Institut für
Automatisierungstechnik

Master Thesis

Swing up of the Double Inverted Pendulum using Model Predictive Control

Miaoshun Wu

January 14, 2020

Tutor:
Dr.-Ing. Jochen Schuettler

1st Examiner:
Prof. Dr.-Ing. Kai Mueller

2nd Examiner:
Dr.-Ing. Jochen Schuettler

Universität Bremen

Declaration of honor

I hereby confirm on my honor that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisor has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

Date: _____ Signature: _____

Declaration of publication

- ☐ I hereby agree, that my thesis will be available for third party review in purpose of academic research.
- ☐ I hereby agree, that my thesis will be available after 30 years (§7 Abs. 2 BremArchivG) in the university archive for third party review in purpose of academic research.
- ☐ I hereby do *not* agree, that my thesis will be available for third party review in purpose of academic research.

Date: _____ Signature: _____

Acknowledgement

First of all, I would like to thank my tutor Dr.-Ing. Jochen Schuettler who gave me the golden opportunity to do this wonderful project on the topic Model Predictive Control, who also helped me in doing a lot of researches and visualization about this project. I came to know about so many new things about MPC control and classic study in dynamic models. I am really thankful to him.

Then I would like to especially show my respect to my professor Prof. Dr.-Ing. Kai Mueller. During my master, he is always there whenever I had a question about my sessions. He really helps me a lot during my study.

Finally I have to give my thanks to my parents who financially support me a lot during my master as well as my girlfriend Yixiao.With her accompany I can finally finish my master thesis. These accomplishments would not be possible without them. Thank you!

Abstract

The aim of this thesis is to swing up a double inverted pendulum by using model predictive control (MPC). The optimal algorithm WORHP(European NLP solver) is a mathematical software library for solving continuous large scale nonlinear optimization problems numerically[5].

Swinging up two pendulums from downward position to up position by using model predictive control should build the dynamic model of the double inverted pendulum and set nonlinear model predictive control with objective function.

Finally, set up the double inverted pendulum for TransWORHP in Visiual Studio. Additionally, we have optimal the output value and correct the predicted value by using model prediction error.

Contents

1. Introduction

1.1. Background

1.1.1. Inverted Pendulum

The inverted pendulum model is a very common model of a control system that applies external forces to keep it in a dynamically balanced state of natural balance. The inverted pendulum can be divided into single, double, and multi-level inverted pendulums by how many swinging pendulum it has. The control problem of the inverted pendulum is to make the pendulum reaches a balance position as soon as possible, and keep in a stable position.

The inverted pendulum is a typical multi-variable, nonlinear, natural unstable system. The stability control of the inverted pendulum system is a typical problem in control theory. It can effectively reflect many key problems in control theory, such as nonlinear problem, tracking problems, stability problems, etc.

As a typical physical model in control theory, we often use an inverted pendulum system to test the correctness of new control theory algorithms and find out its effectiveness in practical application.

1.1.2. Model Predictive Control

Model Predictive Control (MPC) is a modern control theory developed in the beginning of 1960s. It has been successfully applied and developed rapidly in oil, electric power and aerospace industries. Therefore, the appearing of predictive control is not the product of some theoretical research, but an effective control method developed in the course of industrial practice.

Model Predictive Control (MPC) is a special control method. It can obtain the optimal solution in a specific time domain at each sampling time. The current state of the process is the initial state of the optimal control problem, and the optimal control sequence only performs the first control action.

The biggest difference between this and an algorithm is that MPC uses a precomputed control law. In essence, model predictive control solves the problem of open-loop optimal control. The idea is not about the concrete model, but the implementation of the model.

At the same time, model predictive control is widely used in military industry, aerospace, robot technology and general industrial process. For example, balancing control in robot walking, vertical control in rocket launching, etc.

1.2. Aim

We managed to design a controller for swinging up the double inverted pendulum (which is also unstable, nonlinear, and we neglect friction) by coupling a highly efficient solver (TransWORHP).
Dynamic model of double inverted pendulum shall be presented as well as MPC control design will be explained. The TransWORHP solver shall be carried out by running program in Visual Studio. The output of the double inverted pendulum shall be visualized in Matlab.
In chapter 2, all the details of the dynamic model will be explained. In chapter 3 and chapter 4, the control design will be introduced and double inverted pendulum will be set up for TransWORHP. In chapter 5, the visualization of output after running program in Visual Studio will be shown, the result of optimization by TransWORHP will be discussed. In chapter 6, the conclusion and future work.

2. Dynamic Model

2.1. Double inverted pendulum(DIP)

2.1.1. Dynamic system of DIP

The dynamic system of the double inverted pendulum is described bases on the analysis presented in[8].

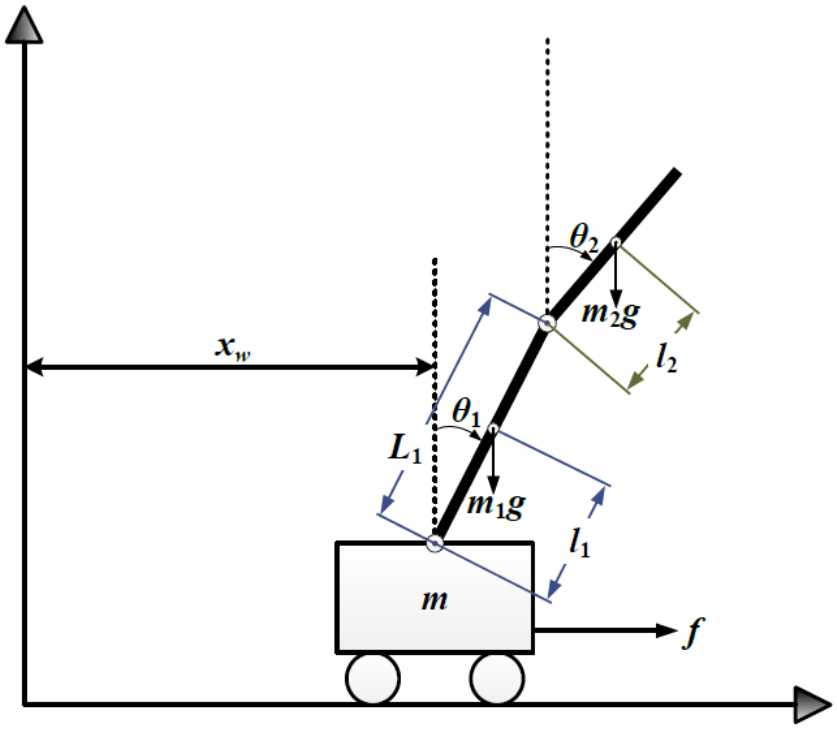

Figure 2.1.: The double inverted pendulum model

Where:
x_w : Cart displacement in x_w axis
f : Force in x_w axis
m : Mass of the cart
L_1 : Length of pendulum 1
l_1 : Distance between endpoint and center of mass of pendulum 1
$m_1 g$: Product of the mass of pendulum 1 and gravity acceleration
θ_1 : Clockwise angle displacement of pendulum 1 (Top position = 0)
l_2 : Distance between endpoint and center of mass of pendulum 2
$m_2 g$: Product of the mass of pendulum 2 and gravity acceleration
θ_2 : Clockwise angle displacement of pendulum 2 (Top position = 0)
This system consists of two linked pendulums mounted on a cart that can move linearly with force f in the axis, in order to swing up both pendulums (to keep the θ_1 and θ_2 to be 0). The masses of the cart and pendulums are assumed to be homogeneously distributed and concentrated in their center of gravity. Additionally, we do not consider their frictions.

The mathematical model of the DIP system can obtain by using the Euler-Lagrange equation. From the first approach, it is necessary to describe the kinematic and potential energy of the system components. Only after that the Euler-Lagrange equation can be formed.

$$L = T - V \tag{2.1}$$

here,

T : Kinetic energy
V : Potential energy

See here the form of the Euler-Lagrange equation used:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{q}}\right] - \frac{\partial L}{\partial q} = Q_q \tag{2.2}$$

where:
Q_q : Generalized forces not taken into account in T, V
$q = [x_w, \theta_1, \theta_2]^T$ generalized coordinates
The system kinetic energy T is given by:

$$T = T_{cart} + T_{pendulum1} + T_{pendulum2} = T_1 + T_2 + T_3 \tag{2.3}$$

where:
$T_1 = \frac{1}{2} m \dot{x}_w^2$
$T_2 = \frac{1}{2} m_1 \left[\left(\dot{x}_w + l_1 \dot{\theta}_1 cos\theta_1 \right)^2 + \left(l_1 \dot{\theta}_1 sin\theta_1 \right)^2 \right] + \frac{1}{2} J_1 \dot{\theta}_1^2$
$T_3 = \frac{1}{2} m_2 \left[\left(\dot{x}_w + L_1 \dot{\theta}_1 cos\theta_1 + l_2 \dot{\theta}_2 cos\theta_2 \right)^2 + \left(L_1 \dot{\theta}_1 sin\theta_1 + l_2 \dot{\theta}_2 sin\theta_2 \right)^2 \right] + \frac{1}{2} J_2 \dot{\theta}_2^2$

J_1, J_2 are inertia force of the first and second pendulum, L_1 is the length of the first pendulum, here $L1 = 2 * 1_1$. So the potential energy V is:

$$\begin{aligned} V &= V_{cart} + V_{pendulum1} + V_{pendulum2} = V_1 + V_2 + V_2 \\ &= 0 + m_1 g l_1 cos\theta_1 + m_2 g (L_1 cos\theta_1 + l_2 cos\theta_2) \end{aligned} \tag{2.4}$$

The Euler-Lagrangian equations for the DIP system result in:

$$\begin{aligned} h_1 \ddot{x} + h_2 \ddot{\theta}_1 \cos\theta_1 + h_3 \ddot{\theta}_2 \cos\theta_2 &= h_2 \dot{\theta}_1^2 \sin\theta_1 + h_3 \dot{\theta}_2^2 \sin\theta_2 + f \\ h_2 \ddot{x} \cos\theta_1 + h_4 \ddot{\theta}_1 + h_5 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) &= h_7 \sin\theta_1 - h_5 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) \\ h_3 \ddot{x} \cos\theta_2 + h_6 \ddot{\theta}_2 + h_5 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) &= h_8 \sin\theta_2 + h_5 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) \end{aligned} \tag{2.5}$$

The above equations were simplified by using following parameters:

$$\begin{aligned} h_1 &= m + m_1 + m_2 & h_5 &= m_2 l_2 L_1 \\ h_2 &= m_1 l_1 + m_2 L_1 & h_6 &= m_2 l_2^2 + J_2 \\ h_3 &= m_2 l_2 & h_7 &= m_1 l_1 g + m_2 L_1 g \\ h_4 &= m_1 l_1^2 + m_2 L_1^2 + J_1 & h_8 &= m_2 l_2 g \end{aligned} \tag{2.6}$$

The Euler-Lagrange equations can be put into a frequently used compact form [4]:

$$M(q)\ddot{q} + C(q, \dot{q}) + g(q) = Q_q \tag{2.7}$$

where:

$$M(q) = \begin{bmatrix} h_1 & h_2\cos\theta_1 & h_3\cos\theta_2 \\ h_2\cos\theta_1 & h_4 & h_5\cos(\theta_1-\theta_2) \\ h_3\cos\theta_2 & h_5\cos(\theta_1-\theta_2) & h_6 \end{bmatrix}, q = \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

$$C(c,\dot{q}) = \begin{bmatrix} 0 & -h_2\dot{\theta}_1\sin(\theta_1) & -h_3\sin\theta_2\dot{\theta}_2 \\ 0 & 0 & h_5\sin(\theta_1-\theta_2)\dot{\theta}_2 \\ 0 & -h_5\sin(\theta_1-\theta_2)\dot{\theta}_1 & 0 \end{bmatrix}$$

$$g(q) = \begin{bmatrix} 0 \\ -h_7\sin\theta_1 \\ -h_8\sin\theta_2 \end{bmatrix}, Q_q = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix} \tag{2.8}$$

Since the control inputs of DIP systems are less than degrees of freedom, we have to think about general under-actuated mechanical systems with n generalized coordinates $q_l, .., q_i$ and $j < i$ actuators. By dividing the vector q, we obtain $q^T = (q_1^T, q_2^T)$, where q_1 corresponds to the passive variables and q_2 to the driving variables. then the Euler-Lagrange equations of the n-degree-of-freedom Mechanical System Dynamics with q_1 passive coordinates and q_2 activation coordinate can be expressed in the following general form [4]:

$$\begin{aligned} M_{11}\ddot{q}_1 + M_{12}\ddot{q}_2 + C_1(q,\dot{q}) + g_1(q) &= 0 \\ M_{21}\ddot{q}_1 + M_{22}\ddot{q}_2 + C_2(q,\dot{q}) + g_2(q) &= f \end{aligned} \tag{2.9}$$

where:

$$\begin{aligned} M(q) &= \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \qquad q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \\ C(q,\dot{q}) &= \begin{bmatrix} C_1(q,\dot{q}) \\ C_2(q,\dot{q}) \end{bmatrix}, \qquad g(q) = \begin{bmatrix} g_1(q) \\ g_2(q) \end{bmatrix} \end{aligned} \tag{2.10}$$

It follows from 2.8 with 2.10 that:

$$
\begin{aligned}
M_{11} &= \begin{bmatrix} h_4 & h_5\cos(\theta_1-\theta_2) \\ h_5\cos(\theta_1-\theta_2) & h_6 \end{bmatrix}, \\
M_{22} &= \begin{bmatrix} h_1 \end{bmatrix}, \qquad M_{12} = M_{21}^T = \begin{bmatrix} h_2\cos\theta_1 \\ h_3\cos\theta_2 \end{bmatrix}, \\
q_1 &= \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}, q_2 = \begin{bmatrix} x_w \end{bmatrix} \\
C_1 &= \begin{bmatrix} h_5\sin(\theta_1-\theta_2)\dot{\theta}_1 \\ -h_5\sin(\theta_1-\theta_2)\dot{\theta}_1 \end{bmatrix}, C_2 = \begin{bmatrix} -h_2\dot{\theta}_1\sin(\theta_1) - h_3\sin\theta_2\dot{\theta}_2 \end{bmatrix} \\
g_1 &= \begin{bmatrix} -h_7\sin\theta_1 \\ -h_8\sin\theta_2 \end{bmatrix}, \qquad g_2 = \begin{bmatrix} 0 \end{bmatrix}
\end{aligned}
\tag{2.11}
$$

It can be derived from (2.8) that $M(q)$ is a symmetric and positive definite inertia matrix, we can find the proof in [8], as well as that the vector C includes Coriolis terms and centrifugal terms, while g includes terms derived from potential energy, such as gravity and the generalized elastic force. The vector f represents the input of the generalized force generated by the m actuators at q_2. To simplify the notation, we will no longer write explicit dependencies in some equations.
Consider the first formula in (2.9), and replace $\ddot{q}_2$ by the acceleration of the cart $\ddot{x}_w$:

$$M_{11}\ddot{q}_1 + M_{12}\ddot{x}_w + C_1(q,\dot{q}) + g_1(q) = 0 \tag{2.12}$$

So we can get the $\ddot{q}_1$:

$$\ddot{q}_1 = -M_{11}^{-1}(M_{12}\ddot{x}_w + C_1 + g_1) \tag{2.13}$$

through [3] the inverted pendulum we can find out

$$\ddot{x}_w = -\frac{1}{\tau}\cdot\dot{x}_w + \frac{1}{\tau}\cdot u_q \tag{2.14}$$

Where $u_q \in R^m$ is the control input that will be designed in control part. Finally, all variables are rearranged in order to obtain a dynamic model of the

double inverted pendulum in the form $\dot{Q} = a(Q) + B(Q) \cdot u_q$[2].

$$\dot{Q} = \begin{bmatrix} -M_{11}^{-1}(M_{12}\frac{-Q_6}{\tau} + C_1 + g_1) \\ Q_1 \\ Q_2 \\ Q_6 \\ -\frac{Q_6}{\tau} \end{bmatrix} + \begin{bmatrix} -M_{11}^{-1}(\frac{M_{12}}{\tau}) \\ 0 \\ 0 \\ 0 \\ \frac{1}{\tau} \end{bmatrix} \cdot u_q \tag{2.15}$$

Where the state space vector Q is defined as:

$$Q = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \theta_1 \\ \theta_2 \\ x_w \\ \dot{x}_w \end{bmatrix} \tag{2.16}$$

3. Control Design

3.1. Model Predictive Control

Model Predictive Control (MPC) algorithm is an advanced control algorithm, which can be divided into linear and nonlinear. According to the dynamic model, we use nonlinear model predictive control.
In recent years, nonlinear model predictive control (NMPC) has become the hotspot of predictive control research. The research object is disturbed and restricted by nonlinear process control. The difficulties of nonlinear model predictive control are mainly embodied in model selection, energy function solution and nonlinear algorithm. The nonlinear model is the basis of nonlinear MPC and the basic factor to distinguish nonlinear MPC from linear MPC.

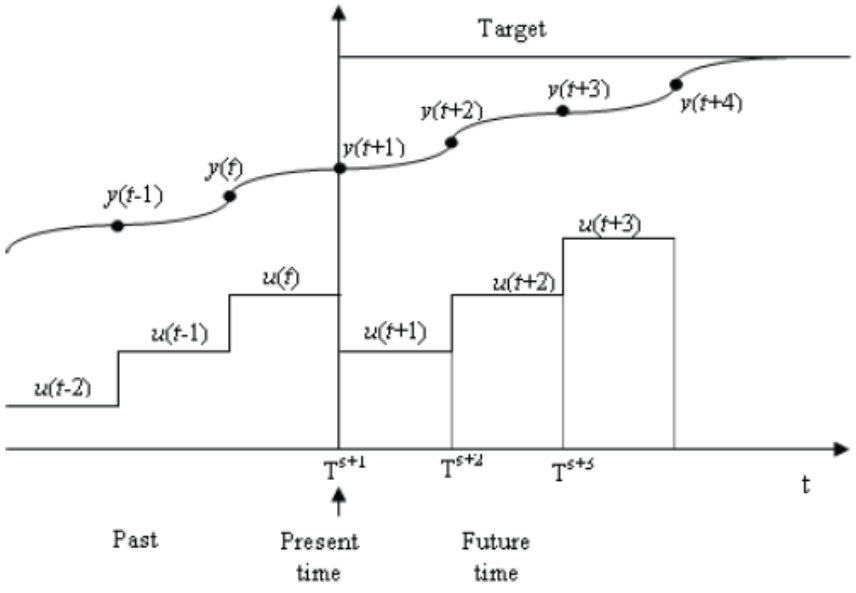

Figure 3.1.: MPC multi-step prediction scheme[1]

Although there are many algorithms for model predictive control, their prin-

ciples are generally the same. The algorithm block diagram is shown in figure 3.2:

Model predictive control uses four basic components:

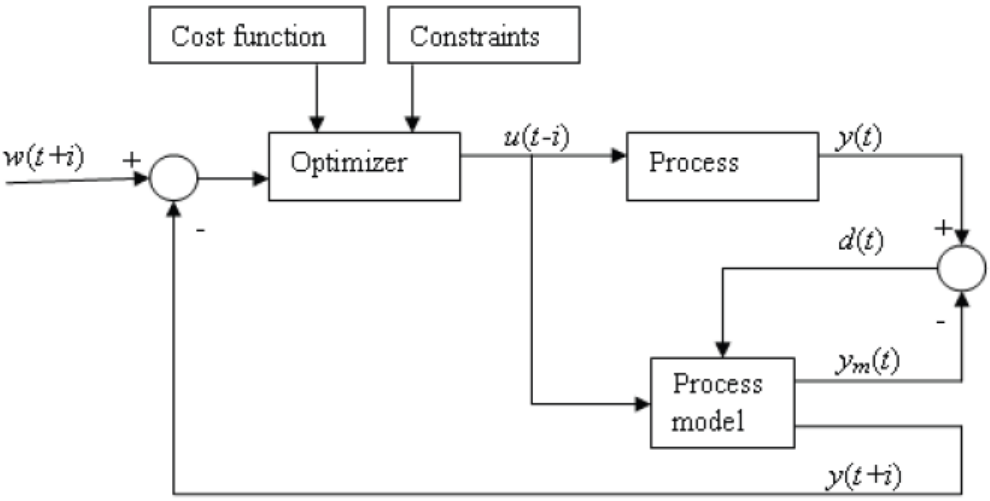

Figure 3.2.: MPC block diagram[1]

Predictive Model: Predictive control should have a predictive function. It can predict the future value of the process output according to the current control input of the system and the historical information of the process. Therefore, a model describing the dynamic behavior of the system is needed as the predictive model.

Reference Track: In predictive control,in order to avoid sharp changes in input and output, the process output $y(t)$ is often required to reach a set-point value $w(t+i)$ along a prospective, gentle curve. This curve is often referred as the reference trajectory y.

Rolling Optimization: Predictive control is an optimal control algorithm that needs to be optimized by a certain performance indicator called objective (function) which determines future control effects. This performance metric also relates to the future behavior of the process, which is determined by future control strategies based on the predictive model.

Feedback Correction: In predictive control, the use of predictive models for the estimation of process output values is only an ideal way in actual process. Model-based predictions are unlikely to be accurately consistent with reality due to uncertainties such as non-linearity, model mismatch, and interference.

Therefore, in the predictive control, the measured value $y(t)$ of the output is compared with the estimated value $y_m(t)$ of the model to obtain the prediction error $d(t)$ of the model. And the predicted value $y(t+i)$ of the model is corrected by using the model prediction error.

Nonlinear Model Predictive Control(NMPC)

However, when the object is highly non-linear, it must be predicted and optimized based on a non-linear model. Otherwise, the predicted output and the actual deviation from the linear model are too large to achieve the purpose of optimal control.
We can set the nonlinear model in [7]:

$$\left.\begin{array}{c} x(t+1) = f(x(t), u(t)) \\ y(t) = g(x(t)) \end{array}\right\} \tag{3.1}$$

where: $x \in \mathbb{R}^m, u \in \mathbb{R}^m, y \in \mathbb{R}^p$. So according to this model above, as long as the initial state $x(t)$ of the object and its future control inputs $u(t), u(t+1), ...$ are known at time t, the model output $\hat{y}_m$ of the object at each moment in the future can be predicted.

$$\left.\begin{array}{ll} \hat{x}(t+i \mid t) & = f(\hat{x}(t+i-1 \mid t), u(t+i-1)) \\ \hat{y}(t+i \mid t) & = g(\hat{x}(t+i \mid t)) = g(f(\hat{x}(t+i-1 \mid t), u(t+i-1))) \\ i = 1, 2... & \end{array}\right\} \tag{3.2}$$

By recursion, you can get i ($t+i$ is multiples of t) step predictions [1]:

$$\hat{y}_m(t+i \mid t) = F_i\left[x(t), u(t), \ldots, u(t+i-1)\right] \quad i = 1, 2, \ldots \tag{3.3}$$

$$\hat{y}_m(t+1 \mid t) = f\left[x(t), \ldots, x(t+1-n_y), u(t), \ldots, u(t+1-n_u)\right] \quad i = 1, 2, \ldots \tag{3.4}$$

Among them, $F_i()$ is a nonlinear function consisting of $f()$ and $g()$. The equation (3.4) is the predictive model for nonlinear system.
Because the actual controlled system always contains some uncertain factors, the above model prediction can not describe the dynamic behavior of the object completely. Therefore, the prediction model can be corrected by error prediction and compensation based on the measured output. If the actual output measured at time t is $y(t)$, then:

$$e(t) = y(t) - \hat{y}_m(t) \tag{3.5}$$

Correcting model-based predictions and form closed-loop predictions of outputs:

$$\hat{y}_p(t+i) = \hat{y}_m(t+i) + e(t+i) \tag{3.6}$$

When we get the $\hat{y}_p(t)$, the optimal control input sequence in NMPC is computed by minimizing an objective function based on a desired output trajectory over a prediction horizon N and control horizon M :

$$\min J = \sum_{i=1}^{N} \lambda(w(t+i) - \hat{y}_p(t+i))^2 + \sum_{i=1}^{M} \gamma(\Delta u(t+i-1)^T \Delta u(t+i-1)) \tag{3.7}$$

Where $w(t+i)$ is the setpoint at time $t+i$, $\hat{y}_p$ are the future process outputs predicted over the prediction horizon N, and $u(t+i)$ is the future control signal. λ and γ represent the output and input weightings respectively. Also, online scrolling optimization is under the constraint of closed-loop prediction

$$\begin{aligned} y_{min} \leq \hat{y}_p(t+i) \leqslant y_{max} \quad & (i = 1, \ldots, N) \\ u_{min} \leq u(t+i) \leqslant u_{max} \quad & (i = 1, \ldots, M-1) \\ \Delta u_{min} \leq \Delta u(t+i) \leqslant \Delta u_{max} \quad & (i = 1, \ldots, M-1) \end{aligned} \tag{3.8}$$

The u_{min} and u_{max} are the minimum and maximum values of the operation inputs. Δu_{min} and Δu_{min} represent their corresponding changes. The calculation of future control signals involves minimizing the objective function so that the process output is as close as possible to the given reference trajectory even in the presence of loading disturbances. The control actions are computed at every sampling time by solving an optimization problem while taking consideration of constraints on the output and inputs[1]. The control signal u is manipulated only within the control horizon, and remains constant afterwards, i.e. $u(k+i) = u(k+M-1)$ for $i = M, ..., N-1$.

4. WORHP Solver

After completing the dynamic model and control model, next step is to setup the DIP for TransWORHP.

WORHP[5],which also refers as eNLP (European NLP solver) by ESA, is designed to find locally optimal points of optimization problems and solve numeric problems. In this project it works as a NLP-solver.

Now we use TransWORHP as NLP-solver. It is an addon which can optimize the control systems who can give ODE form.It can be optimized under certain conditions given by: objective function, additional boundary conditions, and limits of states and inputs.

4.1. DIP Setup for TransWORHP

Before formulating the optimal control problem we have some terms need to be introduced[6].

- Dynamic of the system (ode-structure, ode-diff).
- Variables of the system.
- Objective function (obj-structure, obj-diff).
- Boundary conditions.
- Control and state constraints.

4.1.1. Dynamic system

From Chapter 2 we can get the dynamic model. So the state vector Q for double inverted pendulum is defined as:

$$Q = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \theta_1 \\ \theta_2 \\ x_w \\ \dot{x}_w \end{bmatrix} \tag{4.1}$$

Where we can get differential equation with additional entry input energy E_u:

$$x[7] = \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \end{bmatrix} = \begin{bmatrix} Q \\ E_u \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \theta_1 \\ \theta_2 \\ x_w \\ \dot{x}_w \\ E_u \end{bmatrix} \tag{4.2}$$

Also from (2.9) we can get the ODE functions:

$$\begin{aligned}
&\dot{x}[0] = -1 * M_{11}^{-1}[0][0] * (M_{12}[0] * \dot{x}[5] + C_1[0] + g_1[0]) - M_{11}^{-1}[0][1] \\
&* (M_{12}[1] * \dot{x}[5] + C_1[1] + g_1[1]) \\
&\dot{x}[1] = -1 * M_{11}^{-1}[1][0] * (M_{12}[0] * \dot{x}[5] + C_1[0] + g_1[0]) - M_{11}^{-1}[1][1] \\
&* (M_{12}[1] * \dot{x}[5] + C_1[1] + g_1[1]) \\
&\dot{x}[2] = x[0] \\
&\dot{x}[3] = x[1] \\
&\dot{x}[4] = x[5] \\
&\dot{x}[5] = -1 * (x[5] - u)/\tau \\
&\dot{x}[6] = u * u
\end{aligned} \tag{4.3}$$

4.1.2. Objective function

In nonlinear control design we can get the objective function from 3.7 :

$$
\begin{aligned}
J_0 &= (x(j,0) - ziel[0])^2 * weight[0] \\
J_1 &= (x(j,1) - ziel[1])^2 * weight[1]; \\
J_2 &= (fabs(fmod(x(j,2) - ziel[2] + Pi, Pi2) - Pi))^2 * weight[2]; \\
J_3 &= (fabs(fmod(x(j,3) - ziel[3] + Pi, Pi2) - Pi))^2 * weight[3]; \\
J_4 &= (x(j,4) - ziel[4])^2 * weight[4]; \\
J_5 &= (x(j,5) - ziel[5])^2 * weight[5]; \\
J_6 &= (x(j,6) - ziel[6]) * weight[6]; \\
N_1 &= cos(x(j,2)) - cos(ziel[2]), N_2 = cos(x(j,3)) - cos(ziel[3]) \\
J_7 &= N_1 * (l_1 * g * m_2 + g * m_1 * a_1) + (N_2 * a_2 * m_2 * g))^2 * weight[7] \\
minJ &= \sum_{i=0}^{7} J_i
\end{aligned}
\tag{4.4}
$$

Here n_dis is the number of grid points, j is the last time step of the prediction, Pi as π , $Pi2$ as $2 * \pi$. We define a setpoint vector $ziel[7] = [0, 0, 0, pi, 0, 0, 0]$. $weight[8]$ represents the output and input weightings. The control horizon is very well known, it is also n_dis, or rather $T[n\_dis-1]$. we set up control horizon $T[n\_dis - 1]$ as $dis\_times[] = \{0, 0.1, 0.21, 0.42, 0.63, 0.84, 1.05\}$, we leave away u^2, as we don't need a small acceleration. Also, I don't know how to use u in obj. On the other hand, we do use the integral of u^2, to prefer smaller velocities in our solution, in $x(j, 6)$. J_7 is the Pendulum energy, J_6 is the input energy, J_5 is the velocity of the cart, and J_4 is the distance of the cart. J_3 is the angle of pendulum 2, so we need to use function to calculate the true angle. J_2 is the angle of pendulum 1. J_1 and J_0 should be angle speed of pendulum 2 and pendulum 1.

4.1.3. Boundary conditions & constraints

As for Boundary conditions of state values, we have to set:

$$
\begin{aligned}
up\_bnd[7] &= \{1.2, 25, 25, 1e20, 1e20, 0.38, 1.2, 0.5\}; \\
lw\_bnd[7] &= \{-1.2, -25, -25, -1e20, -1e20, -0.38, -1.2, -0.5\};
\end{aligned}
\tag{4.5}
$$

u_boundary (control boundary). control value is the first state of state value:

$$\begin{aligned} u\_low[0] = lw\_bnd[0] = -1.2 \quad J \\ u\_upp[0] = up\_bnd[0] = 1.2 \quad J \end{aligned} \tag{4.6}$$

x_boundary (system boundary). System value is from second state value to 7th state value, which represents pendulum1 angle speed, pendulum2 angle speed, pendulum1 angle, pendulum2 angle, cart distance, and cart speed.

$$\begin{aligned} x\_low[0] = lw\_bnd[1] = -25 \quad rad/s \qquad & x\_upp[0] = up\_bnd[1] = 25 \quad rad/s \\ x\_low[1] = lw\_bnd[2] = -25 \quad rad/s \qquad & x\_upp[1] = up\_bnd[2] = 25 \quad rad/s \\ x\_low[2] = lw\_bnd[3] = -1e20 \quad rad \qquad & x\_upp[2] = up\_bnd[3] = 1e20 \quad rad \\ x\_low[3] = lw\_bnd[4] = -1e20 \quad rad \qquad & x\_upp[3] = up\_bnd[4] = 1e20 \quad rad \\ x\_low[4] = lw\_bnd[5] = -0.38 \quad m \qquad & x\_upp[4] = up\_bnd[5] = 0.38 \quad m \\ x\_low[5] = lw\_bnd[6] = -1.2 \quad m/s \qquad & x\_upp[5] = up\_bnd[6] = 1.2 \quad m/s \\ x\_low[6] = lw\_bnd[7] = -0.5 \quad J \qquad & x\_upp[6] = up\_bnd[7] = 0.5 \quad J \end{aligned} \tag{4.7}$$

var_boundary is the initial value of state value $start[7] = [0, 0, pi, pi, 0, 0, 0]$:

$$\begin{aligned} x\_low[x\_index(0,0)] = x\_upp[x\_index(0,0)] = start[0]; \\ x\_low[x\_index(0,1)] = x\_upp[x\_index(0,1)] = start[1]; \\ x\_low[x\_index(0,2)] = x\_upp[x\_index(0,2)] = start[2]; \\ x\_low[x\_index(0,3)] = x\_upp[x\_index(0,3)] = start[3]; \\ x\_low[x\_index(0,4)] = x\_upp[x\_index(0,4)] = start[4]; \\ x\_low[x\_index(0,5)] = x\_upp[x\_index(0,5)] = start[5]; \\ x\_low[x\_index(0,6)] = x\_upp[x\_index(0,6)] = start[6]; \end{aligned} \tag{4.8}$$

(4.9)

4.1.4. Derivative structures

The derivative structures in TransWORHP [6] will be discussed as follows. Computation time can be reduced if the structure of the Jacobi matrix is well known. However we have to discover the structure of dynamic model and objective function.
In *ode_structure* from TransWORHP and (4.3), we know that every entry of

the Jacobian that is not 0 has to be stated, so we can get:

$$
\begin{aligned}
&s(0, x\_indexode(0)); && \%d\dot{x}[0]/dx[0]\\
&s(0, x\_indexode(1)); && \%d\dot{x}[0]/dx[1]\\
&s(0, x\_indexode(2)); && \%d\dot{x}[0]/dx[2]\\
&s(0, x\_indexode(3)); && \%d\dot{x}[0]/dx[3]\\
&s(0, x\_indexode(5)); && \%d\dot{x}[0]/dx[5]\\
&s(0, u\_indexode(0)); && \%d\dot{x}[0]/du[0]\\
&s(1, x\_indexode(0)); && \%d\dot{x}[1]/dx[0]\\
&s(1, x\_indexode(1)); && \%d\dot{x}[1]/dx[1]\\
&s(1, x\_indexode(2)); && \%d\dot{x}[1]/dx[2]\\
&s(1, x\_indexode(3)); && \%d\dot{x}[1]/dx[3]\\
&s(1, x\_indexode(5)); && \%d\dot{x}[1]/dx[5]\\
&s(1, u\_indexode(0)); && \%d\dot{x}[1]/du[0]\\
&s(2, x\_indexode(0)); && \%d\dot{x}[2]/dx[0]\\
&s(3, x\_indexode(1)); && \%d\dot{x}[3]/dx[1]\\
&s(4, x\_indexode(5)); && \%d\dot{x}[4]/dx[5]\\
&s(5, x\_indexode(5)); && \%d\dot{x}[5]/dx[5]\\
&s(5, u\_indexode(0)); && \%d\dot{x}[5]/du[0]\\
&s(6, u\_indexode(0)); && \%d\dot{x}[6]/du[0]
\end{aligned}
\tag{4.10}
$$

In the same way, we can get *obj_structure* from the objective function 4.4

$$
\begin{aligned}
&s(0, x\_indexode(0)); \% && d\dot{x}[0]/dx[0]\\
&s(0, x\_indexode(1)); \% && d\dot{x}[0]/dx[1]\\
&s(0, x\_indexode(2)); \% && d\dot{x}[0]/dx[2]\\
&s(0, x\_indexode(3)); \% && d\dot{x}[0]/dx[3]\\
&s(0, x\_indexode(4)); \% && d\dot{x}[0]/dx[4]\\
&s(0, x\_indexode(5)); \% && d\dot{x}[0]/dx[5]\\
&s(0, x\_indexode(6)); \% && d\dot{x}[0]/dx[6]
\end{aligned}
\tag{4.11}
$$

Derivative values

At discrete points, these values become more imprecise than the discrete distances, and the analysis of derivatives is more accurate.

Another way to specify this in TransWORHP, is to overloaded Ode_diff for the ordinary differential equation differential, and Obj_diff for objective function differential.
After a series of partial derivative calculations, we can get the Obj_diff as follows:

$$
\begin{aligned}
&s(0, x\_indexode(0)) = 2 * (x(j,0) - ziel[0]) * weight[0];\\
&s(0, x\_indexode(1)) = 2 * (x(j,1) - ziel[1]) * weight[1];\\
&s(0, x\_indexode(2)) = 2 * fabs(fmod(x(j,2) - ziel[2] + Pi, Pi2) - Pi) * weight[2];\\
&N = cos(x(j,2)) - cos(ziel[2]);\\
&s(0, x\_indexode(2)) += 2 * N * (l1 * g * m2 + g * m1 * a1) * sin(x(j,2)) * weight[7];\\
&s(0, x\_indexode(3)) = 2 * fabs(fmod(x(j,3) - ziel[3] + Pi, Pi2) - Pi) * weight[3];\\
&M = cos(x(j,3)) - cos(ziel[3]);\\
&s(0, x\_indexode(3)) += -2 * (M) * a2 * m2 * g * sin(x(j,3)) * weight[7];\\
&s(0, x\_indexode(4)) = 2 * (x(j,4) - ziel[4]) * weight[4];\\
&s(0, x\_indexode(5)) = 2 * (x(j,5) - ziel[5]) * weight[5];\\
&s(0, x\_indexode(6)) = weight[6];
\end{aligned}
\tag{4.12}
$$

As for Ode_diff, there are massive calculations to do, including long formula. You can check the result from the Appendix. Here you can see two different ways to test if the calculations are correct or not.
1. Using Matlab Simulink (which is already builed by student A.Arreortua) to build the Ode_diff function and Ode function (you can find the ode Matlab code in Appendix). The aim of this simulation is to make sure if the results are same. We can find it out from the scope and we have to calculate again. For example, we set the third value of constant1 and constant2 (other states should keep 0) in figure 4.1, we can correct the Ode_diff with original ODE. The result will be plotted like following (because of the space reason, I didn't put the the 7th diagram, so you can only find 6 diagrams figure now):

We can see the results that different functions are matching now. If there are any mistakes in derivative values calculation, the different plot will appears. For example, in figure 4.2, it is obvious that there is one state which does not

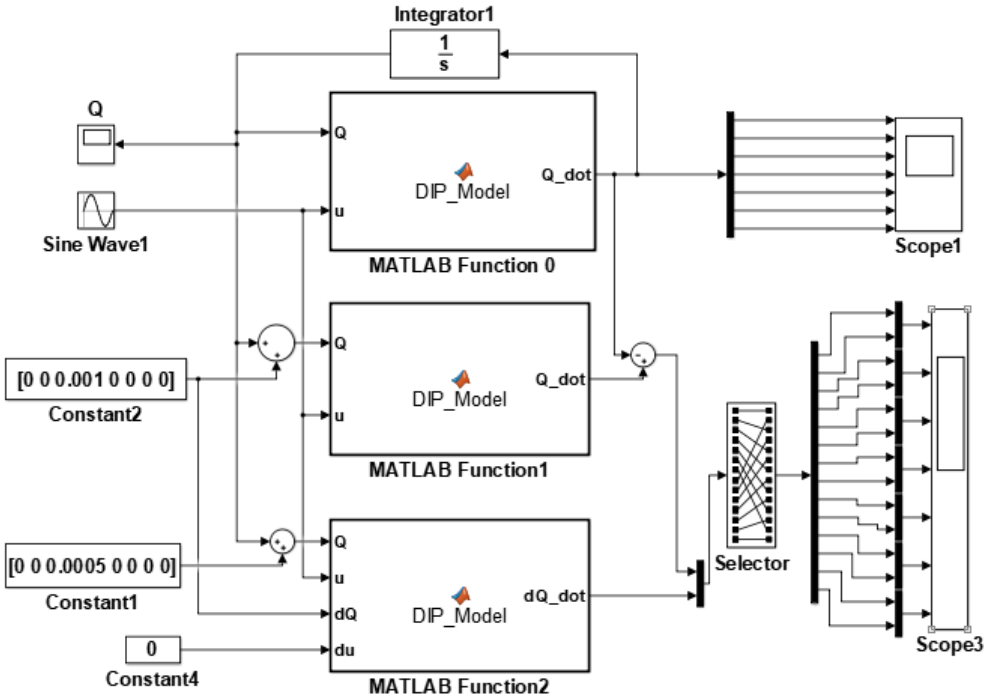

Figure 4.1.: ODE matlab Simulink diagram.

match, as the caves are different. However, the value is really small that we can ignore the error.

2. We can reference documents [6] and use the Viewer function from TransWORHP. When we run the code and have a look at the DG-Window, you can see how constraints and variables are ordered. If the ode_diff formula is right, the result will be printed out in green color as figure 4.9 shows. Else the color will turns red. Then you can see which part of ode is wrong and then make the correction.

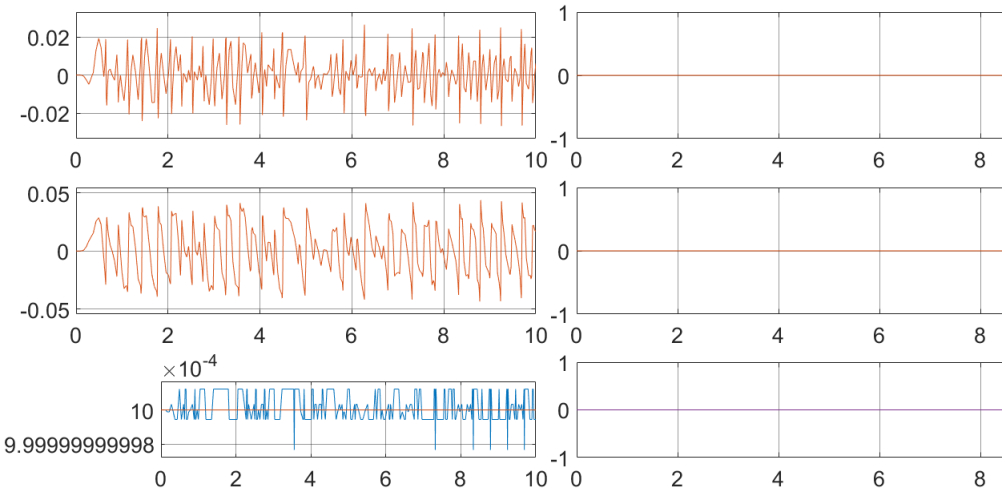

Figure 4.2.: Derivation of x[0] for each formula diagram.

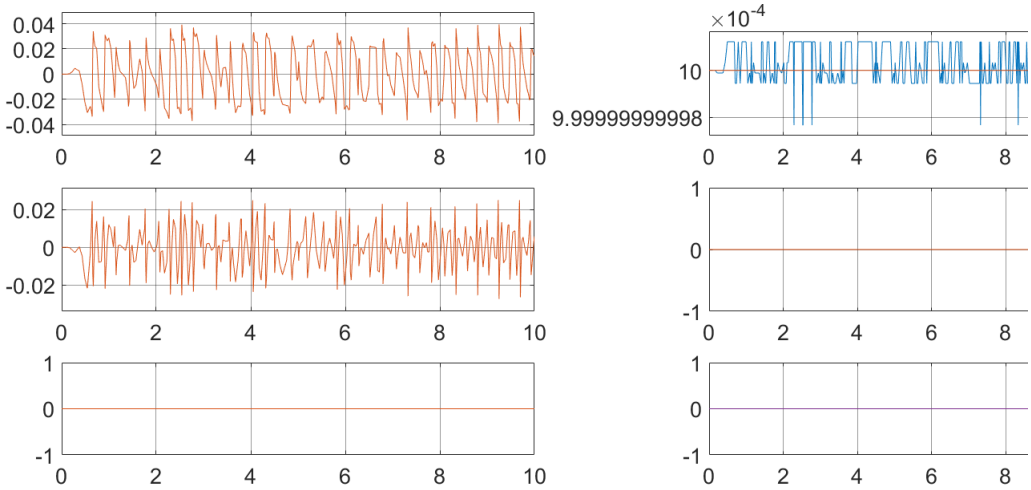

Figure 4.3.: Derivation of x[1] for each formula diagram.

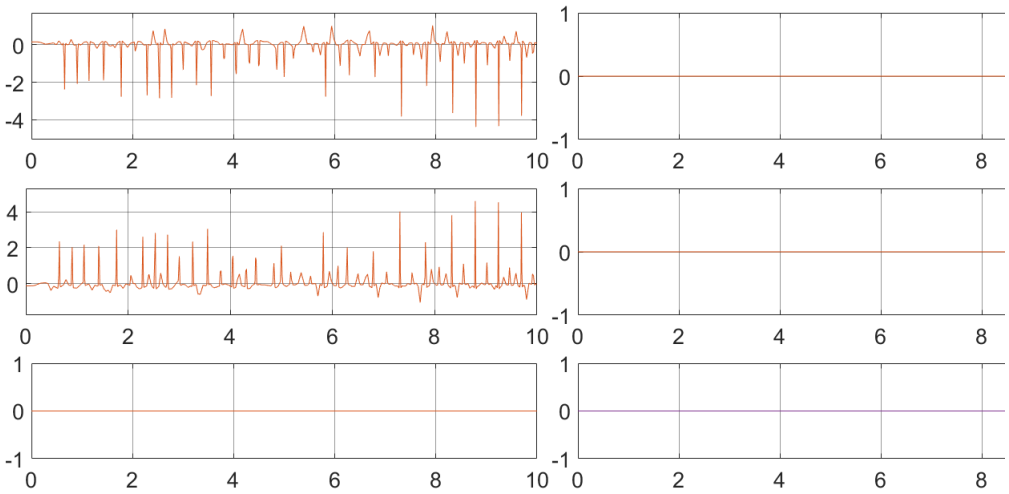

Figure 4.4.: Derivation of x[2] for each formula diagram.

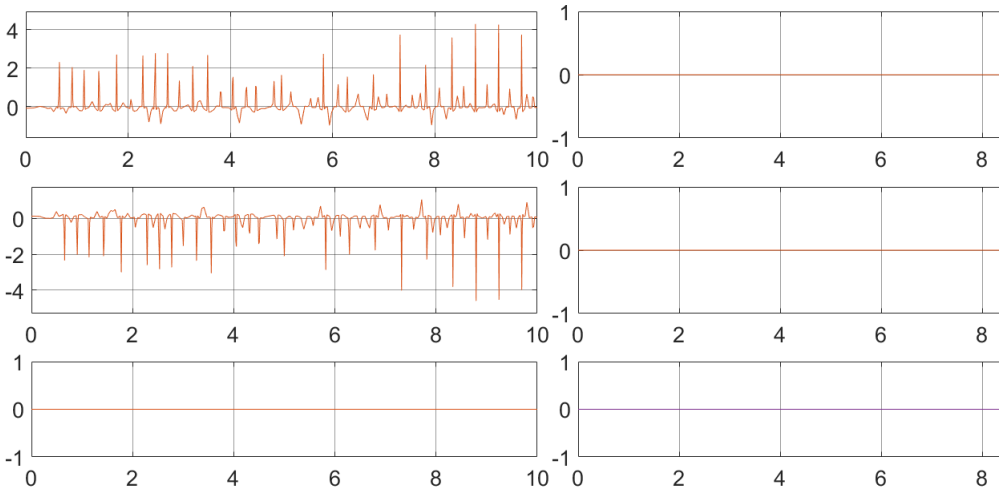

Figure 4.5.: Derivation of x[3] for each formula diagram.

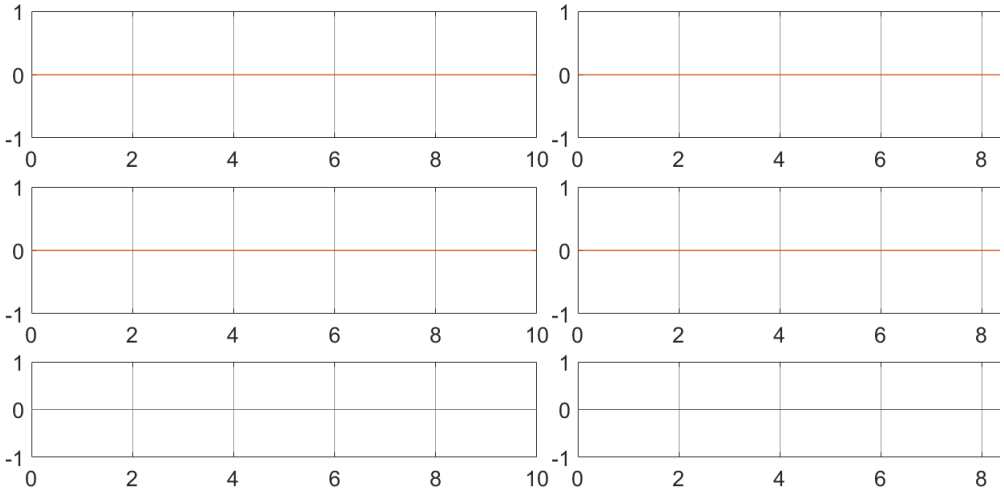

Figure 4.6.: Derivation of x[4] for each formula diagram.

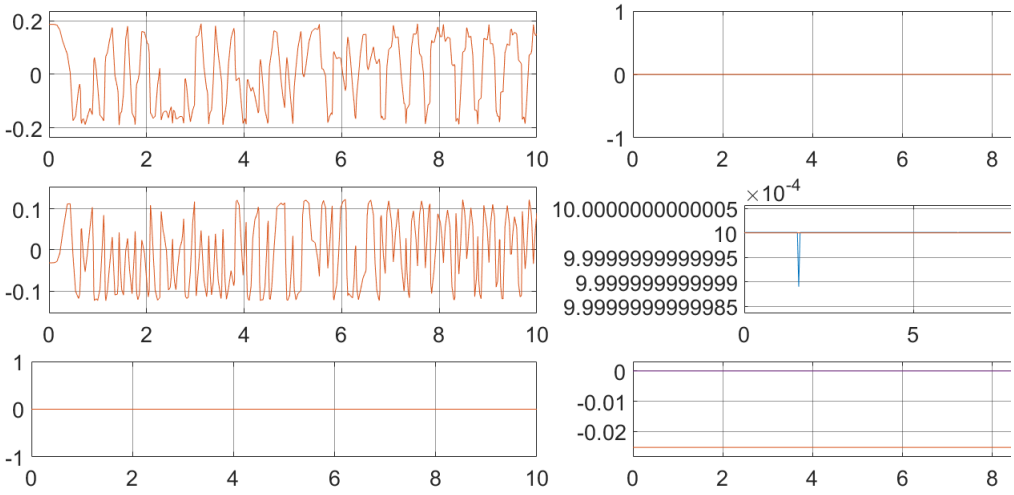

Figure 4.7.: Derivation of x[5] for each formula diagram.

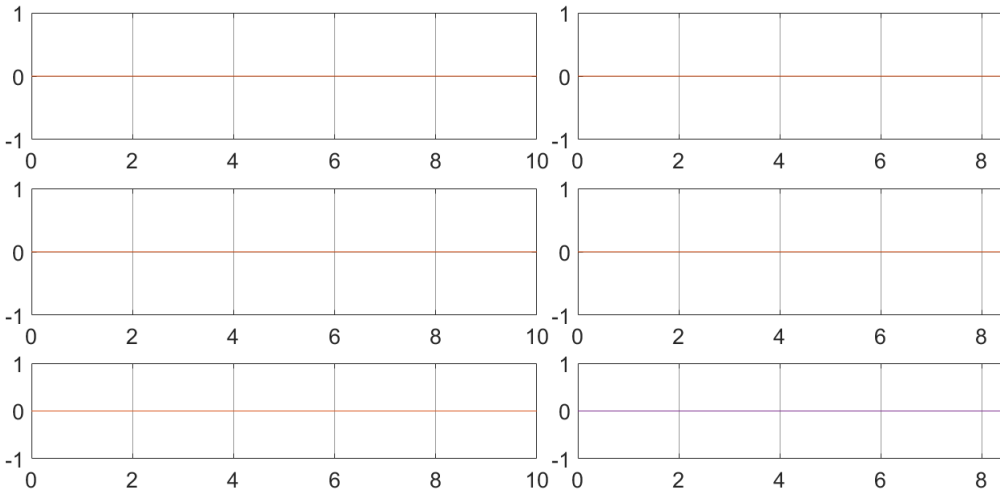

Figure 4.8.: Derivation of u[0] for each formula diagram.

DG(2/ 0)-> phase 0 ODE 0 (dx 2)	7.82266	7.82266
DG(2/ 1)-> phase 0 ODE 1 (dx 2)	-6.3528	-6.3528
DG(2/ 2)-> phase 0 ODE 2 (dx 2)	-1	-1
DG(3/ 0)-> phase 0 ODE 0 (dx 3)	-4.19366	-4.19366
DG(3/ 1)-> phase 0 ODE 1 (dx 3)	5.74883	5.74883
DG(3/ 3)-> phase 0 ODE 3 (dx 3)	-1	-1
DG(4/ 4)-> phase 0 ODE 4 (dx 4)	-1	-1
DG(5/ 0)-> phase 0 ODE 0 (dx 5)	9.36528	9.36528
DG(5/ 1)-> phase 0 ODE 1 (dx 5)	-1.55863	-1.55863
DG(5/ 4)-> phase 0 ODE 4 (dx 5)	-0.05	-0.05
DG(5/ 5)-> phase 0 ODE 5 (dx 5)	0.265823	0.265823
DG(6/ 6)-> phase 0 ODE 6 (dx 6)	-1	-1
DG(7/ 0)-> phase 0 ODE 0 (du 0)	-9.36528	-9.36528
DG(7/ 1)-> phase 0 ODE 1 (du 0)	1.55863	1.55863
DG(7/ 5)-> phase 0 ODE 5 (du 0)	-1.26582	-1.26582
DG(7/ 6)-> phase 0 ODE 6 (du 0)	0.0215977	0.0215977
DG(8/ 0)-> phase 0 ODE 0 (dx 0)	1.14039	1.14039
DG(8/ 1)-> phase 0 ODE 1 (dx 0)	-0.196162	-0.196162
DG(8/ 2)-> phase 0 ODE 2 (dx 0)	-0.05	-0.05
DG(8/ 7)-> phase 0 ODE 0 (dx 0)	-0.845576	-0.845576
DG(8/ 8)-> phase 0 ODE 1 (dx 0)	-0.215778	-0.215778
DG(8/ 9)-> phase 0 ODE 2 (dx 0)	-0.055	-0.055
DG(9/ 0)-> phase 0 ODE 0 (dx 1)	0.0198064	0.0198064
DG(9/ 1)-> phase 0 ODE 1 (dx 1)	0.98422	0.98422
DG(9/ 3)-> phase 0 ODE 3 (dx 1)	-0.05	-0.05
DG(9/ 7)-> phase 0 ODE 0 (dx 1)	0.0217871	0.0217871
DG(9/ 8)-> phase 0 ODE 1 (dx 1)	-1.01736	-1.01736
DG(9/ 10)-> phase 0 ODE 3 (dx 1)	-0.055	-0.055
DG(10/ 0)-> phase 0 ODE 0 (dx 2)	9.6019	9.6019
DG(10/ 1)-> phase 0 ODE 1 (dx 2)	-8.72086	-8.72086
DG(10/ 2)-> phase 0 ODE 2 (dx 2)	1	1
DG(10/ 7)-> phase 0 ODE 0 (dx 2)	10.5621	10.5621
DG(10/ 8)-> phase 0 ODE 1 (dx 2)	-9.59294	-9.59294
DG(10/ 9)-> phase 0 ODE 2 (dx 2)	-1	-1
DG(11/ 0)-> phase 0 ODE 0 (dx 3)	-5.41796	-5.41796
DG(11/ 1)-> phase 0 ODE 1 (dx 3)	7.86724	7.86724
DG(11/ 3)-> phase 0 ODE 3 (dx 3)	1	1
DG(11/ 7)-> phase 0 ODE 0 (dx 3)	-5.95976	-5.95976
DG(11/ 8)-> phase 0 ODE 1 (dx 3)	8.65397	8.65397
DG(11/ 10)-> phase 0 ODE 3 (dx 3)	-1	-1
DG(12/ 4)-> phase 0 ODE 4 (dx 4)	1	1
DG(12/ 11)-> phase 0 ODE 4 (dx 4)	-1	-1
DG(13/ 0)-> phase 0 ODE 0 (dx 5)	8.20773	8.20773
DG(13/ 1)-> phase 0 ODE 1 (dx 5)	-0.535591	-0.535591

Figure 4.9.: ODE variables from DG-Window.

5. Run the program from Visual Studio with TransWORHP

We get an earlier C code version which is running by Visual Studio. So I start to check every part i have mentioned. After correcting the ode_diff and objective function, we need to run the program in Visual Studio with TransWORHP Solver.

For NMPC we additionally programmed a loop running the optimization repeatedly, using the first input signal on another DIP model called "real", and taking its states as measurement for the next MPC run. To make things simple for starters, the MPC model is identical to the "real" model.

To make the NMPC work under real time conditions, the calculation times are measured, and the optimization is not started with taking the measurement. Instead of the measurement, we integrate the computation time roughly, and take the result as start state of the optimization. So the optimization will finish roughly when the integrated start state is reached in reality. To make things simpler for starters, the computation time is faked to be constant value as $0.1s$.

During the running process of the program, the TransWORHP tries to find the optimal solution in each time step. I ran the same code for four times with different weights. If finding the optimal solution is marked as 1, failing to find the optimal solution as 0, which can be seen from figure 5.1. Thus we can find out that the optimization often fails so the curves are not optimal in these time steps. This is due to an old version of TransWORHP.

5.1. Visualization result in Matlab

Although we can not find the optimal solution at each time step, we can also look at the ouput data from MPC.dat(detail data in each loop), MPC2.dat(operation data), MPC3.dat(optimization data + simulation data), and MPC4.dat(computation time after each loop).
We can put the data file MPC3.dat(Shun1b.dat in Matlab) and MPC2.dat(

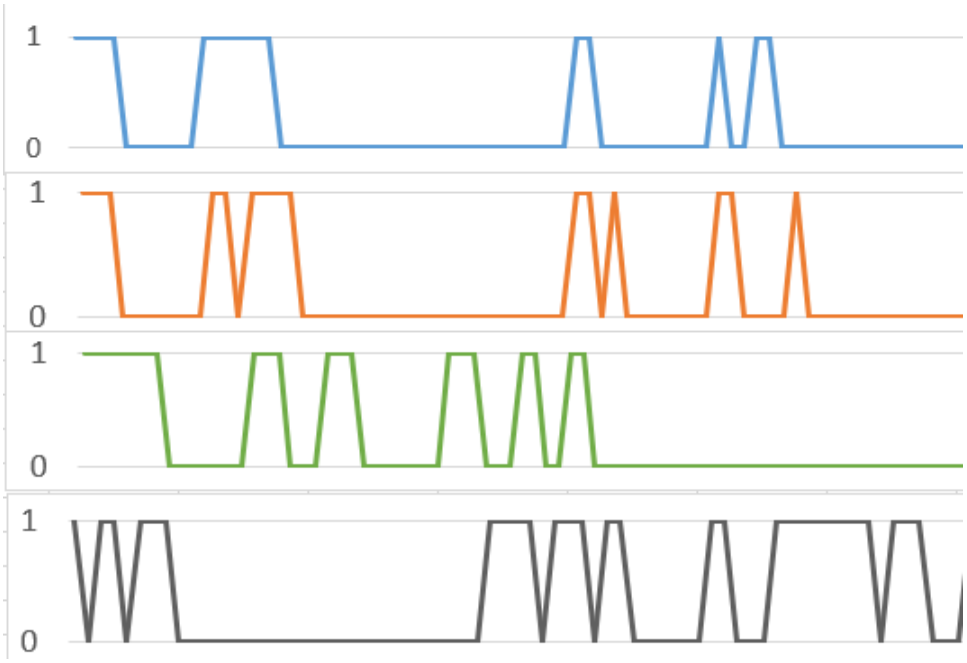

Figure 5.1.: Optimal Solution

Shun1.dat in Matlab) into Matlab visualization(which is provided by thr supervisor, the Matlab code is put into the appendix). Then we can get the figure as figure 5.2:

As we can see in Figure 5.2: when the pendulum is swinged up, the total energy will decreased to 0, especially when it is between 2s and 3s the pendulum is almost swinged up to the aim position. So the energy is reduced to 0. However in the old version of transWORHP, it might directly use objective function to calculate the final aim (swingup two pendulum). As we can find out in the ODE Part in Figure 5.3, this is the result of dynamic model calculation. In the pendulum position part(2s-3s), we can find that the Pendulum 1 and Pendulum 2 position tends to be at aimed position (pendulum 1=0, pendulum 2=Pi). In the sub-figure Pendulum velocity, the velocity tend to be 0. So we can get the result that the TransWORHP calculation is trying to find out the correct solutions, but sometimes also fails to find out the optimal solution.

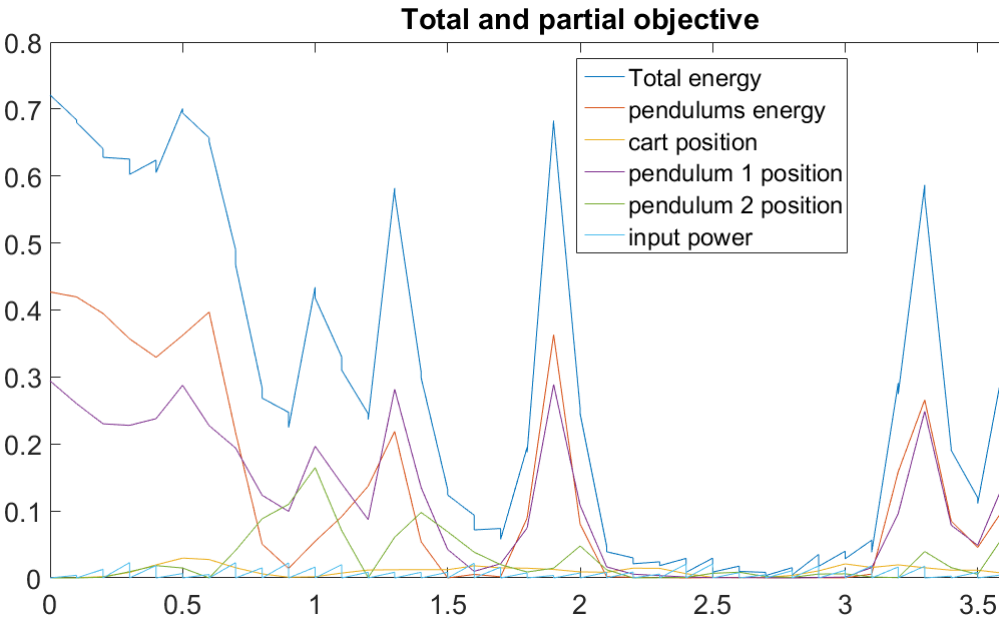

Figure 5.2.: total and partial objective

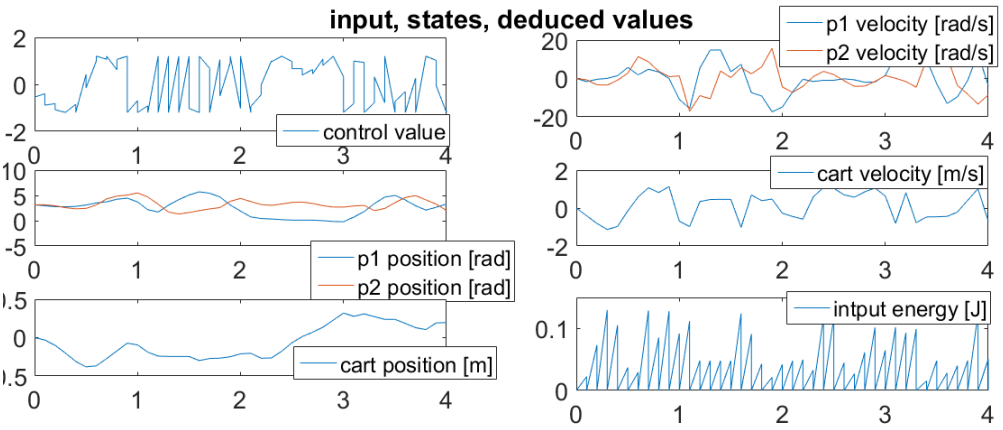

Figure 5.3.: Input, state, deduced values

6. Conclusion and Future Work

During the process, we setup the controller for swinging up the double inverted pendulum by coupling a highly efficient solver(TransWORHP). However, the situation till now is that we can swing up the double inverted pendulum but it can not stand stable. So we need to set an additional balance controller to stabilize the pendulum in future work.
Moreover, system should also includes real-time issues, as well as some model errors to check for robustness.
As I have mentioned, after the discussion with supervisor, I still used the old version of TransWORHP (optimal solver). Although I know that the results of the optimization may not be successful, we still have some successful optimizations. A new version of TransWORHP would be used if possible, which might provide better performance and optimization with higher success rates
Finally, I think we also need to find out the suitable parameters $weight[8]$ for the variables in the objective function. As another improvement, not only the weight parameters but also all of the objective functions would be reviewed, there would be a better function for these problems. I hope that in the future work of this project, we can use our efficient solver to get the desired effect in dealing with some non-linear process control.

A. ODE Diff in C code

```
bool ode_diff(DiffStructure& s, double t, const double* x,
const double* u, const double* p) {
//return false;
def = (h4 * h6 - h5 * h5 * cos(x[2] - x[3])* cos(x[2] - x[3]));
P = h5 * h5 * sin(2 * (x[2] - x[3]));

//dx[0] = -1 * M11_inv[0][0] * N1 - M11_inv[0][1] * N2;

s(0, x_indexode(0)) = -(2*x[0]*h5*h5*cos(x[2]-x[3])*sin(x[2]-x[3]))/def;

s(0, x_indexode(1)) = -(2 * x[1] * h5 * h6 * sin(x[2] - x[3]))/ def;

s(0, x_indexode(2)) = h2*h6*(u[0] - x[5])*(sin(x[2])*def
        + P*cos(x[2]))/(T1*def*def) - h5*h6*x[1] * x[1]
        * (cos(x[2]-x[3])*def-P*sin(x[2]-x[3]))/(def*def)+h6*h7
        * (cos(x[2])*def-P*sin(x[2]))/(def*def)-h3*h5*(u[0]-x[5])
        * cos(x[3])*(sin(x[2]-x[3])*def+P*cos(x[2]-x[3]))
        / (def*def*T1)-h5*h5*x[0]*x[0]*(2*cos(2*(x[2]-x[3]))
        * def-P*sin(2*(x[2]-x[3])))/(2*def*def)+h5*h8*sin(x[3])
        * (sin(x[2]-x[3])*def+P*cos(x[2]-x[3]))/(def*def);

s(0, x_indexode(3)) = -h2*h6*cos(x[2])*(u[0] - x[5])*P
        / (T1*def*def)-h6*h5*x[1]*x[1]*(-cos(x[2]-x[3])*def
        + P*sin(x[2]-x[3]))/(def*def)+h6*h7*sin(x[2])*P
        / (def*def)+ h5*h3*(u[0]- x[5])*(sin(x[2]-2*x[3])
        * def+P*cos(x[2]-x[3])*cos(x[3]))/(T1*def*def)
        - h5*h5*x[0]* x[0]*(-2*cos(2*(x[2]-x[3]))*def+P
        * sin(2*(x[2]-x[3])))/(2*def*def)-h8*h5
        * (cos(x[2]-2*x[3])*def+P*cos(x[2]-x[3])*sin(x[3]))/(def*def);

s(0, x_indexode(4))=0;
s(0, x_indexode(5))=(h2*h6*cos(x[2])-h3*h5*cos(x[3])*cos(x[2]-x[3]))/(T1*(def));
s(0, u_indexode(0))=-(h2*h6*cos(x[2])-h3*h5*cos(x[3])*cos(x[2]-x[3]))/(T1*(def));

// dx[1]=dx[1] = -1 * M11_inv[1][0] * N1 - M11_inv[1][1] * N2;
s(1 , x_indexode(0))=(2*x[0]*h4*h5*sin(x[2]-x[3]))
                    /(h4*h6-h5*h5*cos(x[2]-x[3])*cos(x[2]-x[3]));

s(1 , x_indexode(1)) = (2 * x[1] * h5 * h5 * cos(x[2]-x[3])
    * sin(x[2]-x[3]))/(h4*h6-h5*h5*cos(x[2]-x[3])*cos(x[2]-x[3]));

s(1 , x_indexode(2)) = h5*h2*(u[0] - x[5])*((-cos(x[2] - x[3])
        * sin(x[2])-sin(x[2]-x[3])*cos(x[2]))*def-P*cos(x[2])
        * cos(x[2]-x[3]))/(T1*def*def) + h5*h5*x[1]*x[1]
        * (2*cos(2*(x[2]-x[3]))*def-P*sin(2*(x[2]-x[3])))
        / (2*def*def)-h5*h7*(cos(2 *x[2]-x[3])*def-P
        * cos(x[2]-x[3])*sin(x[2]))/(def*def)+h4*h3*(u[0]-x[5])
        * P*cos(x[3])/(T1*def*def)+h5*h4*x[0]*x[0]
        * (cos(x[2]-x[3])*def-P*sin(x[2]-x[3]))/(def*def)
        - h4*h8*sin(x[3])*P/(def*def);

s(1, x_indexode(3)) = h2*h5*(u[0]-x[5])*cos(x[2])
        * (sin(x[2]-x[3])*def+P*cos(x[2]-x[3]))/(def*def*T1)
        + h5*h5*x[1]*x[1]*(-2*cos(2*(x[2]-x[3]))*def
        + P *sin(2*(x[2]-x[3])))/(2*def*def)-h5*h7*sin(x[2])
        * (sin(x[2]-x[3])*def+P*cos(x[2]-x[3]))/(def*def)
        - h4*h3*(u[0]-x[5])*(-sin(x[3])*def+P*cos(x[3]))
        / (T1*def*def)+h4*h5*x[0]*x[0]*(-cos(x[2] - x[3])*def
        + P*sin(x[2] - x[3])) / (def*def) + h4*h8*(cos(x[3])*def
        + P*sin(x[3])) / (def*def);

s(1, x_indexode(4)) = 0;
s(1, x_indexode(5)) = ((h3 * h4 * cos(x[3])) - (h2 * h5
        * cos(x[2]) * cos(x[2] - x[3]))) / (T1 * (h4 * h6 - h5 * h5
```

```
        * cos(x[2] - x[3]) * cos(x[2] - x[3])));

s(1, u_indexode(0)) = -((h3 * h4 * cos(x[3])) - (h2 * h5
        * cos(x[2]) * cos(x[2] - x[3]))) / (T1 * (h4 * h6 - h5 * h5
        * cos(x[2] - x[3]) * cos(x[2] - x[3])));

// dx[2]= x[0];
s(2, x_indexode(0)) = 1;
s(2, x_indexode(1)) = 0;
s(2, x_indexode(2)) = 0;
s(2, x_indexode(3)) = 0;
s(2, x_indexode(4)) = 0;
s(2, x_indexode(5)) = 0;
s(2, u_indexode(0)) = 0;

// dx[3]= x[1];
s(3, x_indexode(0)) = 0;
s(3, x_indexode(1)) = 1;
s(3, x_indexode(2)) = 0;
s(3, x_indexode(3)) = 0;
s(3, x_indexode(4)) = 0;
s(3, x_indexode(5)) = 0;
s(3, u_indexode(0)) = 0;

// dx[4]=x[5];
s(4, x_indexode(0)) = 0;
s(4, x_indexode(1)) = 0;
s(4, x_indexode(2)) = 0;
s(4, x_indexode(3)) = 0;
s(4, x_indexode(4)) = 0;
s(4, x_indexode(5)) = 1;
s(4, u_indexode(0)) = 0;

// dx[5]= -1 * (x[5] - u[0]) / T1;
s(5, x_indexode(0)) = 0;
s(5, x_indexode(1)) = 0;
s(5, x_indexode(2)) = 0;
s(5, x_indexode(3)) = 0;
s(5, x_indexode(4)) = 0;
s(5, x_indexode(5)) = -1 / T1;
s(5, u_indexode(0)) = 1 / T1;

// dx[6] = (u[0] * u[0]);
s(6, x_indexode(0)) = 0;      //!!!!!
s(6, x_indexode(1)) = 0;
s(6, x_indexode(2)) = 0;
s(6, x_indexode(3)) = 0;
s(6, x_indexode(4)) = 0;
s(6, x_indexode(5)) = 0;
s(6, u_indexode(0)) = 2 * u[0];

return true;
}
```

B. ODE in Matlab

B.1. Function 0

```
function Q_dot  = DIP_Model(Q,u)
%{
%Description: Double inverted pendulum model
%Purpose: Simulates the double inverted pendulum through
%         its mathematical dynamic model
%Input Parameters: State space "Q", control input "u"
%Output Parameters: Derivative state space "Q_dot"
%Author:A. Arreortua, MiaoshunWu
%*************************************************************
%}
%% Definition of constants
% gravity acceleration
g = 9.81; % [m/s^2]
a1=0.085; %Distance between pivot and center of mass of pendulum1
a2=0.157; %Distance between pivot and center of mass of pendulum2
l1= 0.170; %Length of pendulum1 [m]
l2= 0.314; %Length of pendulum2 [m]
m1=0.162; %Mass of pendulum1 [kg]
m2=0.203; %Mass of pendulum2 [kg]
J1 = (m1*l1^2)/12; %Inertia of pendulum1 around the joint [kg m^2]
J2 = (m2*l2^2)/12; %Inertia of pendulum2 around the joint [kg m^2]
% Model of motor behaviour: PT1 tau
T1 = 0.0395; % [s]
% Dinamic model constants
h2 = m1*a1+m2*l1;
h3 = m2*a2;
h4 = m1*a1^2+m2*l1^2+J1;
h5 = m2*a2*l1;
h6 = m2*a2^2+J2;
h7 = m1*a1*g + m2*l1*g;
h8 = m2*a2*g;

Q0=Q(1);
Q1=Q(2);
Q2=Q(3);
Q3=Q(4);
Q5=Q(6);

det = (h4 * h6 - h5 * h5 * cos(Q2 - Q3) * cos(Q2 - Q3));
M11inv= [h6/det ,-h5*cos(Q2-Q3)/det ; -h5*cos(Q2-Q3)/det ,h4/det];
M12 = [h2*cos(Q2); h3*cos(Q3)];
C1 = [h5*(Q1^2)*sin(Q2-Q3); -h5*(Q0^2)*sin(Q2-Q3)];
G1 = [-h7*sin(Q2); -h8*sin(Q3)];

%% Dynamic equations
x_2dot = ((u-Q5)/T1);
Theta_2dot = -M11inv*(M12* x_2dot + C1 + G1);
Q_dot = [Theta_2dot;Q0;Q1;Q5;x_2dot;u*u];
```

B.2. Function 1

```
function Q_dot = DIP_Model(Q,u)
% gravity acceleration
g = 9.81; % [m/s^2]
a1=0.085; %Distance between pivot and center of mass of pendulum1
a2=0.157; %Distance between pivot and center of mass of pendulum2
```

```
l1= 0.170; %Length of pendulum1 [m]
l2= 0.314;  %Length of pendulum2 [m]
m1=0.162; %Mass of pendulum1 [kg]
m2=0.203; %Mass of pendulum2 [kg]
J1 = (m1*l1^2)/12;     %Inertia of pendulum1 around the joint
J2 = (m2*l2^2)/12;     %Inertia of pendulum2 around the joint
% Model of motor behaviour: PT1 tau
T1 = 0.0395; % [s]
%Dinamic model constants
h2 = m1*a1+m2*l1;
h3 = m2*a2;
h4 = m1*a1^2+m2*l1^2+J1;
h5 = m2*a2*l1;
h6 = m2*a2^2+J2;
h7 = m1*a1*g + m2*l1*g;
h8 = m2*a2*g;

Q0=Q(1);
Q1=Q(2);
Q2=Q(3);
Q3=Q(4);
Q5=Q(6);

det = (h4* h6 - h5 * h5 * cos(Q2 - Q3) * cos(Q2 - Q3));
M11inv= [h6/det ,-h5*cos(Q2-Q3)/det ; -h5*cos(Q2-Q3)/det ,h4/det];
M12 = [h2*cos(Q2); h3*cos(Q3)];
C1 = [h5*(Q1^2)*sin(Q2-Q3); -h5*(Q0^2)*sin(Q2-Q3)];
G1 = [-h7*sin(Q2); -h8*sin(Q3)];


x05 = ((u-Q5)/T1);
N1 = (M12(1)*x05 + C1(1) + G1(1));
N2 = (M12(2)*x05 + C1(2) + G1(2));
x00 = -M11inv(1,1) *N1 +(- M11inv(1,2) * N2 );
x01 = -M11inv(2,1) *N1 +(- M11inv(2,2) * N2 );
x02 = Q0;
x03 = Q1;
x04 = Q5;
x06 = u * u;
dx=[x00;x01;x02;x03;x04;x05;x06];
Q_dot=dx;
end
```

B.3. Function 2

```
function dQ_dot  = DIP_Model(Q,u,dQ,du)
%{
%Description: Double inverted pendulum model
%Purpose: Simulates the double inverted pendulum through
%         its mathematical dynamic model
%Input Parameters: State space "Q", control input "u"
%Output Parameters: Derivative state space "Q_dot"
%Author: A.Arreortua, Miaoshun Wu
%*****************************************************************
%}
%% Definition of constants
g = 9.81; % [m/s^2]
a1=0.085; %Distance between pivot and center of mass of pendulum1
a2=0.157; %Distance between pivot and center of mass of pendulum2
l1= 0.170; %Length of pendulum1 [m]
l2= 0.314;  %Length of pendulum2 [m]
m1=0.162; %Mass of pendulum1 [kg]
m2=0.203; %Mass of pendulum2 [kg]
J1 = (m1*l1^2)/12;     %Inertia of pendulum1 around the joint
J2 = (m2*l2^2)/12;     %Inertia of pendulum2 around the joint
% Model of motor behaviour: PT1 tau
T1 = 0.0395; % [s]
%Dinamic model constants
h2 = m1*a1+m2*l1;
h3 = m2*a2;
h4 = m1*a1^2+m2*l1^2+J1;
h5 = m2*a2*l1;
h6 = m2*a2^2+J2;
h7 = m1*a1*g + m2*l1*g;
```

```
h8 = m2*a2*g;

x0=Q(1);
x1=Q(2);
x2=Q(3);
x3=Q(4);
x5=Q(6);
def=(h4 * h6 - h5 * h5 * cos(x2 - x3)* cos(x2 - x3));
P=h5 * h5 * sin(2*(x2 - x3));
%x00 = -M11inv(1,1) *N1 +(- M11inv(1,2) * N2 );
%x05 = ((u-x5)/T1);
%N1  = (M12(1)*x05 + C1(1) + G1(1));
%N2  = (M12(2)*x05 + C1(2) + G1(2));

%x00 = -M11inv(1,1) *N1 +(- M11inv(1,2) * N2 );
s00  = -(2 * x0 * h5 * h5 * cos(x2 - x3) * sin(x2 - x3)) / def;
s01  = -(2 * x1 * h5 * h6 * sin(x2 - x3)) / def;

s02= h2*h6*(u-x5)*(sin(x2)*def+P*cos(x2))/(T1*def*def)
        - h5*h6*x1*x1*(cos(x2-x3)*def-P*sin(x2-x3))/(def*def)
        + h6*h7*(cos(x2)*def-P*sin(x2))/(def*def)-h3*h5*(u-x5)*cos(x3)
        * (sin(x2-x3)*def+P*cos(x2-x3))/(def*def*T1)- h5*h5*x0*x0
        * (2*cos(2*(x2-x3))*def-P*sin(2*(x2-x3)))/(2*def*def)
        + h5*h8*sin(x3)*(sin(x2-x3)*def+P*cos(x2-x3))/(def*def);

s03= -h2*h6*cos(x2)*(u-x5)*P/(T1*def*def) - h6*h5*x1*x1
        * (-cos(x2-x3)*def+P*sin(x2-x3))/(def*def) + h6*h7*sin(x2)*P
        / (def*def) + h5*h3*(u-x5)*(sin(x2-2*x3)*def+P*cos(x2-x3)
        * cos(x3))/(T1*def*def)-h5*h5*x0*x0*(-2*cos(2*(x2-x3))*def
        + P*sin(2*(x2-x3)))/(2*def*def)-h8*h5*(cos(x2-2*x3)*def
        + P*cos(x2-x3)*sin(x3))/(def*def);

s04 = 0;
s05 = ( h2 * h6 * cos(x2)- h3 * h5 * cos(x3) * cos(x2 - x3))
    / (T1 * (def));

s06 = -(h2 * h6 * cos(x2)- h3 * h5 * cos(x3) * cos(x2 - x3))
     /(T1 * (def));

ddx0 = dQ(1) * s00 + dQ(2)*s01+dQ(3)*s02+dQ(4)*s03+dQ(5)*s04
     + dQ(6)*s05+du*s06;


%x01 = -M11inv(2,1) *N1 +(- M11inv(2,2) * N2 );
s10 = (2 * x0 * h4 * h5 * sin(x2 - x3)) / def;
s11 = (2 * x1 * h5 * h5 * cos(x2 - x3) * sin(x2 - x3)) /def;
s12 = h5*h2*(u-x5)*((-cos(x2-x3)*sin(x2)-sin(x2-x3)*cos(x2))
        * def-P*cos(x2)*cos(x2-x3))/(T1*def*def) + h5*h5*x1*x1
        * (2*cos(2*(x2-x3))*def-P*sin(2*(x2-x3)))/(2*def*def) - h5*h7
        * (cos(2*x2-x3)*def-P*cos(x2-x3)*sin(x2))/(def*def) + h4*h3*(u-x5)
        * P*cos(x3)/(T1*def*def) + h5*h4*x0*x0*(cos(x2-x3)*def-P
        * sin(x2-x3))/(def*def) - h4*h8*sin(x3)*P/(def*def);

s13= h2*h5*(u-x5)*cos(x2)*(sin(x2-x3)*def+P*cos(x2-x3))
        / (def*def*T1)+h5*h5*x1*x1*(-2*cos(2*(x2-x3))*def+P*sin(2*(x2-x3)))
        / (2*def*def)-h5*h7*sin(x2)*(sin(x2-x3)*def+P*cos(x2-x3))
        / (def*def)-h4*h3*(u-x5)*(-sin(x3)*def+P*cos(x3))/(T1*def*def)
        + h4*h5*x0*x0*(-cos(x2-x3)*def+P*sin(x2-x3))/(def*def)
        + h4*h8*(cos(x3)*def+P*sin(x3))/(def*def);

s14 = 0;

s15 =  ((h3 * h4 * cos(x3)) - (h2 * h5 * cos(x2) * cos(x2 - x3)))
    / (T1 * (h4 * h6 - h5 * h5 * cos(x2 - x3) * cos(x2 - x3)));

s16 = -((h3 * h4 * cos(x3)) - (h2 * h5 * cos(x2) * cos(x2 - x3)))
    / (T1 * (h4 * h6 - h5 * h5 * cos(x2 - x3) * cos(x2 - x3)));

ddx1=dQ(1)*s10+dQ(2)*s11+dQ(3)*s12+dQ(4)*s13+dQ(5)*s14+dQ(6)* s15+du*s16;

ddx2=dQ(1);
%s(2, x_indexode(0)) = 1;
%s(2, x_indexode(1)) = 0;
%s(2, x_indexode(2)) = 0;
%s(2, x_indexode(3)) = 0;
%s(2, x_indexode(4)) = 0;
```

```
%s(2, x_indexode(5)) = 0;
%s(2, u_indexode(0)) = 0;

ddx3=dQ(2);
%s(3, x_indexode(0)) = 0;
%s(3, x_indexode(1)) = 1;
%s(3, x_indexode(2)) = 0;
%s(3, x_indexode(3)) = 0;
%s(3, x_indexode(4)) = 0;
%s(3, x_indexode(5)) = 0;
%s(3, u_indexode(0)) = 0;

ddx4=dQ(6);
%s(4, x_indexode(0)) = 0;
%s(4, x_indexode(1)) = 0;
%s(4, x_indexode(2)) = 0;
%s(4, x_indexode(3)) = 0;
%s(4, x_indexode(4)) = 0;
%s(4, x_indexode(5)) = 1;
%s(4, u_indexode(0)) = 0;

ddx5=(du-dQ(6))/ T1;
%s(5, x_indexode(0)) = 0;
%s(5, x_indexode(1)) = 0;
%s(5, x_indexode(2)) = 0;
%s(5, x_indexode(3)) = 0;
%s(5, x_indexode(4)) = 0;
%s(5, x_indexode(5)) = -1 / T1;
%s(5, u_indexode(0)) = 1 / T1;
ddx6 =(du* du);
%s(6, x_indexode(0)) = 0;      //!!!!!
%s(6, x_indexode(1)) = 0;
%s(6, x_indexode(2)) = 0;
%s(6, x_indexode(3)) = 0;
%s(6, x_indexode(4)) = 0;
%s(6, x_indexode(5)) = 0;
%s(6, u_indexode(0)) = 2 * u;

%% Dynamic equations
dQ_dot=[ddx0;ddx1;ddx2;ddx3;ddx4;ddx5;ddx6];
```

C. MPC C code in Visual Studio

```
/*-----------------------------------------------------
*
* MPC_Doppelpendel: Test MPC controller for the Double
* Inverse pendulum
*----------------------------------------------------- */

#ifdef WIN32
#include "windows.h"
#endif
//#define NOGRAPHICS -> Buggy!
#include <conio.h>
#include "TransWORHP.h"
using namespace std;
// define
const double Pi =      3.14159265358898;
const double Pi2 =     6.2831853071796;
const double Pi2inv = 0.159154943091895;

//Pendulum 1,Pendulum 2 should hang down  <Reference>
double        start[] = { 0, 0, Pi, Pi,  0, 0, 0 };
//Pendulum 1 swinged up ,Pendulum 2 should hang down
const  double  ziel[] = { 0, 0, 0,  Pi,  0, 0, 0 };
double start_real[] = { 0, 0, Pi, Pi, 0, 0, 0 };
double weight[] = {0.0,0.0,1.0,1.0,0.2,0.0,0.2,1.0};
// parameter in Objective function
// anglespeed ,1anglespeed 2,angle1,angle2,cartposition
   ,cartspeed,input power,pendulum energy

// Set limits for pendulum 2  <constraints>
const double up_bnd[]= {1.2,25,25,1e20,1e20,0.38,1.2, 0.5 };
const double lw_bnd[]={-1.2,-25,-25,-1e20,-1e20,-0.38,-1.2,0};
const double dis_times[]= {0,0.1,0.21,0.42,0.63, 0.84, 1.05 };

// if the current computation time is lower than pre_calc_time
// started only at the time of the control sequence
// Pre-calculated time
const double pre_calc_time = 0.0;
//Original assumption of the calculation period
//executes the estimated computation time
const double ext_estim_time = 0.1;

double control[6];                //Control matrix
double states[6][7];          //State matrix
int error;

double max_bnd[7];
//  to simple the angle like 3pi= 1pi
inline double wrap2pi(double angle) {
return (angle - Pi2 * floor(angle*Pi2inv + 0.5));
}
//  transfer the continuous mode to discrete mode
inline double interp1(const double x[2], double y[2],
 double xi) {
return (((xi - x[0])*y[1] + (x[1] - xi)*y[0])/(x[1] - x[0]));
}
// model constants
const double  a[2] = { 81.48 , 5135.0 };
const double  g = 9.81;           //gravitation
const double  T1 = 0.0395;        // time constant [s]
const double  a1 = 0.085;         // Half pendulum neck 1 [m]
const double  a2 = 0.157;         // Half pendulum neck 2 [m]
const double  l1 = 0.17;          // pendulum neck  1 [m]
const double  l2 = 0.314;         // pendulum neck  2 [m]
const double  m = 0.5;            // Cart mass [kg]
```

```
const double  m1 = 0.162;        // Mass pendulum 1 [kg]
const double  m2 = 0.203;        // Mass pendulum 2 [kg]
const double  J1 = ((m1 * l1 * l1) / 12);
//Moment of inertia pendulum 1 [kg m^2]

const double  J2 = ((m2 * l2 * l2) / 12);
//Moment of inertia pendulum 2 [kg m^2]

const double  h1 = m + m1 + m2;
const double  h2 = m1*a1 + m2*l1;
const double  h3 = m2*a2;
const double  h4 = m1*a1*a1 + m2*l1*l1 + J1;
const double  h5 = m2*a2*l1;
const double  h6 = m2*a2*a2 + J2;
const double  h7 = m1*a1*g + m2*l1*g;
const double  h8 = m2*a2*g;
class MPC_Doppelpendel_optim : public TransWORHP {public:
MPC_Doppelpendel_optim(int dis) :
TransWORHP("MPC_Doppelpendel_optim", dis, 7, 1, 0, 0, 0) {}

void GetXTitle(int d, char *s) {
if (d == 0) strcpy(s, "angle_velocity_of_pendulum_1");
if (d == 1) strcpy(s, "angle_velocity_of_pendulum_2");
if (d == 2) strcpy(s, "angle_of_pendulum_1");
if (d == 3) strcpy(s, "angle_of_pendulum_2");
if (d == 4) strcpy(s, "cart_displacement");
if (d == 5) strcpy(s, "cart_velocity");
if (d == 6) strcpy(s, "input_energy");
}
void GetUTitle(int d, char *s) {
if (d == 0) strcpy(s, "input_velocity");
}


//------- Start optimization (optional) --------------------//
void x_init(double *x, int i, int dis) {
for (j = 0; j < n_ode; j++) {   //n_ode:number of states
x[j] = states[i][j];
}
}
void u_init(double *u, int i, int dis) {
u[0] = 0.0;
}

//---------- Objective Function ------------------------------//
// J=(x-ziel)^2+u^2
double obj() {
double tmp, alpha, ret;
j = n_dis - 1;                          // number of grid points

tmp = (x(j, 0) - ziel[0]);
ret = tmp*tmp * weight[0];              // Pendulum angle speed 1

tmp = (x(j, 1) - ziel[1]);
ret += tmp*tmp * weight[1];             // Pendulum angle speed 2

//alpha = fabs(fmod(x(j, 2) - ziel[0] + Pi, Pi2) - Pi);
tmp = fabs(fmod(x(j, 2) - ziel[2] + Pi, Pi2) - Pi);
ret += tmp*tmp * weight[2];          // pendulum angle 1

tmp = fabs(fmod(x(j, 3) - ziel[3] + Pi, Pi2) - Pi);
ret += tmp*tmp * weight[3];             // pendulum angle 2

tmp = (x(j, 4) - ziel[4]);
ret += tmp*tmp * weight[4];                  // carriage position

tmp = (x(j, 5) - ziel[5]);
ret += tmp*tmp * weight[5];                  // cart speed

tmp = (x(j, 6) - ziel[6]);                   // input power
ret += tmp * weight[6];

// Here pendulum energy is calculated
tmp = cos(x(j, 2)) - cos(ziel[2]);
tmp*= l1*g*m2 + g*m1*a1;
tmp += (cos(x(j, 3)) - cos(ziel[3]))*a2*m2*g;
```

```
ret += tmp*tmp* weight[7];
//  V = Vcart + Vpendulum1 + Vpendulum2
//    = T1 + T2 + T3 = 0+ m1*g*a1*cosθ1+m2g(l1*cosθ1 + a2*cosθ2)

return ret;
}

bool obj_structure(DiffStructure &s) {

s(0, x_indexode(0));
s(0, x_indexode(1));
s(0, x_indexode(2));
s(0, x_indexode(3));
s(0, x_indexode(4));
s(0, x_indexode(5));
s(0, x_indexode(6));
return true;
}

bool obj_diff(DiffStructure &s) {
//return false;
s(0, x_indexode(0))  = 2 * (x(j, 0) - ziel[0])* weight[0];
s(0, x_indexode(1))  = 2 * (x(j, 1) - ziel[1])* weight[1];
N= fabs(fmod(x(j, 2)-ziel[2] + Pi,Pi2) - Pi);
s(0, x_indexode(2))  = 2 *N* weight[2];
N1=cos(x(j, 2))-cos(ziel[2]))*(l1*g*m2 + g*m1*a1)*sin(x(j, 2);
s(0, x_indexode(2)) += -2 *N1*weight[7];
M=fabs(fmod(x(j, 3) - ziel[3] + Pi, Pi2) - Pi);
s(0, x_indexode(3)) = 2 * M* weight[3];
M1= (cos(x(j, 3)) - cos(ziel[3]))*  a2*m2*g*(-sin(x(j, 3)));
s(0, x_indexode(3)) += 2 *M1*weight[7];
s(0, x_indexode(4))  = 2 * (x(j, 4) - ziel[4])* weight[4];
s(0, x_indexode(5))  = 2 * (x(j, 5) - ziel[5])* weight[5];
s(0, x_indexode(6))  = weight[6];

return true;
}

//--------- System model ----------------------------------//

double det;
double M11_inv[2][2];
double M12[2];
double C1[2];
double G1[2];
double N1;
double N2;
double def;
double P;

void ode(double *dx, double t, const double *x,
const double *u,const double *p) {

det = (h4*h6-h5*h5*cos(x[2]-x[3])*cos(x[2]-x[3]));

M11_inv[0][0] = h6 / det;
M11_inv[0][1] = (-1 * h5 * cos(x[2] - x[3])) / det;
M11_inv[1][0] = (-1 * h5 * cos(x[2] - x[3])) / det;
M11_inv[1][1] = h4 / det;

M12[0] = h2 * cos(x[2]);
M12[1] = h3 * cos(x[3]);

C1[0] = h5* x[1] * x[1] * sin(x[2] - x[3]);
C1[1] = -1 *h5* x[0] * x[0] * sin(x[2] - x[3]);

G1[0] = -1 * h7 * sin(x[2]);
G1[1] = -1 * h8 * sin(x[3]);
dx[5] = -1 * (x[5] - u[0]) / T1;
N1 = (M12[0] * dx[5] + C1[0] + G1[0]);
N2 = (M12[1] * dx[5] + C1[1] + G1[1]);
dx[0] = -1 * M11_inv[0][0] * N1 - M11_inv[0][1] * N2;
dx[1] = -1 * M11_inv[1][0] * N1 - M11_inv[1][1] * N2;
dx[2] = x[0];
dx[3] = x[1];
dx[4] = x[5];
```

```
dx[6] = u[0] * u[0];
}
//--------- Jacobian for Derivatives (Optional) ----------//
bool ode_structure(DiffStructure &s) {
//return false;
s(0, x_indexode(0));
s(0, x_indexode(1));
s(0, x_indexode(2));
s(0, x_indexode(3));
s(0, x_indexode(5));
s(0, u_indexode(0));

s(1, x_indexode(0));
s(1, x_indexode(1));
s(1, x_indexode(2));
s(1, x_indexode(3));
s(1, x_indexode(5));
s(1, u_indexode(0));

s(2, x_indexode(0));

s(3, x_indexode(1));

s(4, x_indexode(5));

s(5, x_indexode(5));
s(5, u_indexode(0));

s(6, u_indexode(0));

return true;
}
bool ode_diff(DiffStructure& s, double t, const double* x,
const double* u, const double* p) {

//return false;
def = (h4*h6-h5*h5*cos(x[2]-x[3])*cos(x[2]-x[3]));
P = h5 * h5 * sin(2 * (x[2] - x[3]));

//dx[0] = -1 * M11_inv[0][0] * N1 - M11_inv[0][1] * N2;
s(0, x_indexode(0))=-(2*x[0]*h5*h5*cos(x[2]-x[3])
*sin(x[2]-x[3]))/ def;

s(0, x_indexode(1)) = -(2 * x[1] * h5 * h6
*sin(x[2] - x[3])) / def;

s(0, x_indexode(2)) = h2*h6*(u[0] - x[5])*(sin(x[2])*def
+ P*cos(x[2])) /(T1*def*def) - h5*h6*x[1] * x[1]
* (cos(x[2] - x[3])*def - P*sin(x[2] - x[3])) / (def*def)
+ h6*h7*(cos(x[2])*def - P*sin(x[2])) / (def*def)
- h3*h5*(u[0] - x[5])*cos(x[3])*(sin(x[2] - x[3])*def + P
* cos(x[2] - x[3])) / (def*def*T1) - h5*h5*x[0] * x[0]
* (2 * cos(2 * (x[2] - x[3]))*def - P*sin(2 * (x[2] - x[3])))
/ (2 * def*def) + h5*h8*sin(x[3])* (sin(x[2] - x[3])*def
+ P*cos(x[2] - x[3])) / (def*def);

s(0, x_indexode(3)) = -h2*h6*cos(x[2])*(u[0] - x[5])*P
/ (T1*def*def) - h6*h5*x[1] * x[1] * (-cos(x[2] - x[3])* def
+ P*sin(x[2] - x[3])) / (def*def) + h6*h7*sin(x[2])* P
/ (def*def) + h5*h3*(u[0] - x[5])*(sin(x[2] - 2 * x[3]* def
+ P*cos(x[2] - x[3])*cos(x[3])) / (T1*def*def)
- h5*h5*x[0] * x[0] * (-2 * cos(2 * (x[2] - x[3]))*def+ P
* sin(2 * (x[2] - x[3]))) / (2 * def*def) - h8*h5
* (cos(x[2] - 2 * x[3])*def + P*cos(x[2] - x[3])*sin(x[3]))
/ (def*def);

s(0, x_indexode(4)) = 0;

s(0, x_indexode(5)) = (h2 * h6 * cos(x[2]) - h3 * h5
* cos(x[3]) * cos(x[2] - x[3])) / (T1 * (def));

s(0, u_indexode(0)) = -(h2 * h6 * cos(x[2]) - h3 * h5
* cos(x[3]) * cos(x[2] - x[3])) / (T1 * (def));

// dx[1]=dx[1] = -1 * M11_inv[1][0] * N1 - M11_inv[1][1] * N2;
s(1, x_indexode(0)) = (2 * x[0] * h4 * h5 * sin(x[2] - x[3]))
```

```
/ (h4 * h6 - h5 * h5 * cos(x[2] - x[3]) * cos(x[2] - x[3]));

s(1, x_indexode(1)) = (2 * x[1] * h5 * h5 * cos(x[2] - x[3])
* sin(x[2] - x[3])) / (h4 * h6 - h5 * h5 * cos(x[2] - x[3])
* cos(x[2] - x[3]));

s(1, x_indexode(2)) = h5*h2*(u[0] - x[5])*((-cos(x[2] -x[3])
* sin(x[2]) - sin(x[2]- x[3])*cos(x[2]))*def - P*cos(x[2])
* cos(x[2] - x[3])) / (T1*def*def) + h5*h5*x[1]* x[1]
* (2 * cos(2 * (x[2] - x[3]))*def - P*sin(2*(x[2] - x[3])))
/ (2 * def*def) - h5*h7*(cos(2 * x[2]- x[3])*def - P
* cos(x[2] - x[3])*sin(x[2])) / (def*def)+ h4*h3*(u[0]-x[5])
* P*cos(x[3]) / (T1*def*def) + h5*h4*x[0] *x[0]
* (cos(x[2] - x[3])* def - P*sin(x[2]- x[3])) / (def*def)
- h4*h8*sin(x[3])*P / (def*def);

s(1, x_indexode(3)) = h2*h5*(u[0] - x[5])*cos(x[2])*(sin(x[2]
- x[3])*def + P*cos(x[2] - x[3])) / (def*def*T1) + h5*h5*x[1]
* x[1]*(-2 * cos(2 * (x[2] - x[3]))*def + P*sin(2 * (x[2]
- x[3])))/(2 * def*def) - h5*h7*sin(x[2])*(sin(x[2] - x[3])
* def + P*cos(x[2] - x[3]))/(def*def) - h4*h3*(u[0] - x[5])
* (-sin(x[3])*def + P*cos(x[3]))/(T1*def*def) + h4*h5*x[0]
* x[0] * (-cos(x[2] - x[3])*def + P*sin(x[2] - x[3]))
/ (def*def) + h4*h8*(cos(x[3])*def + P*sin(x[3])) / (def*def);

s(1, x_indexode(4)) = 0;

s(1, x_indexode(5)) = ((h3 * h4 * cos(x[3])) - (h2 * h5
* cos(x[2]) * cos(x[2] - x[3]))) / (T1 * (h4 * h6 - h5 * h5
* cos(x[2] - x[3]) * cos(x[2] - x[3])));

s(1, u_indexode(0)) = -((h3 * h4 * cos(x[3])) - (h2 * h5
* cos(x[2]) * cos(x[2] - x[3]))) / (T1 * (h4 * h6 - h5 * h5
* cos(x[2] - x[3]) * cos(x[2] - x[3])));


// dx[2]= x[0];
s(2, x_indexode(0)) = 1;
s(2, x_indexode(1)) = 0;
s(2, x_indexode(2)) = 0;
s(2, x_indexode(3)) = 0;
s(2, x_indexode(4)) = 0;
s(2, x_indexode(5)) = 0;
s(2, u_indexode(0)) = 0;

// dx[3]= x[1];
s(3, x_indexode(0)) = 0;
s(3, x_indexode(1)) = 1;
s(3, x_indexode(2)) = 0;
s(3, x_indexode(3)) = 0;
s(3, x_indexode(4)) = 0;
s(3, x_indexode(5)) = 0;
s(3, u_indexode(0)) = 0;

// dx[4]=x[5];
s(4, x_indexode(0)) = 0;
s(4, x_indexode(1)) = 0;
s(4, x_indexode(2)) = 0;
s(4, x_indexode(3)) = 0;
s(4, x_indexode(4)) = 0;
s(4, x_indexode(5)) = 1;
s(4, u_indexode(0)) = 0;

// dx[5]= -1 * (x[5] - u[0]) / T1;
s(5, x_indexode(0)) = 0;
s(5, x_indexode(1)) = 0;
s(5, x_indexode(2)) = 0;
s(5, x_indexode(3)) = 0;
s(5, x_indexode(4)) = 0;
s(5, x_indexode(5)) = -1 / T1;
s(5, u_indexode(0)) = 1 / T1;

// dx[6] = (u[0] * u[0]);
s(6, x_indexode(0)) = 0;
s(6, x_indexode(1)) = 0;
s(6, x_indexode(2)) = 0;
```

```
s(6, x_indexode(3)) = 0;
s(6, x_indexode(4)) = 0;
s(6, x_indexode(5)) = 0;
s(6, u_indexode(0)) = 2 * u[0];

return true;
}
bool ode_diff_p(DiffStructure& s, double t, const double *x,
const double *u, const double *p, int index) {
return false;
}


//--------- Boundaries -----------------------------------------//

void u_boundary(double *u_low, double *u_upp) {
//get the constraints from above
u_low[0] = lw_bnd[0];
u_upp[0] = up_bnd[0];
}

void x_boundary(double *x_low, double *x_upp) {
for (i = 0; i < n_ode; i++) {
x_low[i] = lw_bnd[i+1];
x_upp[i] = up_bnd[i+1];
}
}
void p_boundary(double *p_low, double *p_upp) {
}
void var_boundary(double *x_low, double *x_upp) {
for (i = 0; i < n_ode; i++) {
x_low[x_index(0, i)] = start[i];
x_upp[x_index(0, i)] = start[i];
}
}

private:
int i = 0, j = 0;
};
class MPC_Doppelpendel_real : public TransWORHP {

public:

MPC_Doppelpendel_real(int dis):TransWORHP(
"MPC_Doppelpendel_real",dis, 7, 1, 0, 0, 0) {}

void GetXTitle(int d, char *s) {
if (d == 0) strcpy(s, "angle_velocity_of_pendulum_1");
if (d == 1) strcpy(s, "angle_velocity_of_pendulum_2");
if (d == 2) strcpy(s, "angle_of_pendulum_1");
if (d == 3) strcpy(s, "angle_of_pendulum_2");
if (d == 4) strcpy(s, "cart_displacement");
if (d == 5) strcpy(s, "cart_velocity");
if (d == 6) strcpy(s, "input_energy");
}

void GetUTitle(int d, char *s) {
if (d == 0) strcpy(s, "input_velocity");
}

//--------- Start optimization (optional) ------------------//
void x_init(double *x, int i, int dis) {
}
void u_init(double *u, int i, int dis) {
}


//--------- Objective Function ---------------------------//
double obj() {
return 0;
}

bool obj_structure(DiffStructure &s) {
return false;
}
```

```
bool obj_diff(DiffStructure &s) {
return false;
}

//--------- System model ----------------------------------//

double det;
double M11_inv[2][2];
double M12[2];
double C1[2];
double G1[2];
double N1;
double N2;
double def;
double P;

void ode(double *dx, double t, const double *x,
const double *u,const double *p) {

det = (h4*h6-h5*h5*cos(x[2]-x[3])*cos(x[2]-x[3]));

M11_inv[0][0] = h6 / det;
M11_inv[0][1] = (-1 * h5 * cos(x[2] - x[3])) / det;
M11_inv[1][0] = (-1 * h5 * cos(x[2] - x[3])) / det;
M11_inv[1][1] = h4 / det;

M12[0] = h2 * cos(x[2]);
M12[1] = h3 * cos(x[3]);

C1[0] = h5* x[1] * x[1] * sin(x[2] - x[3]);
C1[1] = -1 *h5* x[0] * x[0] * sin(x[2] - x[3]);

G1[0] = -1 * h7 * sin(x[2]);
G1[1] = -1 * h8 * sin(x[3]);
dx[5] = -1 * (x[5] - u[0]) / T1;
N1 = (M12[0] * dx[5] + C1[0] + G1[0]);
N2 = (M12[1] * dx[5] + C1[1] + G1[1]);
dx[0] = -1 * M11_inv[0][0] * N1 - M11_inv[0][1] * N2;
dx[1] = -1 * M11_inv[1][0] * N1 - M11_inv[1][1] * N2;
dx[2] = x[0];
dx[3] = x[1];
dx[4] = x[5];
dx[6] = u[0] * u[0];
}
//--------- Jacobian for Derivatives (Optional) -------//
bool ode_structure(DiffStructure &s) {
//return false;
s(0, x_indexode(0));
s(0, x_indexode(1));
s(0, x_indexode(2));
s(0, x_indexode(3));
s(0, x_indexode(5));
s(0, u_indexode(0));

s(1, x_indexode(0));
s(1, x_indexode(1));
s(1, x_indexode(2));
s(1, x_indexode(3));
s(1, x_indexode(5));
s(1, u_indexode(0));

s(2, x_indexode(0));

s(3, x_indexode(1));

s(4, x_indexode(5));

s(5, x_indexode(5));
s(5, u_indexode(0));

s(6, u_indexode(0));

return true;
}
bool ode_diff(DiffStructure& s, double t, const double* x,
```

```
const double* u, const double* p) {

//return false;
s(0, x_indexode(0));
s(0, x_indexode(1));
s(0, x_indexode(2));
s(0, x_indexode(3));
s(0, x_indexode(5));
s(0, u_indexode(0));

s(1, x_indexode(0));
s(1, x_indexode(1));
s(1, x_indexode(2));
s(1, x_indexode(3));
s(1, x_indexode(5));
s(1, u_indexode(0));

s(2, x_indexode(0));

s(3, x_indexode(1));

s(4, x_indexode(5));

s(5, x_indexode(5));
s(5, u_indexode(0));

s(6, u_indexode(0));

return true;
}
bool ode_diff(DiffStructure& s, double t, const double* x,
const double* u, const double* p) {

//return false;
def = (h4*h6-h5*h5*cos(x[2]-x[3])*cos(x[2]-x[3]));
P = h5 * h5 * sin(2 * (x[2] - x[3]));

//dx[0] = -1 * M11_inv[0][0] * N1 - M11_inv[0][1] * N2;
s(0, x_indexode(0))=-(2*x[0]*h5*h5*cos(x[2]-x[3])
*sin(x[2]-x[3]))/ def;

s(0, x_indexode(1)) = -(2 * x[1] * h5 * h6
*sin(x[2] - x[3])) / def;

s(0, x_indexode(2)) = h2*h6*(u[0] - x[5])*(sin(x[2])*def
+ P*cos(x[2])) /(T1*def*def) - h5*h6*x[1] * x[1]
* (cos(x[2] - x[3])*def - P*sin(x[2] - x[3])) / (def*def)
+ h6*h7*(cos(x[2])*def - P*sin(x[2])) / (def*def)
- h3*h5*(u[0] - x[5])*cos(x[3])*(sin(x[2] - x[3])*def + P
* cos(x[2] - x[3])) / (def*def*T1) - h5*h5*x[0] * x[0]
* (2 * cos(2 * (x[2] - x[3]))*def - P*sin(2 * (x[2] - x[3])))
/ (2 * def*def) + h5*h8*sin(x[3])* (sin(x[2] - x[3])*def
+ P*cos(x[2] - x[3])) / (def*def);

s(0, x_indexode(3)) = -h2*h6*cos(x[2])*(u[0] - x[5])*P
/ (T1*def*def) - h6*h5*x[1] * x[1] * (-cos(x[2] - x[3])* def
+ P*sin(x[2] - x[3])) / (def*def) + h6*h7*sin(x[2])* P
/ (def*def) + h5*h3*(u[0] - x[5])*(sin(x[2] - 2 * x[3]* def
+ P*cos(x[2] - x[3])*cos(x[3])) / (T1*def*def)
- h5*h5*x[0] * x[0] * (-2 * cos(2 * (x[2] - x[3]))*def+ P
* sin(2 * (x[2] - x[3]))) / (2 * def*def) - h8*h5
* (cos(x[2] - 2 * x[3])*def + P*cos(x[2] - x[3])*sin(x[3]))
/ (def*def);

s(0, x_indexode(4)) = 0;

s(0, x_indexode(5)) = (h2 * h6 * cos(x[2]) - h3 * h5
* cos(x[3]) * cos(x[2] - x[3])) / (T1 * (def));

s(0, u_indexode(0)) = -(h2 * h6 * cos(x[2]) - h3 * h5
* cos(x[3]) * cos(x[2] - x[3])) / (T1 * (def));

// dx[1]=dx[1] = -1 * M11_inv[1][0] * N1 - M11_inv[1][1] * N2;
s(1, x_indexode(0)) = (2 * x[0] * h4 * h5 * sin(x[2] - x[3]))
/ (h4 * h6 - h5 * h5 * cos(x[2] - x[3]) * cos(x[2] - x[3]));
```

```
s(1, x_indexode(1)) = (2 * x[1] * h5 * h5 * cos(x[2] - x[3])
* sin(x[2] - x[3])) / (h4 * h6 - h5 * h5 * cos(x[2] - x[3])
* cos(x[2] - x[3]));

s(1, x_indexode(2)) = h5*h2*(u[0] - x[5])*((-cos(x[2] -x[3])
* sin(x[2]) - sin(x[2]- x[3])*cos(x[2]))*def - P*cos(x[2])
* cos(x[2] - x[3])) / (T1*def*def) + h5*h5*x[1]* x[1]
* (2 * cos(2 * (x[2] - x[3]))*def - P*sin(2*(x[2] - x[3])))
/ (2 * def*def) - h5*h7*(cos(2 * x[2]- x[3])*def - P
* cos(x[2] - x[3])*sin(x[2])) / (def*def)+ h4*h3*(u[0]-x[5])
* P*cos(x[3]) / (T1*def*def) + h5*h4*x[0] *x[0]
* (cos(x[2] - x[3])* def - P*sin(x[2]- x[3])) / (def*def)
- h4*h8*sin(x[3])*P / (def*def);

s(1, x_indexode(3)) = h2*h5*(u[0] - x[5])*cos(x[2])*(sin(x[2]
- x[3])*def + P*cos(x[2] - x[3])) / (def*def*T1) + h5*h5*x[1]
* x[1]*(-2 * cos(2 * (x[2] - x[3]))*def + P*sin(2 * (x[2]
- x[3])))/(2 * def*def) - h5*h7*sin(x[2])*(sin(x[2] - x[3])
* def + P*cos(x[2] - x[3]))/(def*def) - h4*h3*(u[0] - x[5])
* (-sin(x[3])*def + P*cos(x[3]))/(T1*def*def) + h4*h5*x[0]
* x[0] * (-cos(x[2] - x[3])*def + P*sin(x[2] - x[3]))
/ (def*def) + h4*h8*(cos(x[3])*def + P*sin(x[3])) / (def*def);

s(1, x_indexode(4)) = 0;

s(1, x_indexode(5)) = ((h3 * h4 * cos(x[3])) - (h2 * h5
* cos(x[2]) * cos(x[2] - x[3]))) / (T1 * (h4 * h6 - h5 * h5
* cos(x[2] - x[3]) * cos(x[2] - x[3])));

s(1, u_indexode(0)) = -((h3 * h4 * cos(x[3])) - (h2 * h5
* cos(x[2]) * cos(x[2] - x[3]))) / (T1 * (h4 * h6 - h5 * h5
* cos(x[2] - x[3]) * cos(x[2] - x[3])));


// dx[2]= x[0];
s(2, x_indexode(0)) = 1;
s(2, x_indexode(1)) = 0;
s(2, x_indexode(2)) = 0;
s(2, x_indexode(3)) = 0;
s(2, x_indexode(4)) = 0;
s(2, x_indexode(5)) = 0;
s(2, u_indexode(0)) = 0;

// dx[3]= x[1];
s(3, x_indexode(0)) = 0;
s(3, x_indexode(1)) = 1;
s(3, x_indexode(2)) = 0;
s(3, x_indexode(3)) = 0;
s(3, x_indexode(4)) = 0;
s(3, x_indexode(5)) = 0;
s(3, u_indexode(0)) = 0;

// dx[4]=x[5];
s(4, x_indexode(0)) = 0;
s(4, x_indexode(1)) = 0;
s(4, x_indexode(2)) = 0;
s(4, x_indexode(3)) = 0;
s(4, x_indexode(4)) = 0;
s(4, x_indexode(5)) = 1;
s(4, u_indexode(0)) = 0;

// dx[5]= -1 * (x[5] - u[0]) / T1;
s(5, x_indexode(0)) = 0;
s(5, x_indexode(1)) = 0;
s(5, x_indexode(2)) = 0;
s(5, x_indexode(3)) = 0;
s(5, x_indexode(4)) = 0;
s(5, x_indexode(5)) = -1 / T1;
s(5, u_indexode(0)) = 1 / T1;

// dx[6] = (u[0] * u[0]);
s(6, x_indexode(0)) = 0;
s(6, x_indexode(1)) = 0;
s(6, x_indexode(2)) = 0;
s(6, x_indexode(3)) = 0;
s(6, x_indexode(4)) = 0;
```

```
s(6, x_indexode(5)) = 0;
s(6, u_indexode(0)) = 2 * u[0];

return true;
}


//--------- Boundaries -----------------------------------//

void u_boundary(double *u_low, double *u_upp) {
}

void x_boundary(double *x_low, double *x_upp) {
}

void p_boundary(double *p_low, double *p_upp) {
}

void var_boundary(double *x_low, double *x_upp) {
}

private:

int i = 0, j = 0;
};

////////////////////////////////////////////////
//To calculate with WHORP it will be more fluent
//if we use this initialize_weights part!
// here and above at weights check the factors
void initialize_weights() {
double weightmax = 0;
for (int i = 0; i <7; i++) {
max_bnd[i] = (up_bnd[i + 1] > -lw_bnd[i + 1])
? up_bnd[i + 1] : -lw_bnd[i + 1];
weightmax += weight[i];
}
max_bnd[2] = Pi;
max_bnd[3] = Pi;
weightmax += weight[7];
for (int i = 0; i < 6; i++){
weight[i] = weight[i]/max_bnd[i]/weightmax;
}
weight[6] = weight[6] / max_bnd[6]/ weightmax;
double E2max = m1*g*a1 + m2*g*l1 + m2*g*a2;
// the energy of pendulum
weight[7] = weight[7] / E2max / E2max/ weightmax;
}

int main(int argv, char* argc[]) {

initialize_weights();

TWparameter twparameter6("TransWORHP.xml");
twparameter6.Arguments(argv, argc);
twparameter6.NDIS = 6;
TWfolder folder_simu6(&twparameter6, 0);
TWfolder folder_real6(&twparameter6, 0);
TWfolder folder_optim6(&twparameter6, 0);
TWparameter twparameter5("TransWORHP.xml");
twparameter5.Arguments(argv, argc);
twparameter5.NDIS = 5;
TWfolder folder_simu5(&twparameter5, 0);
TWfolder folder_real5(&twparameter5, 0);
TWfolder folder_optim5(&twparameter5, 0);
TWparameter twparameter4("TransWORHP.xml");
twparameter4.Arguments(argv, argc);
twparameter4.NDIS = 4;
TWfolder folder_simu4(&twparameter4, 0);
TWfolder folder_real4(&twparameter4, 0);
TWfolder folder_optim4(&twparameter4, 0);
TWparameter twparameter3("TransWORHP.xml");
twparameter3.Arguments(argv, argc);
twparameter3.NDIS = 3;
TWfolder folder_simu3(&twparameter3, 0);
TWfolder folder_real3(&twparameter3, 0);
```

```
TWfolder folder_optim3(&twparameter3, 0);
TWparameter twparameter2("TransWORHP.xml");
twparameter2.Arguments(argv, argc);
twparameter2.NDIS = 2;
TWfolder folder_simu2(&twparameter2, 0);
TWfolder folder_real2(&twparameter2, 0);
TWfolder folder_optim2(&twparameter2, 0);
TWparameter twparameter1("TransWORHP.xml");
twparameter1.Arguments(argv, argc);
twparameter1.NDIS = 1;
TWfolder folder_simu1(&twparameter1, 0);
TWfolder folder_real1(&twparameter1, 0);
TWfolder folder_optim1(&twparameter1, 0);
TWparameter twparameter0("TransWORHP.xml");
twparameter0.Arguments(argv, argc);
twparameter0.NDIS = 0;
TWfolder folder_simu0(&twparameter0, 0);
TWfolder folder_real0(&twparameter0, 0);
TWfolder folder_optim0(&twparameter0, 0);

MPC_Doppelpendel_optim ph_optim6(6);
MPC_Doppelpendel_optim ph_simu6(6);
MPC_Doppelpendel_real ph_real6(6);
MPC_Doppelpendel_optim ph_optim5(5);
MPC_Doppelpendel_optim ph_simu5(5);
MPC_Doppelpendel_real ph_real5(5);
MPC_Doppelpendel_optim ph_optim4(4);
MPC_Doppelpendel_optim ph_simu4(4);
MPC_Doppelpendel_real ph_real4(4);
MPC_Doppelpendel_optim ph_optim3(3);
MPC_Doppelpendel_optim ph_simu3(3);
MPC_Doppelpendel_real ph_real3(3);
MPC_Doppelpendel_optim ph_optim2(2);
MPC_Doppelpendel_optim ph_simu2(2);
MPC_Doppelpendel_real ph_real2(2);
MPC_Doppelpendel_optim ph_optim1(1);
MPC_Doppelpendel_optim ph_simu1(1);
MPC_Doppelpendel_real ph_real1(1);
MPC_Doppelpendel_optim ph_optim0(0);
MPC_Doppelpendel_optim ph_simu0(0);
MPC_Doppelpendel_real ph_real0(0);

folder_optim6.Add(&ph_optim6);
folder_optim5.Add(&ph_optim5);
folder_optim4.Add(&ph_optim4);
folder_optim3.Add(&ph_optim3);
folder_optim2.Add(&ph_optim2);
folder_optim1.Add(&ph_optim1);
folder_optim0.Add(&ph_optim0);
folder_real6.Add(&ph_real6);
folder_real5.Add(&ph_real5);
folder_real4.Add(&ph_real4);
folder_real3.Add(&ph_real3);
folder_real2.Add(&ph_real2);
folder_real1.Add(&ph_real1);
folder_real0.Add(&ph_real0);
folder_simu6.Add(&ph_simu6);
folder_simu5.Add(&ph_simu5);
folder_simu4.Add(&ph_simu4);
folder_simu3.Add(&ph_simu3);
folder_simu2.Add(&ph_simu2);
folder_simu1.Add(&ph_simu1);
folder_simu0.Add(&ph_simu0);
MPC_Doppelpendel_optim* p_ph_optim[6];
MPC_Doppelpendel_optim* p_ph_simu[6];
MPC_Doppelpendel_real* p_ph_real[6];
TWfolder* p_folder_optim[6];
TWfolder* p_folder_simu[6];
TWfolder* p_folder_real[6];
TWparameter* p_twparameter[6];

p_ph_optim[5] = &ph_optim6;
p_ph_optim[4] = &ph_optim5;
p_ph_optim[3] = &ph_optim4;
p_ph_optim[2] = &ph_optim3;
p_ph_optim[1] = &ph_optim2;
```

```
p_ph_optim[0] = &ph_optim1;
p_ph_simu[5] = &ph_simu6;
p_ph_simu[4] = &ph_simu5;
p_ph_simu[3] = &ph_simu4;
p_ph_simu[2] = &ph_simu3;
p_ph_simu[1] = &ph_simu2;
p_ph_simu[0] = &ph_simu1;

p_ph_real[5] = &ph_real6;
p_ph_real[4] = &ph_real5;
p_ph_real[3] = &ph_real4;
p_ph_real[2] = &ph_real3;
p_ph_real[1] = &ph_real2;
p_ph_real[0] = &ph_real1;

p_folder_optim[5] = &folder_optim6;
p_folder_optim[4] = &folder_optim5;
p_folder_optim[3] = &folder_optim4;
p_folder_optim[2] = &folder_optim3;
p_folder_optim[1] = &folder_optim2;
p_folder_optim[0] = &folder_optim1;

p_folder_simu[5] = &folder_simu6;
p_folder_simu[4] = &folder_simu5;
p_folder_simu[3] = &folder_simu4;
p_folder_simu[2] = &folder_simu3;
p_folder_simu[1] = &folder_simu2;
p_folder_simu[0] = &folder_simu1;

p_folder_real[5] = &folder_real6;
p_folder_real[4] = &folder_real5;
p_folder_real[3] = &folder_real4;
p_folder_real[2] = &folder_real3;
p_folder_real[1] = &folder_real2;
p_folder_real[0] = &folder_real1;

p_twparameter[5] = &twparameter6;
p_twparameter[4] = &twparameter5;
p_twparameter[3] = &twparameter4;
p_twparameter[2] = &twparameter3;
p_twparameter[1] = &twparameter2;
p_twparameter[0] = &twparameter1;

ofstream mpcfile("mpc.dat");
ofstream mpcfile2("mpc2.dat");
ofstream mpcfile3("mpc3.dat");
ofstream mpcfile4("mpc4.dat");

int state,i, steps, thrsh, thrsh2;
int temp_n_dis, temp_n_ode, n_dis[3],dummy;
double base_time_optim, base_time_real, start_time;
double calc_time_current, calc_time_estim, calc_time_last
       ,sync_time_current,sync_time_estim;
double calc_time_estim_next, compare_time
       ,temp_time[3], temp_u[6], energy_offset;
double y_interp[2];
try
{
        Viewer *viewer = nullptr;
        if (p_twparameter[5]->PLOT) {
        std::cout << std::endl;
        viewer = new Viewer(p_twparameter[5]);
}

for (temp_n_dis = 2; temp_n_dis < 7; temp_n_dis++) {
   for (i = 0; i < temp_n_dis; i++){
        p_ph_simu[temp_n_dis - 1]->T[i] = dis_times[i];
        p_ph_real[temp_n_dis - 1]->T[i] = dis_times[i];
        p_folder_simu[temp_n_dis - 1]->Init();
        p_folder_real[temp_n_dis - 1]->Init();
  }
}
for (temp_n_dis = 2; temp_n_dis < 7; temp_n_dis++) {
   for (i = 0; i < temp_n_dis; i++){
        p_ph_optim[temp_n_dis - 1]->T[i] = dis_times[i];
        p_folder_optim[temp_n_dis - 1]->Init();
```

```
    }
}
n_dis[0] = 6; //optim
n_dis[1] = 6; //simu
n_dis[2] = 6; //real
//only serves slimmer code
p_ph_optim[0] = p_ph_optim[n_dis[0] - 1];

//lin Interp. others, MATLAB course, then save, and import,
//also for control
for (i = 0; i < n_dis[0]; i++) {
     control[i] = 0.0;
  for (state = 0; state < p_ph_optim[0]->n_ode; state++)
  //n_ode number of states
  states[i][state] = (start[state] * (dis_times[n_dis[0]-1]
                    - dis_times[i])+ziel[state]*dis_times[i])
                   / dis_times[n_dis[0] - 1];
}

p_folder_optim[5]->Init(viewer);
p_folder_optim[n_dis[0] - 1]->Loop(0,0);
//loop(wait, terminate).
calc_time_estim    = ext_estim_time;
sync_time_estim    = calc_time_estim;
base_time_optim    = 0.0;
base_time_real     = 0.0;
calc_time_current = 0.1;

for (int K = 0; K < 80; K++) { //INFINITY

start_time = 0.001*(double)GetTickCount();
// integrate starting detection from initial state and control
// determine the length of the time vector
thrsh =1;
while (dis_times[thrsh] < sync_time_estim - 0.0001)
thrsh++;
// store new time vector lines and pointers to a suitable
simulation object
n_dis[1] = thrsh;
p_ph_simu[0] = p_ph_simu[n_dis[1]];
// only serves slimmer code
// set initial state
for (state = 0; state < p_ph_simu[0]->n_ode; state++)
  p_ph_simu[0]->X[p_ph_simu[0]->x_index(0, state)]
               = start[state];

// Set control
for (i = 0; i < thrsh; i++)
p_ph_simu[0]->X[p_ph_simu[0]->u_index(i, 0)]
             = p_ph_optim[0]->u(i, 0);

if (fabs(dis_times[thrsh] - sync_time_estim) > 0.0001)
{
 //linear interpolation
 y_interp[0] = p_ph_optim[0]->u(thrsh - 1, 0);
 y_interp[1] = p_ph_optim[0]->u(thrsh, 0);
 p_ph_simu[0]->X[p_ph_simu[0]->u_index(thrsh, 0)]
 =interp1(&dis_times[thrsh-1], y_interp, sync_time_estim);
 p_ph_simu[0]->T[thrsh] = sync_time_estim;
}
else
 p_ph_simu[0]->X[p_ph_simu[0]->u_index(thrsh, 0)]
              = p_ph_optim[0]->u(thrsh, 0);
//Integrate system model
steps = p_ph_simu[0]->Integrate(p_twparameter[n_dis[1]]
      ->butchertableau);
// Time for start-up
calc_time_current += 0.001*(double)GetTickCount()
                   - start_time;
//Hold the controller for the real system
for (i = 0; i < n_dis[0]; i++)
  temp_u[i] = p_ph_optim[0]->u(i, 0);

// Textausgabe
mpcfile << "Unchangeable control signals:" << endl;
for (i = 0; i <= thrsh; ++i)
```

```
mpcfile << base_time_optim + p_ph_simu[0]->T[i] << "s:"
        << p_ph_simu[0]->u(i, 0) << endl;

mpcfile << endl << "Expected States:";
for (i = 0; i <= thrsh; i++) {
mpcfile << endl << base_time_optim + p_ph_simu[0]->T[i]<<"s:";
mpcfile3 << base_time_optim + p_ph_simu[0]->T[i] << ","
         << p_ph_simu[0]->u(i, 0);

for (state = 0; state <p_ph_simu[0]->n_ode; state++) {
mpcfile << p_ph_simu[0]->x(i, state) << ",";
mpcfile3 << "," << p_ph_simu[0]->x(i, state);
//simulation part of x(i,state)
}
if ((i == 0) && (base_time_real == base_time_optim))
mpcfile3 << ",1";
else
mpcfile3 << ",0";

mpcfile3 << endl;
}
mpcfile << endl << endl;


start_time = 0.001*(double)GetTickCount();

// Make changes to the bill
p_ph_simu[0]->T[thrsh] = dis_times[thrsh];

// optimization

//Optimization Start values set to simulated  result point
for (state = 0; state < p_ph_optim[0]->n_ode; state++)
start[state] = p_ph_simu[0]->x(thrsh, state);

// take the last run as a start-up optimization
// Determine energy offset between last planned trajectory
and  result value

temp_n_ode = p_ph_optim[0]->n_ode - 1; //Energie-Index
y_interp[0] = p_ph_optim[0]->x(thrsh - 1, temp_n_ode);
y_interp[1] = p_ph_optim[0]->x(thrsh, temp_n_ode);
energy_offset = start[temp_n_ode]-interp1(
&dis_times[thrsh - 1], y_interp, sync_time_estim);

// take delivery point from Simu
control[0] = p_ph_simu[0]->u(thrsh, 0);
for (state = 0; state < p_ph_optim[0]->n_ode; state++)
states[0][state] = start[state];

compare_time = sync_time_estim + dis_times[1];
for (i = 1; compare_time < dis_times[n_dis[0] - 1]; i++) {
while (dis_times[thrsh] < compare_time) thrsh++;
temp_time[0] = compare_time - p_ph_optim[0]->T[thrsh - 1];
temp_time[1] = p_ph_optim[0]->T[thrsh] - compare_time;
temp_time[2] = p_ph_optim[0]->T[thrsh] - p_ph_optim[0]
              ->T[thrsh - 1];
control[i] = (p_ph_optim[0]->u(thrsh, 0)*temp_time[0] +
 p_ph_optim[0]->u(thrsh - 1, 0)*temp_time[1]) / temp_time[2];

for (state = 0; state < p_ph_optim[0]->n_ode; state++) {
states[i][state] = (p_ph_optim[0]->x(thrsh, state)
  *temp_time[0] +p_ph_optim[0]->x(thrsh - 1, state)
  *temp_time[1]) / temp_time[2];
}
states[i][temp_n_ode] += energy_offset;
// Energy should start at 0 and otherwise develop the same

compare_time = sync_time_estim + dis_times[i + 1];
}

// above consider less points in time
temp_n_dis = 3 + (int)fabs(wrap2pi(start[2]));
temp_n_dis = (temp_n_dis > 3) ? temp_n_dis : 4;
if ((temp_n_dis < 4) || (temp_n_dis > 6)) {
mpcfile << base_time_real << "s:_Too_many_or_too_few_elements
```

```
in_the_time_vector!_\_n";
exit(7);
}

//ggf. Longer start-up inspection possible
for (; i < temp_n_dis; i++) {
control[i] = control[i - 1];
for (state = 0; state < p_ph_optim[0]->n_ode; state++) {
states[i][state] = states[i - 1][state];
}
}

// store new time vectors and pointers to matching
// optimization object
n_dis[0] = temp_n_dis;
p_ph_optim[0] = p_ph_optim[n_dis[0] - 1];
//only serves slim code
//Perform optimization run
p_folder_optim[n_dis[0] - 1]->Reinit();
p_folder_optim[n_dis[0] - 1]->Loop(0, 0);
//Loop (wait,terminate).
// Time for start-up + optimization
calc_time_current += 0.001*(double)GetTickCount()-start_time;

// lab computer
mpcfile4 << p_ph_optim[0]->n_dis << ","
<< calc_time_current << endl;

calc_time_current = 0.1;

//Estimate next time
calc_time_estim_next = (calc_time_estim + calc_time_current)
                       * 0.5;
calc_time_estim_next = max(calc_time_estim_next
                       , calc_time_estim * 0.5);
calc_time_estim_next = min(calc_time_estim_next
                       , calc_time_estim * 1.5);

// text output
mpcfile << "Optimal_control:" << endl;
for (i = 0; i < p_ph_optim[0]->n_dis; i++)
 mpcfile << base_time_optim + calc_time_estim
   + p_ph_optim[0]->T[i] << "s:" <<p_ph_optim[0]
   ->u(i, 0) << endl;
mpcfile << endl << "States_aimed_at:";
for (i = 0; i < p_ph_optim[0]->n_dis; i++) {
 mpcfile << endl << base_time_optim + calc_time_estim +
 p_ph_optim[0]-> T[i] << "s:";
mpcfile3 << base_time_optim + calc_time_estim + p_ph_optim[0]
 ->T[i]<< "," << p_ph_optim[0]->u(i, 0);
for (state = 0; state < p_ph_optim[0]->n_ode; state++) {
 mpcfile <<    p_ph_optim[0]->x(i, state) << ",";
//  optimal part of x(i,state)
mpcfile3 << "," << p_ph_optim[0]->x(i, state);
}
if (i == 0)
mpcfile3 << ",2";
else
mpcfile3 << ",0";
mpcfile3 << endl;
}
mpcfile << endl << endl;

//mpcfile4 << calc_time_current << endl;
//sample_time += calc_time_current;
//Synchronize times
sync_time_current = calc_time_current + base_time_optim
                  - base_time_real;
sync_time_estim   = calc_time_estim   + base_time_optim
                  - base_time_real;

if (sync_time_estim > dis_times[5]) {
mpcfile << base_time_real + sync_time_estim <<
"s:_last_optim_run_took_too_long!\n";
exit(4);
}
```

```
// If calculation took less than expected
// the sequence will not be sent until later,
// so it fits the condition. The next
// Calculation can still start when the distance
// justifies the effort

if (sync_time_current < sync_time_estim - pre_calc_time)
{
calc_time_last = calc_time_current; //caching
// unter 0 ?
sync_time_current = max(0.001, sync_time_current);
// more than the hip shorter than imagined?
if (sync_time_current < sync_time_estim + 0.001
                      - calc_time_current)
mpcfile4 << sync_time_current;
sync_time_current = max(sync_time_estim + 0.001
 - calc_time_current, sync_time_current);
// integrate real system
//mpcfile4 << sync_time_estim - sync_time_current << endl;
// Determine the length of the time vector
thrsh2 = 0;
while (dis_times[thrsh2] < sync_time_current) thrsh2++;
thrsh = thrsh2;
while (dis_times[thrsh] < sync_time_estim) thrsh++;
if (thrsh >= n_dis[0]) {
// Then sync_time_estim is within the max. 6 time steps,
// but not in the n_dis [0] time steps, those for the last one
// optimization were used.
mpcfile << base_time_real + sync_time_estim << "s:
last optimization run too short, or calculation now too long
 => control sequence interrupted! \n";
exit(6);
}

// store new time vector lines and pointers to a suitable
 simulation object

n_dis[2] = thrsh + 2;
p_ph_real[0] = p_ph_real[n_dis[2] - 1];
// only serves slim code
//reinitialize the real system
for (state = 0; state < p_ph_real[0]->n_ode; state++)
      p_ph_real[0]->X[p_ph_real[0]->x_index(0, state)]
                  = start_real[state];

// Set control, first without special points
for (i = 0; i < n_dis[2]; i++)
p_ph_real[0]->X[p_ph_real[0]->u_index(i, 0)] = temp_u[i];
//linear interpolation  result point
y_interp[0] = p_ph_real[0]->u(thrsh - 1, 0);
y_interp[1] = p_ph_real[0]->u(thrsh, 0);
p_ph_real[0]->X[p_ph_real[0]->u_index(thrsh + 1, 0)]
=interp1(&dis_times[thrsh-1], y_interp, sync_time_estim);
p_ph_real[0]->T[thrsh + 1] = sync_time_estim;
//Erase values in between
for (i = thrsh; i > thrsh2; i--) {
p_ph_real[0]->X[p_ph_real[0]->u_index(i, 0)]= p_ph_real[0]
            ->u(i - 1,0);
p_ph_real[0]->T[i] = p_ph_real[0]->T[i - 1];
}
//linear interpolation measuring point
y_interp[0] = p_ph_real[0]->u(thrsh2 - 1, 0);
y_interp[1] = p_ph_real[0]->u(thrsh2, 0);
p_ph_real[0]->X[p_ph_real[0]->u_index(thrsh2, 0)]
=interp1(&dis_times[thrsh2-1], y_interp, sync_time_current);
p_ph_real[0]->T[thrsh2] = sync_time_current;

//Integrate system
steps = p_ph_real[0]->Integrate(p_twparameter
[n_dis[2] - 1]->butchertableau);

// text output
mpcfile << "Control signals done:" << endl;
```

```
for (i = 0; i < thrsh; i++) {
  mpcfile << base_time_real + p_ph_real[0]->T[i] << "s:"
        << p_ph_real[0]->u(i, 0) << endl;
}
mpcfile << endl << "Real states: ";
for (i = 0; i <= thrsh; i++) {
mpcfile << endl << base_time_real + p_ph_real[0]->T[i]<<"s:";
mpcfile2 << base_time_real + p_ph_real[0]->T[i] << ","
         << p_ph_real[0]->u(i, 0) << ",";

for (state = 0; state < p_ph_real[0]->n_ode; state++) {
mpcfile << p_ph_real[0]->x(i, state) << ",";
mpcfile2 << p_ph_real[0]->x(i, state) << ",";
}
mpcfile2 << "1" << endl;
}
mpcfile << endl << endl;

// undo changes
for (i = thrsh2; i < n_dis[2]; i++) {
p_ph_real[0]->X[p_ph_real[0]->u_index(i, 0)] = temp_u[i];
p_ph_real[0]->T[i] = dis_times[i];
}

// Fest Hold down the result point for the real system
for (state = 0; state < p_ph_real[0]->n_ode - 1; state++) {
start_real[state] = p_ph_real[0]->x(thrsh, state);
}
start_real[p_ph_real[0]->n_ode - 1] = 0.0;
//Start energy again at 0
// Start detection from initial state and integrate control
start_time = 0.001*(double)GetTickCount();

//Determine the length of the time vector
thrsh = 1;
while (dis_times[thrsh] < sync_time_current) thrsh++;
i = 0;
while (dis_times[thrsh + i] < sync_time_estim) i++;
//store new time vector lines and pointers to
//matching simulation objects
n_dis[1] = i + 2;
p_ph_simu[0] = p_ph_simu[n_dis[1] - 1];
//only serves slimmer code

// Initial state Determine start detection
p_ph_simu[0]->X[p_ph_simu[0]->x_index(0,0)]=p_ph_real[0]
->x(thrsh2,0);
p_ph_simu[0]->X[p_ph_simu[0]->x_index(0,1)]=p_ph_real[0]
->x(thrsh2,1);
p_ph_simu[0]->X[p_ph_simu[0]->x_index(0,2)]=p_ph_real[0]
->x(thrsh2,3);
p_ph_simu[0]->X[p_ph_simu[0]->x_index(0,3)]=p_ph_real[0]
->x(thrsh2,4);
p_ph_simu[0]->X[p_ph_simu[0]->x_index(0,4)]=0.0;
//Start energy again at 0

// set control
// linear interpolation for the measuring point
y_interp[0] = p_ph_real[0]->u(thrsh - 1, 0);
y_interp[1] = p_ph_real[0]->u(thrsh, 0);
p_ph_simu[0]->X[p_ph_simu[0]->u_index(0, 0)]
 =interp1(&dis_times[thrsh-1], y_interp, sync_time_current);
//Values in between
for (thrsh2 = thrsh; thrsh2 < thrsh+n_dis[1]-2;thrsh2++){
i = thrsh2 - thrsh + 1;
p_ph_simu[0]->X[p_ph_simu[0]->u_index(i,0)]
= p_ph_real[0]->u(thrsh2,0);
p_ph_simu[0]->T[i] = dis_times[thrsh2]-sync_time_current;
}
//linear interpolation for the result point
y_interp[0] = p_ph_real[0]->u(thrsh2 - 1, 0);
y_interp[1] = p_ph_real[0]->u(thrsh2, 0);
p_ph_simu[0]->X[p_ph_simu[0]->u_index(n_dis[1] - 1, 0)]
=interp1(&dis_times[thrsh2-1], y_interp, sync_time_estim);
p_ph_simu[0]->T[n_dis[1]-1]=sync_time_estim-sync_time_current;
```

```
// Integrate system
steps = p_ph_simu[0]->Integrate
(p_twparameter[n_dis[1] - 1]->butchertableau);

//Integral as a new start value
for (state = 0; state < p_ph_simu[0]->n_ode; state++)
start[state] = p_ph_simu[0]->x(thrsh, state);
calc_time_current =0.001*(double)GetTickCount()-start_time;

//text output
mpcfile << "Unchangeable control signals:" << endl;
for (i = 0; i < thrsh; i++)
mpcfile << base_time_real + p_ph_simu[0]->T[i] <<"s:"
        << p_ph_simu[0]->u(i, 0) << endl;

mpcfile << endl << "Expected States:";
for (i = 0; i <= thrsh; i++) {
  mpcfile << endl << base_time_optim+p_ph_simu[0]->T[i]<<"s:";
  mpcfile3 << base_time_optim + p_ph_simu[0]->T[i] << ","
           << p_ph_simu[0]->u(i, 0);
  for (state = 0; state < p_ph_simu[0]->n_ode; state++) {
    mpcfile << p_ph_simu[0]->x(i, state) << ",";
    mpcfile3 << "," << p_ph_simu[0]->x(i, state);
    }
if (i == 0)
mpcfile3 << ",3";
else
mpcfile3 << ",0";
mpcfile3 << endl;
}
mpcfile << endl << endl;
// Undo changes
for (i = 0; i < n_dis[1]; i++)
p_ph_simu[0]->T[i] = dis_times[i];

// Continue timestamp
base_time_real += sync_time_estim;
base_time_optim += calc_time_last;
}

else
// at less than pre_calc_time remaining time, exactly hit or
// too long calculation of control sequence
{
// integrate real system

// transfer time
sync_time_current = max(sync_time_current, sync_time_estim);

// Determine the length of the time vector
thrsh = 1;
while ( dis_times[thrsh] < sync_time_current - 0.0001 )
thrsh++;
//Check if optimization vector was sufficient
if (thrsh >= n_dis[0]) {
// Then sync_time_estim is within the max. 6 time steps,
// but not in the n_dis [0] time steps, those for the last one
// optimization were used.
mpcfile << base_time_real + sync_time_estim << "s: last optimization
 run too short, or calculation now too long => control sequence interrupted! \n";
exit(6);
}

// store new time vector lines and pointers to
// a suitable simulation object
n_dis[2] = thrsh;
p_ph_real[0] = p_ph_real[n_dis[2]];
// only serves slimmer code
// reinitialize the real system
for (state = 0; state < p_ph_real[0]->n_ode; state++)
p_ph_real[0]->X[p_ph_real[0]->x_index(0, state)]= start_real[state];
// Set control
for (i = 0; i < n_dis[2]; i++)
p_ph_real[0]->X[p_ph_real[0]->u_index(i, 0)] = temp_u[i];
if (fabs(dis_times[thrsh] - sync_time_current) > 0.0001)
{//linear interpolation
```

```
y_interp[0] = p_ph_real[0]->u(thrsh - 1, 0);
y_interp[1] = p_ph_real[0]->u(thrsh, 0);
p_ph_real[0]->X[p_ph_real[0]->u_index(thrsh, 0)] =
interp1(&dis_times[thrsh - 1], y_interp, sync_time_current);
p_ph_real[0]->T[thrsh] = sync_time_current;
}
else
p_ph_real[0]->X[p_ph_real[0]->u_index(thrsh, 0)] = temp_u[thrsh];

//Integrate system
steps = p_ph_real[0]->Integrate(p_twparameter[n_dis[2]]->butchertableau);
// text output
mpcfile << "Detailed control signals:" << endl;
for (i = 0; i < thrsh; i++) {
mpcfile << base_time_real + p_ph_real[0]->T[i] <<"s:"<< p_ph_real[0]->u(i,0)<<endl;
}
mpcfile << endl << "Real condition:";
for (i = 0; i <= thrsh; i++) {
mpcfile << endl << base_time_real + p_ph_real[0]->T[i]<<"s:";
mpcfile2 << base_time_real + p_ph_real[0]->T[i]<<","<< p_ph_real[0]->u(i, 0)<<",";
for (state = 0; state < p_ph_real[0]->n_ode; state++) {
mpcfile  << p_ph_real[0]->x(i, state) << ",";
mpcfile2 << p_ph_real[0]->x(i, state) << ",";
}
mpcfile2 << "0" << endl;
}
mpcfile << endl << endl;

// Undo changes
p_ph_real[0]->T[thrsh] = dis_times[thrsh];

// measurement signal after start estimation + optimization as new start value
//  Hold down the result of the real system
for (state = 0; state < p_ph_real[0]->n_ode - 1; state++) {
start[state] = p_ph_real[0]->x(thrsh, state);
start_real[state] = start[state];
}
//Start energy again at 0
start[p_ph_simu[0]->n_ode - 1] = 0.0;
start_real[p_ph_real[0]->n_ode - 1] = 0.0;


// Continue total time
base_time_real += sync_time_current;
base_time_optim = base_time_real;

}
calc_time_current = 0.1;
// Average expected time for start-up + optimization
calc_time_estim = calc_time_estim_next;
}
mpcfile.close();
mpcfile2.close();
mpcfile3.close();
mpcfile4.close();
}
catch (int e)
{
printf("An error accured: error = %d\n", e);
printf("Application will stop!");
printf("PRESS ANY KEY!\r\n");
while (!_kbhit());
_getch();
exit(1);
}

while (!_kbhit());
_getch();
return 0;
}
```

D. Visualization code in Matlab

D.1. Plot Shun1

```
%% initialisation
framerate = 50;
mpc = importfile('Shun1.dat');
%% showing the input, states and deduced values like input "power"
%%or pendulum energy over time
figure(3);
subplot(4,2,1)
plot(mpc(:,1),mpc(:,2));
legend('control value');
subplot(4,2,2)
plot(mpc(:,1),mpc(:,[3 4]));
legend('p1 velocity [rad/s]','p2 velocity [rad/s]');
subplot(4,2,3)
plot(mpc(:,1),mpc(:,[5 6]));
legend('p1 position [rad]','p2 position [rad]');
subplot(4,2,4)
plot(mpc(:,1),mpc(:,8));
legend('cart velocity [m/s]');
subplot(4,2,5)
plot(mpc(:,1),mpc(:,7));
legend('cart position [m]');
subplot(4,2,6)
plot(mpc(:,1),mpc(:,9));
legend('intput energy [J]');
mtit('input, states, deduced values');

%% showing the objective, both total and partial, over time
ziel   = [0, 0,  0,    pi,   0,    0,    0];
weight = [0, 0,  1.0,  1.0, 0.1,   0,  0.3, 1];
% velocity 1,  velocity 2, angle 1, angle 2, cart pos
%, cart speed, input power, energy p1
up_bnd = [ 1.2,  25,   25,  1e20, 1e20,  0.38,  1.2,0.5];
lw_bnd = [-1.2, -25,  -25, -1e20,-1e20,-0.38, -1.2, 0];
% bounds have input bound at first index, and no bound
%for energy p1 => ok!
m1 = 0.162;      % Mass pendulum 1 [kg]
m2 = 0.203;
a2 = 0.157;
a1 = 0.085;      % Half pendulum neck 1 [m]
l1 = 0.17;  % pendulum neck 1 [m]
g  = 9.81;       % gravitation [m/s^2]
gl = 9.81 * l1;
J_eff = (m1 * l1 * l1) / 3;
max_bnd = max(up_bnd(2:end),-lw_bnd(2:end));
max_bnd(3:4) = pi;
weightmax = sum(weight);
weight(1:6) = weight(1:6)./(max_bnd(1:6).^2) / weightmax;
weight(7) = weight(7) / max_bnd(7) / weightmax;
weight(8) = weight(8)/(m1*g*a1+m2*g*l1+m2*g*a2)^2 /weightmax;

tmp = mpc(:,8) - ziel(6);
obj = tmp.*tmp * weight(6);
tmp = abs( mod(mpc(:,6) - ziel(4) + pi, pi*2) - pi);
Pendellage2 = tmp.*tmp * weight(4); %angle 2
obj = obj + Pendellage2;

tmp = abs(mod(mpc(:,5) - ziel(3) + pi, pi*2) - pi);
Pendellage1=tmp.*tmp * weight(3);
obj=obj+ Pendellage1;
```

```
tmp = mpc(:,3) - ziel(1);
obj = obj + tmp.*tmp * weight(1);     %velocity 1
tmp = mpc(:,4) - ziel(2);
obj = obj + tmp.*tmp * weight(2);      %velocity 2

Stellenergie = (mpc(:,9)-ziel(7) )* weight(7); %input power
obj = obj + Stellenergie;

tmp = cos(mpc(:,5)) - cos(ziel(3));
tmp = tmp .*(l1*m2*g + g*m1*a1);
tmp = tmp + (cos(mpc(:,6)) - cos(ziel(4)))*a2*m2*g;
Pendelenergie = tmp.*tmp * weight(8);
obj = obj + Pendelenergie;

tmp = mpc(:,7) - ziel(5);
Wagenlage = tmp.*tmp * weight(5);       %carriage position
obj = obj + Wagenlage;
figure(2);
plot(mpc(:,1),[obj,Pendelenergie,Wagenlage,Pendellage1  ,Pendellage2,Stellenergie]);
legend('Total energy','pendulums energy','cart position'
 ,'pendulum 1 position','pendulum 2 position','input power');
title('Total and partial objective');

%% showing and saving the animation
x = mpc(:,[1  7]);
alpha = mpc(:,[1  5]);
alpha2 = mpc(:,[1  6]);

t = mpc(:,1);
I = find(diff(t)==0);
while numel(I)>0
t(I) = t(I)-1e-12;
I = find(diff(t)==0);
end

%tmax = min(6.5,mpc(end,1));
tmax = mpc(end,1);

dt = 1/framerate;
tv = 0:dt:tmax;
alpha =  interp1(t,-alpha(:,2)+pi,tv);
alpha2 = interp1(t,-alpha2(:,2)+pi,tv);
x = interp1(t,x(:,2),tv);

mpc = importfile1('Shun1b.dat');
x_op = mpc(:,7);
alpha_op  = -mpc(:,5)+pi;
alpha2_op = -mpc(:,6)+pi;
t_op = mpc(:,1);
I_start = find(mpc(:,10)==1);
I_op_start = find(mpc(:,10)==2);

v = VideoWriter('Shun1','MPEG-4');
v.FrameRate = framerate*0.6;
open(v);

traj_counter = 1;
opt_counter = 1;
opt_c_max = min(numel(I_op_start),numel(I_start)-1);

try
draw_p; hold all;
F = getframe();
for i = 1:50
writeVideo(v,F);
end

for j = 1:floor(tmax/dt)

h2.Matrix = makehgtform('translate',[x(j) 0 0]);
h3.Matrix = makehgtform('zrotate',alpha(j));
h4.Matrix = makehgtform('zrotate',alpha2(j));

if tv(j) >= t_op(I_start(opt_counter))
I = I_start(opt_counter):I_op_start(opt_counter)-1;
```

```
x_traj = x_op(I) + 0.5*sin(alpha_op(I));
y_traj = -0.5*cos(alpha_op(I));

%h(traj_counter) = plot(x_traj,y_traj,'Color',[1 0 0]);

I = I_op_start(opt_counter):I_start(opt_counter+1)-1;
x_traj = x_op(I) + 0.5*sin(alpha_op(I));
y_traj = -0.5*cos(alpha_op(I));

%h(traj_counter+1) = plot(x_traj,y_traj,'Color',[0 1 0]);

traj_counter = traj_counter + 2;
opt_counter = opt_counter + 1;
end

%{
i = 1;
while traj_counter-2 >= 2*i && h(traj_counter-2*i-1).isvalid
Color = h(traj_counter-2*i-1).Color;
if Color(1) < 0.975
h(traj_counter-2*i-1).Color = h(traj_counter-2*i-1).Color + [0.025 0 0.025];
h(traj_counter-2*i-2).Color = h(traj_counter-2*i-2).Color + [0 0.025 0.025];
else
h(traj_counter-2*i-1).delete;
h(traj_counter-2*i-2).delete;
end
i = i+1;
end
%}

F = getframe();
writeVideo(v,F);
if opt_counter == opt_c_max
break;
end
end
close(v);
```

D.2. Import file

```
function mpc_data = importfile(filename, startRow, endRow)
%    IMPORTFILE2 Import numeric data from a text file as a matrix.
%    SHUN1 = IMPORTFILE2(FILENAME) Reads data from text
%    file FILENAME forthe default selection.
%    SHUN1 = IMPORTFILE2(FILENAME, STARTROW, ENDROW) Reads data
%    from rows STARTROW through ENDROW of text file FILENAME.
%    Example: Shun1 = importfile2('Shun1.dat', 1, 160);
%    See also TEXTSCAN.
%  Auto-generated by MATLAB on 2019/05/06 09:55:38
%% Initialize variables.
delimiter = {',','␣'};
if nargin<=2
startRow = 1;
endRow = inf;
end

%% Format string for each line of text:
%    column1: double (%f)
%        column2: double (%f)
%    column3: double (%f)
%        column4: double (%f)
%    column5: double (%f)
%        column6: double (%f)
%    column7: double (%f)
%        column8: double (%f)
%    column9: double (%f)
%        column10: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%f%f%f%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
```

```
% This call is based on the structure of the file used to
%  generate this code. If an error occurs for a different
%  file, try regenerating the code from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1
, 'Delimiter', delimiter, 'MultipleDelimsAsOne', true,
'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
frewind(fileID);
dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-
startRow(block)+1, 'Delimiter', delimiter, 'MultipleDelimsAsOne',
true, 'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
for col=1:length(dataArray)
dataArray{col} = [dataArray{col};dataArrayBlock{col}];
end
end
%% Close the text file.
fclose(fileID);
%% Post processing for unimportable data.
% No unimportable data rules were applied during
% the import, so no post processing code is included.
% To generate code which works forun importable data, select
% unimportable cells in a file and regenerate the script.
%% Create output variable
mpc_data = [dataArray{1:end-1}];
```

D.3. Draw pendulum

```
figure(1); clf;
axis manual
axis equal;
ylim([-0.7 0.7]);
xlim([-1.0 1.0]);
h = hgtransform;
rectangle('Position',[-0.51,-0.03,0.06,0.06],'Curvature',[1,1]
,'FaceColor',[0.8,0.8,0.8],'Parent',h);
rectangle('Position',[0.45,-0.03,0.06,0.06],'Curvature',[1,1]
,'FaceColor',[0.8,0.8,0.8],'Parent',h);
h2 = rectangle('Position',[-0.48,-0.03,0.96,0.06],'FaceColor'
,[0.8,0.8,0.8],'Parent',h);
h2.LineStyle='none';
h2 = rectangle('Position',[-0.44,0.02,0.88,0.01],'FaceColor','white','Parent',h);
h2.LineStyle='none';
plot([-0.48,-0.44,-0.44,0.44,0.44,0.48],[0.03,0.03,0.02,0.02,
0.03,0.03],'k','Parent',h);
plot([-0.48,0.48],[-0.03,-0.03],'k','Parent',h);
plot(-0.48,0,'k.','LineWidth',3,'Parent',h);
plot(0.48,0,'k.','LineWidth',3,'Parent',h);
h.Matrix = makehgtform('translate',[0 -0.04 0]);

h2 = hgtransform;
rectangle('Position',[-0.05,-0.02,0.1,0.04],'FaceColor',[0.8,0.8,0.8],'Parent',h2);

h3 = hgtransform('Parent',h2);
rectangle('Position',[-0.02,-0.15,0.04,0.17],'Curvature'
,[1,0.2],'FaceColor',[0.95,0.95,0.95],'Parent',h3);
plot(0,0,'k.','LineWidth',3,'Parent',h2);

h3b = hgtransform('Parent',h3);
h3b.Matrix = makehgtform('translate',[0 -0.15 0]);

h4 = hgtransform('Parent',h3b);
rectangle('Position',[-0.02,-0.294,0.04,0.314],'Curvature',[1,0.15],'FaceColor'
,[0.95,0.95,0.95],'Parent',h4);
plot(0,0,'k.','LineWidth',3,'Parent',h3b);
```

D.4. Mtit

```
%MTIT           creates a major title in a figure with many axes
%               MTIT
%               - creates a major title above all
%                 axes in a figure
```

```
%                - preserves the stack order of
%                  the axis handles
%SYNTAX
%------------------------------------------------------------
%                P = MTIT(TXT,[OPT1,...,OPTn])
%                P = MTIT(FH,TXT,[OPT1,...,OPTn])
%INPUT
%------------------------------------------------------------
%     FH :       a valid figure handle            [def: gcf]
%    TXT :       title string
% OPT    :       argument
% ---------------------------------------
%   xoff :       +/- displacement along X axis
%   yoff :       +/- displacement along Y axis
%   zoff :       +/- displacement along Z axis
%
%                title modifier pair(s)
% ---------------------------------------
%    TPx :       TVx
%                see: get(text) for possible
%                     parameters/values
%
%OUTPUT
%------------------------------------------------------------
%    par :       parameter structure
%   .pos :       position of surrounding axis
%    .oh :       handle of last used axis
%    .ah :       handle of invisible surrounding axis
%    .th :       handle of main title
%
%EXAMPLE
%------------------------------------------------------------
%       subplot(2,3,[1 3]);              title('PLOT 1');
%       subplot(2,3,4);                  title('PLOT 2');
%       subplot(2,3,5);                  title('PLOT 3');
%       axes('units','inches',...
%            'color',[0 1 .5],...
%            'position',[.5 .5 2 2]);    title('PLOT 41');
%       axes('units','inches',...
%            'color',[0 .5 1],...
%            'position',[3.5 .5 2 2]);   title('PLOT 42');
%       shg;
%       p=mtit('the BIG title',...
%            'fontsize',14,'color',[1 0 0],...
%            'xoff',-.1,'yoff',.025);
% % refine title using its handle <p.th>
%       set(p.th,'edgecolor',.5*[1 1 1]);
% created:
%       us       24-Feb-2003              / R13
% modified:
%       us       24-Feb-2003              / CSSM
%       us       06-Apr-2003              / TMW
%       us       13-Nov-2009 17:38:17
%------------------------------------------------------------------------------------------
function          par=mtit(varargin)
defunit='normalized';
if       nargout
par=[];
end
% check input
if       nargin < 1
help(mfilename);
return;
end
if       isempty(get(0,'currentfigure'))
disp('MTIT> no figure');
return;
end
vl=true(size(varargin));
if       ischar(varargin{1})
vl(1)=false;
figh=gcf;
txt=varargin{1};
elseif   any(ishandle(varargin{1}(:)))
ischar(varargin{2})
vl(1:2)=false;
```

```
figh=varargin{1};
txt=varargin{2};
else
error('MTIT> invalid input');
end
vin=varargin(vl);
[off,vout]=get_off(vin{:});
% find surrounding box
ah=findall(figh,'type','axes');
if      isempty(ah)
disp('MTIT> no axis');
return;
end
oah=ah(1);
ou=get(ah,'units');
set(ah,'units',defunit);
ap=get(ah,'position');
if      iscell(ap)
ap=cell2mat(get(ah,'position'));
end
ap=[min(ap(:,1)),max(ap(:,1)+ap(:,3)),...
min(ap(:,2)),max(ap(:,2)+ap(:,4))];
ap=[ap(1),ap(3),...
ap(2)-ap(1),ap(4)-ap(3)];
% create axis...
xh=axes('position',ap);
% ...and title
th=title(txt,vout{:});
tp=get(th,'position');
set(th,'position',tp+off);
set(xh,'visible','off','hittest','on');
set(th,'visible','on');
% reset original units
ix=find(~strcmpi(ou,defunit));
if      ~isempty(ix)
for     i=ix(:).'
set(ah(i),'units',ou{i});
end
end
% ...and axis' order
uistack(xh,'bottom');
axes(oah);
if      nargout
par.pos=ap;
par.oh=oah;
par.ah=xh;
par.th=th;
end
end
%-------------------------------------------------------------
function        [off,vout]=get_off(varargin)
% search for pairs <.off>/<value>
off=zeros(1,3);
io=0;
for     mode={'xoff','yoff','zoff'};
ix=strcmpi(varargin,mode);
if      any(ix)
io=io+1;
yx=find(ix);
ix(yx+1)=1;
off(1,io)=varargin{yx(end)+1};
varargin=varargin(xor(ix,1));
end
end
vout=varargin;
end
%-------------------------------------------------------------
```

E. List of Figures

F. Bibliography

[1] In: Chimmiri, V. : *Model Predictive Control of Nonlinear Processes*. 2010. – ISBN 978–953–307–102–2

[2] Nava, A. A.: *Control Lyapunov Function based controller for a double inverted pendulum*, University Bremen, master, 2015

[3] Schuettler, J. : *Inverted pendulum*. Bremen: Uni Bremen, Version 2.2.1, 2013. – Laboratory Script

[4] Spong, M. W.: ENERGY BASED CONTROL OF A CLASS OF UNDERACTUATED MECHANICAL SYSTEM. In: *IFAC World Congress* (1996), June

[5] Steinbeis-Forschungszentrum Optimierung, S. u. R.: *Users' Guide to WORHP 1.13*. Version 1.13. Bremen: WORHP developer, March 28 2019

[6] Universitaet Bremen (Hrsg.): *TransWORHP*. Bremen: Universitaet Bremen

[7] yue, Y. : *Research on Nonlinear Predictive Control and Its Simulation*, XIAN technology University, master, 2008

[8] Zhong, W. ; Rock*, H. : Energy and Passivity Based Control of the Double Inverted Pendulum on a Cart. In: *Control Systems Magazine, IEEE* (2001), September, S. 896–901