpha;
% u0 = 0;
% x_prev = x0;
% u_prev = u0;
% X = [];
% U = [];
% t4 = tic;
% for i = 1:length(v_ref)
%
%     u = explicit_ctrl_2_2.evaluate(x_prev, 'x.reference', v_ref(i), ...
    'u.previous', u_prev);
%     [temp_t, temp_v] = ode45(@(t,y) dydt_step8(t, y, m, gamma, b, c, g), ...
    [0, Ts], [10; x_prev; u]);
%     x = temp_v(end, 2);
%     X = [X x];
%     U = [U u];
%     u_prev = u;
%     x_prev = x;
% end
% T4 = toc(t4);
%
% x0 = 0.9*alpha;
% u0 = 0;
% x_prev = x0;
% u_prev = u0;
% X = [];
% U = [];
% t5 = tic;
% for i = 1:length(v_ref)
%
%     u = explicit_ctrl_3_3.evaluate(x_prev, 'x.reference', v_ref(i), ...
    'u.previous', u_prev);
%     [temp_t, temp_v] = ode45(@(t,y) dydt_step8(t, y, m, gamma, b, c, g), ...
    [0, Ts], [10; x_prev; u]);
%     x = temp_v(end, 2);
%     X = [X x];
%     U = [U u];
%     u_prev = u;
%     x_prev = x;
% end
% T5 = toc(t5);
%
% x0 = 0.9*alpha;
% u0 = 0;
% x_prev = x0;
% u_prev = u0;
% X = [];
% U = [];
% t6 = tic;
% for i = 1:length(v_ref)
%
%     u = explicit_ctrl_4_4.evaluate(x_prev, 'x.reference', v_ref(i), ...
    'u.previous', u_prev);
%     [temp_t, temp_v] = ode45(@(t,y) dydt_step8(t, y, m, gamma, b, c, g), ...
    [0, Ts], [10; x_prev; u]);
```

```
%      x = temp_v(end, 2);
%      X = [X x];
%      U = [U u];
%      u_prev = u;
%      x_prev = x;
% end
% T6 = toc(t6);

t4 = tic;
loop_explicit_2_2 = ClosedLoop(explicit_ctrl_2_2, sys);
data_explicit_2_2 = loop_explicit_2_2.simulate(x0, Nsim, 'x.reference', xref, ...
    'u.previous', u_0);
T4 = toc(t4);
t5 = tic;
loop_explicit_3_3 = ClosedLoop(explicit_ctrl_3_3, sys);
data_explicit_3_3 = loop_explicit_3_3.simulate(x0, Nsim, 'x.reference', xref, ...
    'u.previous', u_0);
T5 = toc(t5);
t6 = tic;
loop_explicit_4_4 = ClosedLoop(explicit_ctrl_4_4, sys);
data_explicit_4_4 = loop_explicit_4_4.simulate(x0, Nsim, 'x.reference', xref, ...
    'u.previous', u_0);
T6 = toc(t6);

clear t1 t2 t3 t4 t5 t6
```