General Problem Setting

The *Pendulum-Cart* system defines a classic problem in dynamics and control theory and is widely used as a benchmark for testing control algorithms. A simplified variation of this system is illustrated in Figure 2 where a mathematical pendulum is connected to a pair of only-rolling wheels with one translation degree of freedom. This models the *Wheeled Inverted Pendulum* system. An example for that is the Robot *Suricate* (Figure 1) built by the students at LRS / University of Kaiserslautern.

In the simplified model, the point mass m_p is connected to the cart with the mass m_c via a massless arm with the length L. q_1 is the displacement of the cart and q_2 is the angle of the pendulum. As input, u is assumed to be a force.

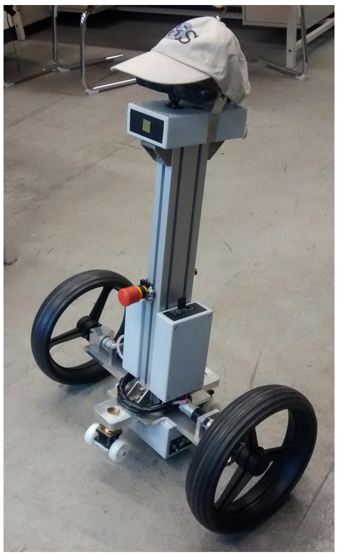
Abbildung 1: Suricate (JIM2)

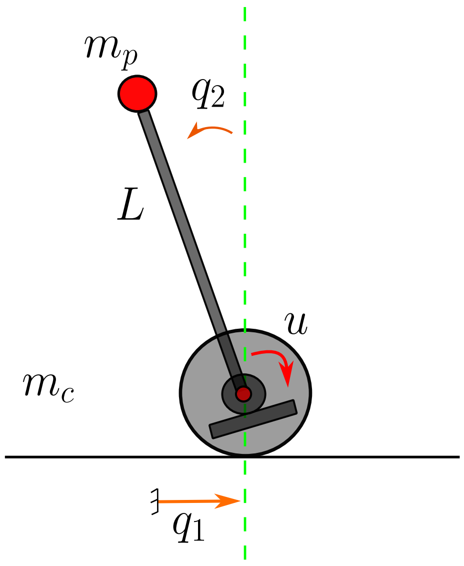

Abbildung 2: Pendulum-Cart System

In our exercises, we will handle the stabilization problem around the upper position of the pendulum. We will only consider the linearised model. The overall objective will be broken down into the following tasks:

- Creating the dynamical system using MATLAB.
- System Analysis, controller design using MATLAB tools and simulation.
- Design of state-space and optimal controllers and simulation.
- Design of state observers and investigating the closed loop including an observer.
- Time-discrete controller and observer design.

The results will be verified by simulation using SIMULINK.

Dynamic Model of the System

The equations of motion for the described system can be derived using different methods such as Lagrange-Formalism or Newton-Euler. The result is in any case a nonlinear system in the following form:

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{g}_q\,\boldsymbol{u}$$

with

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{pmatrix} m_c + m_p & -L\,m_p\,\cos q_2 \\ -L\,m_p\,\cos q_2 & L^2\,m_p \end{pmatrix}$$

$$\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{pmatrix} L\,m_p\,\dot{q}_2^2\,\sin q_2 \\ -L\,g\,m_p\,\sin q_2 \end{pmatrix} + \begin{pmatrix} d_1\,\dot{q}_1 \\ d_2\,\dot{q}_2 \end{pmatrix} \ , \ \boldsymbol{g}_q = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

where d_1 and d_2 are the so-called *damping factors* representing friction in the cart displacement and the joint respectively.

A state-space representation can be derived using the equations above defining $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{pmatrix}$

$$\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x}) + \boldsymbol{G}(\boldsymbol{x})\,u = \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ -\boldsymbol{M}^{-1}(\boldsymbol{q})\,\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \boldsymbol{M}^{-1}(\boldsymbol{q})\,\boldsymbol{g}_q \end{pmatrix} u$$

Linearising about the operation point $q_1^* = q_2^* = 0$ yields the linear state-space equation for the system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{B}\,u \tag{1}$$

with

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{g\,m_p}{m_c} & -\dfrac{d_1}{m_c} & -\dfrac{d_2}{L\,m_c} \\ 0 & \dfrac{g\,(m_c+m_p)}{L\,m_c} & -\dfrac{d_1}{L\,m_c} & -\dfrac{d_2\,(m_c+m_p)}{L^2\,m_c\,m_p} \end{pmatrix} \ , \ \boldsymbol{B} = \begin{pmatrix} 0 \\ 0 \\ \dfrac{1}{m_c} \\ \dfrac{1}{L\,m_c} \end{pmatrix} .$$

We the following values:

$$m_c = 1.5,\ m_p = 0.5,\ g = 9.82,\ L = 1,\ d_1 = d_2 = 0.01$$

Task 1

Within the first task, we analyse the system and compare its different representations.

1. Within a MATLAB script, build the state-space model of the system using the matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and proper matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ by the command `ss`! Suppose the angle q_2 as the system output!
2. Use Matlab commands like `pole`, `eig` and `zero` to analyse the system. Is the system stable? Is it minimal-phase? Draw the zero-pole-map of the system using `pzmap`! Are the results different? Why?
3. Convert the system into the corresponding transfer function using `tf`! How can you do the same using `ss2tf`? Create also the Zero-Pole-Gain representation of the system by converting using `zpk`.
4. For sake of re-usability, write a MATLAB-function that generates the dynamical system in different representations! Use your code from the subtasks above! Your MATLAB-function should take the parameter set m_p, m_c, g, L, d_1 and d_2 as input. Also, the function should have an input to decide whether the position q_1 or the angle q_2 are supposed to be taken as the output of the system. Use for example a string `'MyStr'` as input. The output arguments are the 3 system representations.
5. Use your MATLAB-function now to create the system in 3 representations again and compare them, for instance by their poles and zeros! The output of the SISO system is supposed to be the angle q_2. Are they equal? If there is a difference, how would you explain it?

Task 2

In this task, we consider the results from the task 1 to continue with system analysis and a first controller design. Use your MATLAB-function from the previous exercise to regenerate the system.

1. Reconstruct the system matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ from the state-space system generated by the function from the previous task using `ssdata`! The output of the system is still q_2.
2. Create a SIMULINK model and simulate the system dynamics using an integrator and the matrices as gain-blocks!
3. Use `rlocus` to plot the root locus diagram of the system! Can a P-controller stabilize the closed loop? Why?
4. Open the SISO-TOOL for the system. Try to find a controller using an integrator, a real pole and a complex zero pair to stabilize the system! Consider minimizing the rise time in the step response of the closed loop and the maximum value for the input! Export the controller into the workspace to use it in the simulation!
5. Create the closed loop system using `feedback` and analyse the stability.
6. Add the designed controller to your SIMULINK model and simulate the closed loop! Set the Initial value for the pendulum angle $q_2(t = 0) = 10°$! Does the output convert to zero as desired? Do all of the states convert to zero?

   If you simulate the closed loop long enough (e.g. 20 s or longer), it seams that something is going wrong. How would you explain that? What should change to avoid this problem? What is the practical solution?

Task 3

In the controller designed in the last task, we detected a problem with the total stability of the system. That was, though the input-output mapping of the system seemed to be stabilized using a proper controller and the angle of the pendulum converted to zero, the position of the cart diverged. Within this task, we will investigate the system features to find out the reason for this problem and how to avoid it.

1. Two important features of a dynamical system are controllability and observability. Check these features for the pendulum system using your solutions from the previous tasks!
2. Which difference does it make defining the cart position q_1 as output?
3. Design a state feedback controller $\boldsymbol{K}$ to shift the poles of the closed loop to proper positions!
4. Create the closed loop using the system matrices and $\boldsymbol{K}$ and check if the poles are placed correctly. Investigate the step response of the closed loop!
5. Change the position of the desired closed loop poles and repeat the previous subtask!
6. Design an optimal controller using `lqr` and repeat the previous subtasks!
7. Verify your results by simulation of the closed loop in SIMULINK! Modify the LQR parameter and the desired poles and compare the affect of these entities on the system behaviour! Note the control input!
8. Extend the simulation by a desired position for the cart q_1^*.
9. Test your controller on the "real" non-linear system provided on-line! For which range of initial angles does the controller still stabilize the system? What could possibly make the controller work for a larger (or even every) angle?

Task 4

The previous controller assumes the knowledge and, thus, the measurement of every state. In the praxis, this is hardy achievable. Instead of applying various sensors to measure the states, a state observer is often used to estimate the states using the system outputs only. Within this task we will extend the previous controller by a state observer.

1. For what condition can the entire state of the system be estimated? Check if this condition is fulfilled!
2. Design a state observer using pole placement! Note that the observer should be *faster* than the controller to avoid undesired behaviour in the closed loop!
3. Design an optimal state observer using LQR algorithm!
4. Vary different parameter and compare the simulation results!
5. Test your controller on the ”real” non-linear system provided in on-line! Does everything work properly? If not, what could be the reason? In this case try to tune the controller to make the closed-loop stable!

Task 5

To end up with a real-life control system, we need to use time-discrete controllers. Often, a time-discrete representation of the system is used to consider the whole closed loop a time-discrete system. The control algorithm and the observer are implemented on a digital device and A/D-converters sample the output signals.

1. Convert the system into time-discrete by `c2d`! Use the sample time $T_s = 0.1\,s$!
2. Design a time-discrete controller and a time-discrete observer by pole placement using the equivalent poles to the continuous controller from the previous task! Verify your results by simulation!
3. Design a time-discrete LQR controller and a time-discrete LQR observer using `dlqr` and verify them by simulation! What is the difference to `lqrd`?

Note: To emulate the A/D-converter, use a zero-order-hold block in SIMULINK.