TECHNISCHE UNIVERSITÄT BERLIN
Electrical Engineering and Computer Science
Internet of Things for Smart Buildings
Prof. Dr. Sergio Lucia

Model Predictive Control **SS 20**

P4: System Identification and tracking MPC

System identification is a requirement for all advanced control applications. Two fundamentally different approaches exist here: Modeling a system from first principles and identifying a model from data. In this task you will be working on a data-based black box identification of a simulated system that is given to you in the form of a hidden input-output model. The model comes in the form of an obscured Python class (`state_space_system.py`) and a file to illustrate the API (`test_system.py`). The class method `simulate` can be called with a time and input vector and will return an output vector.

Tasks

Analyze the input-output model. You can import the file in python and simulate it with a time series of inputs. This will return the output sequence for the same time range. You should stimulate the system with an appropriate sequence to obtain useful results for identification. The underlying system is a triple-mass spring system as depicted in Figure 1, where two stepper motors attached to the outermost springs are used to control the system. You are measuring the disc angles in radian.

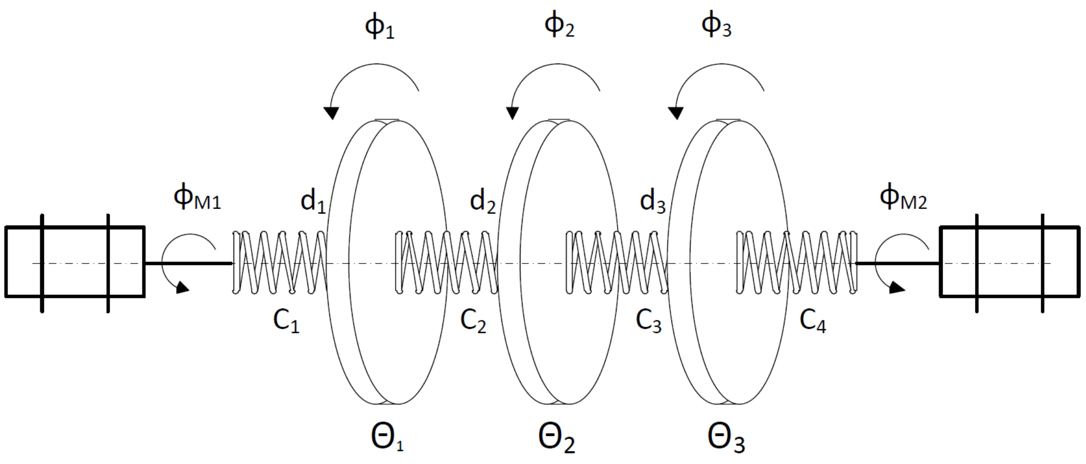

Figure 1: Investigated triple-mass spring system.

Identify linear state-space model

Your first objective is to determine a linear discrete-time state-space representation of the form:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the input and $y \in \mathbb{R}^p$ is the output of the system. The subscript k and $k+1$, respectively, denote a discrete-time instance. Your task is to determine matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$ and $D \in \mathbb{R}^{p\times m}$. There are multiple ways to approach this task. We suggest to investigate, implement and apply the Eigensystem Realization Algorithm (ERA). However, you are free to take other approaches into consideration.

Implement tracking MPC

For the linear discrete-time state-space system in the form of (1) you will be implementing your own Model Predictive Control (MPC) tracking controller. Tracking means, in contrary to regulation, that your system can follow a reference trajectory. You will need to complete the following steps.

First, formulate the optimization problem that allows tracking control of the identified system under consideration of upper and lower bounds on both states and control inputs. You should use a quadratic cost function. Reformulate this optimization problem such that you can **use an arbitrary QP solver**. These solvers are designed for problems of the kind:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^T P x + q^T x \\ \text{subject to:} \quad & Gx \leq h \\ & Ax = b \end{aligned} \tag{2}$$

You task boils down to the determination of the matrices in equation (2) from the original MPC problem. As optimization variables we expect you to only use the control inputs $\mathbf{u} = [u_0, u_1, \ldots, u_{N-1}]$, where N is the prediction horizon. Eliminating the states from the original problem is possible since, given the intial state x_0, (1) can be used to express $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ entirely as a function of $\mathbf{u}$. Eliminating $\mathbf{x}$ from the problem is called "sequential approach". For comparison, you may also implement the "simultaneous approach". In this approach both $\mathbf{x}$ and $\mathbf{u}$ are optimization variables.
Your MPC controller should be able to take future reference trajectories into account. Investigate the performance of your controller for some interesting reference trajectories (e.g. some setpoint changes). Remember to test the controller on the real system (the given system model).

Notes

- You need `scipy` to run our supplied code. Install it with `conda install scipy` or `pip install scipy`.
- `state_space_system.py` is obfuscated such that its content cannot be inspected