Nils Keller
Jan Schilliger
Yannick Schnider

Predictive Control of a Two-Wheeled Balancing Robot

Lab Practice Control Systems Experiment Script

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology Zurich

February 19, 2024

Contents

1 Introduction

We are happy that you decided to do this laboratory practice. Laboratory practices are an integral part of your education towards an engineering degree. They provide you the opportunity to test your engineering skills on realistic problems, and allow you to design and engineer a solution for a specific realistic problem. On the one hand, you are required to apply the theory learned in class, while on the other, you will face the influence of various real-world complications. Learning how to deal with such challenges and coming up with creative solutions are a vital task of a successful engineer.

Up to now, your studies have focused on "classical" control approaches, such as PID. The objective of this laboratory practice is to give you a foretaste for what else exists in the robotic field of control systems. In this laboratory practice, you will learn how to control a two-wheeled balancing robot (similar to a mini version of a Segway) using a control approach known as Model Predictive Control (MPC) and you will compare it with a PID controller. Specifically, we want you to be able to develop an intuition for the underlying idea of MPC and how it distinguishes from PID, and most importantly, we want to motivate you to continue the path of robotics and control systems. To be clear: neither MPC nor PID are inherently better than the other. In fact, both follow different approaches and are suitable for different areas of application.

Please read the following sections up to the fifth chapter at home and complete the exercises. Write down your answers and bring them with you on the day of the laboratory exercise. Please note that the homework exercises are a mandatory prerequisite to the laboratory exercise.

2 Introduction to Model Predictive Control

As the name suggests, MPC is a model-based control approach, meaning that it explicitly incorporates a model of the system into its formulation. This model is used to predict the influence of an input on the system states and on the outputs. Based on these predictions, the MPC then computes the optimal input.

MPC is a very promising control approach that differs from "classical" approaches in many ways. First, the MPC is a discrete-time control approach, where a control input is computed and held constant between discrete time points, called sampling points. In contrast to PID and other "classical" approaches, the input is not computed using an explicit equation, but rather by solving an optimization problem, as described in figure 1.

Besides the system dynamics, an MPC also consists of a **control objective** that is encoded in the **optimality criterion**. Think of this as the way to tell our controller what our goal is. An example for an optimality criterion is to penalize the deviation between the system states and a reference trajectory, meaning that this deviation should be minimized. This can be combined with a penalty on the input energy, meaning that less control action is encouraged. We want our controller to follow our objective, while obeying certain restrictions. These restrictions are integrated in the MPC as **constraints** that arise due to system limitations, such as maximally applicable inputs, and safety considerations, such as force or speed limitations. Every system is subject to limitations, and it is our intention to design controllers that follow our objective, encoded as the optimality criterion, while strictly obeying all limits. MPC achieves this by explicitly encoding the constraints into the controller formulation. In conclusion, the fact that both the objective and the constraints are present in the optimization problem separates MPC from any classical control approaches and represents the single most important advantage of MPC over classical control approaches.

Solving an optimization problem results in the optimal input trajectory, but one calculation is not enough. Instead, we use **receding horizon control** where we compute the optimal input trajectory, then only apply the first input for one sampling interval. After this interval, we re-calculate the optimal input trajectory starting with current measurements and repeat the process. Receding horizon control predicts system behavior over a **prediction horizon** and finds the optimal input

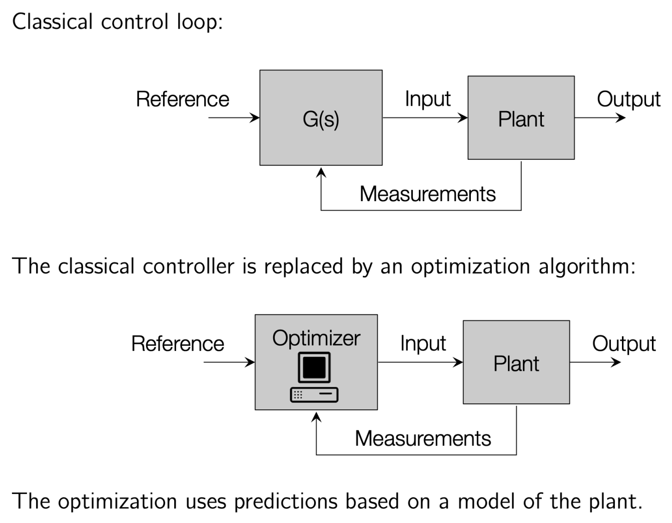

Classical control loop:
Reference
G(s)
Input
Plant
Output
Measurements
The classical controller is replaced by an optimization algorithm:
Reference
Optimizer
Input
Plant
Output
Measurements
The optimization uses predictions based on a model of the plant.

Figure 1: Underlying idea behind a MPC: solving an optimization problem instead of adapting the output based on the current error. From [1].

that minimizes the optimality criterion and meets all constraints, resulting in an optimal input trajectory for each sampling point. The frequency of recomputing the control action depends on the system's speed: slow processes may require recomputation every few minutes, while fast systems may require it at above 1 kHz. This process is summarized in figure 2.

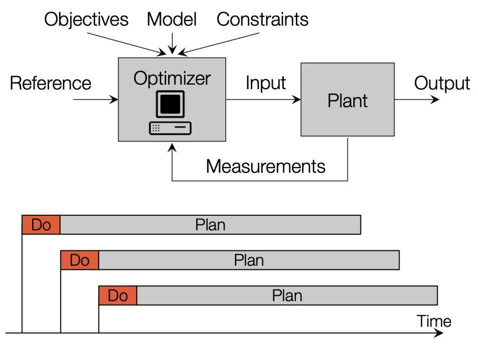


Figure 2: Solving once for the optimal input trajectory is not sufficient. Instead, we must continuously recompute the optimal input trajectory, of which we only apply the first control input. From [1].

Exercise 2.1. *Feed Forward*
List the reasons why it is not possible to just compute an optimal input trajectory for the maneuver that is to be planned (e.g. stabilization or tracking) and apply these inputs to the system.

A good introduction to MPC can be found in this series of videos: `https://www.youtube.com/watch?v=8U0xiOkDcmw&list=PLn8PRpmsu08ozoeoXgxPSBKLyd4YEHww8`.

2.1 Formulation of Nonlinear Model Predictive Control

Now, we would like to design a Nonlinear MPC (NMPC) for our robot. Often the dynamics of a system can be written as

$$\dot{x}(t) = f(x(t), u(t)). \tag{1}$$

As mentioned previously, MPC is a discrete-time control framework. Thus, we cannot use the continuous-time representation of the dynamics (1) directly. Instead, we have to find a discrete-time representation

$$x_{k+1} = f_d(x_k, u_k). \tag{2}$$

We do not have explicit discrete-time dynamics like in (2) available for the robot. Instead we discretize the continuous-time dynamics

$$x_{k+1} = \underbrace{x_k + \int_{t_k}^{t_{k+1}} f(x(\tau), u(\tau))\mathrm{d}\tau}_{:=f_d(x_k,u_k)}, \tag{3}$$

where the integral in (3) can be approximated using a numerical integration scheme of choice. Often used schemes are Euler Forward and Runge Kutta 4.

We can use the discrete-time dynamics from (3) to design the optimization problem for our MPC controller. The dynamics are the first component. The next component we need is the optimality criterion, also known as **cost function**. Consider the following quadratic cost function

$$J(X, U) = \sum_{i=0}^{N-1} (x_i^\top Q x_i + u_i^\top R u_i) + x_N^\top P x_N. \tag{4}$$

Recall that you have encountered a very similar cost function in Linear Quadratic Regulator (LQR) theory.

The cost function of (4) is a weighted sum of the quadratic deviation of the states and inputs from the origin. The cost function is larger if the deviation from the origin is larger and vice versa. In fact, the cost function grows quadratically. Thus, minimizing $J(X, U)$ is identical to controlling the system states to the origin. The matrices Q, R, and P are known as weight matrices, because they encode how much we want to penalize each state and input. If you consider the robot, we may for example want to penalize the pitch angle more than the position and design Q accordingly. You can see that in (4) the sum from 0 to N is taken, where N is known as the prediction horizon. Think of it this way: using the dynamics of the system from (3), our MPC controller plans N steps into the future to design a trajectory $U = \left[u_0, u_1, \dots u_{N-1}\right]$ which minimizes the cost function (4) during the prediction horizon. The resulting system trajectory under U is denoted as $X = \left[x_0, x_1, \dots, x_N\right]$.

The last component of the optimization problem are the constraints. For this, refer to the optimization problem (5). When we try to minimize the cost function (5a), we need to tell it what the relationship between x_{k+1}, x_k, and u_k is. This is achieved through (5b), which is the dynamical constraint from equation (2). Additionally, we can include problem specific constraints, like physical limits of our system, or a safe operating range. We can summarize those constraints as $g_k(x_k, u_k)$, shown in (5c), (5d). Lastly, we constrain the first state of our optimization problem x_0 as equal to the measured state $\bar{x}$, i.e., $x_0 - \bar{x} = 0$ in (5e). The resulting optimization problem then reads as

$$\begin{aligned}
\min_{X,U} \quad & \sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k) + x_N^\top P x_N && \text{(5a)}\\
\text{s.t.} \quad & x_{k+1} - f_d^k(x_k, u_k) = 0, \quad k = 0, \dots, N-1, && \text{(5b)}\\
& g_k(x_k, u_k) \leqslant 0, \quad k = 0, \dots, N-1, && \text{(5c)}\\
& g_N(x_N) \leqslant 0, && \text{(5d)}\\
& x_0 - \bar{x} = 0 && \text{(5e)}
\end{aligned}$$

The optimization problem (5) is known as a Nonlinear Program (NLP), because the dynamics or the constraints may be nonlinear. Note that even though we do have a quadratic cost function, in general this problem does not fall into the important category of Quadratic Programs (QP), due to the general form of the constraints (a QP has linear constraints). Now that we have the NMPC formulation for our problem, we solve it repeatedly at discrete-time steps and at each time apply u_0 to our system, until the new u_0 is available. Note that solving such NLPs is not straightforward and part of ongoing research. Several methods exist, but none are discussed here. However, all of these methods are computationally intense, which is arguably the biggest drawback of MPC.

2.2 LQR and Linear MPC

To get some more insight into MPC, in particular the cost function (4), we show how MPC and LQR are related. This comparison also enables us to make some statements about the stability of Linear MPC (LMPC).

Recall the main idea behind LQR: For a linear system and a quadratic cost function, the linear-quadratic regulator provides a feedback control law that minimizes the cost function when applied to the system. Note that there exists a continuous as well as a discrete time formulation. Furthermore, for each of those, there is a finite-horizon and an infinite-horizon formulation. We quickly recap the discrete case here. For a system

$$x_{k+1} = Ax_k + Bu_k \tag{6}$$

and a cost function

$$J(x_0, U) = \frac{1}{2}\sum_{k=0}^{N-1}\left[x_k^T Q x_k + u_k^T R u_k\right] + \frac{1}{2}x_N^T P_f x_N \tag{7}$$

we want to solve the problem

$$\min_U J(x_0, U) \tag{8}$$

under some technical assumptions on the weight matrices Q, R and P. For a finite horizon N, we obtain the state feedback law

$$u_k = F_k x_k \tag{9}$$

where

$$F_k = -\left(B^T P_{k+1} B + R\right)^{-1} B^T P_{k+1} A \tag{10}$$

and where P_k is found from the backward Riccati iteration

$$P_{k-1} = Q + A^T P_k A - A^T P_k B \left(B^T P_k B + R\right)^{-1} B^T P_k A \tag{11}$$

with terminal condition $P_N = P_f$. The drawback of the finite-horizon formulation is that it is not necessarily stable. Stability can be achieved by using an infinite-horizon controller (under some controllability assumption). Such a controller is obtained by iterating the Riccati equation until it converges. The corresponding cost function is

$$J(x_0, U) = \frac{1}{2}\sum_{k=0}^{\infty}\left[x_k^T Q x_k + u_k^T R u_k\right] \tag{12}$$

The control law is

$$u = Fx \tag{13}$$

with

$$F = -\left(B^T P B + R\right)^{-1} B^T P A \tag{14}$$

and

$$P = Q + A^T P A - A^T P B \left(B^T P B + R\right)^{-1} \tag{15}$$

It can be seen that the formulation of the finite LQR corresponds to an MPC controller of a linear system without any constraints (except for the system dynamics). However, the concepts differ in the way they are applied to the system. Here, the main difference between MPC and LQR is that the horizon of LQR is fixed, while MPC uses a moving (receding) horizon. This means that in LQR, the solution is computed in advance and then applied for the whole horizon. For MPC, the horizon is typically shorter, but the optimization is regularly solved, and only the first obtained input is applied. Do not let the phrase “the solution is computed in advance and then applied” confuse you into thinking that LQR is a feed forward control concept (indeed, it uses state feedback). You can read more about the derivation of LQR, MPC, and their relationship in [2].

The concept of finite horizon LQR can now be extended to include constraints. If we restrict the constraints to be affine, an LMPC controller results. The optimization problem is now a quadratic program, which has many desirable properties. As we have seen, finite-horizon LQR and therefore LMPC is not guaranteed to be stable. In general, the stability of MPC depends on many factors, such as the length of the prediction horizon, the cost functions weights, and the constraints. There exists many results concerning the stability of LMPC. For NMPC, the situation is much more complex, and general statements regarding stability are usually not possible. Some mechanisms to assess the stability of MPC are shown in the following.

First, we consider a infinite-horizon MPC formulation with a quadratic cost function. Using the theory of Lyapunov functions, under some conditions, one can show that such a controller leads to a stable closed-loop system. Note that in fact, the problem need only be solved once, since the horizon is infinite, and the solution in the next step is always the solution from the previous step shifted by one. However, there is a practical complication: Such a problem cannot be solved, due to the infinite number of optimization variables. Nonetheless, notice the parallels to finite- and infinite-horizon LQR. A method of achieving a stable MPC controller is by approximating the infinite-horizon problem. Several methods exist to achieve this:

- Terminal constraints: Assume we want to control a system to the origin, which is assumed to be an equilibrium point of the system. In this method, a terminal constraint is included as

  $$x(N) = 0 \tag{16}$$

  If a solution exists for the first iteration, it is clear that the system reaches $x = 0$ after at most N steps. Since no additional cost is incurred during a hypothetical horizon from N to ∞, this problem is now equivalent to a infinite horizon problem and therefore stabilizing.

- Terminal set and terminal cost: Instead of restricting the final state to the origin, we loosen the constraint to a set, the so-called terminal set $\mathcal{X}_f$, i.e. we require

  $$x(N) \in \mathcal{X}_f \tag{17}$$

  Furthermore, we add a terminal cost of the form $x(N)^T P x(N)$ to the cost function. Broadly speaking, if we can guarantee that there exists a controller that stabilizes the system inside the terminal set, it it sufficient to drive the state into $\mathcal{X}_f$ within the horizon. This formulation is therefore less restrictive than the terminal constraint. There also exist approaches that only use a terminal cost. Here, the terminal cost can be seen as an approximation of the “tail” of the cost from N to ∞. If the approximation is sufficiently good, the stability properties of infinite-horizon MPC are recovered. For linear systems, the stabilizing terminal set and terminal cost are closely related to LQR. However, we do not cover this here.

- No stability constraints: Here, the idea is to approximate the infinite horizon with a sufficiently long but finite horizon N. This method is attractive because it is very simple to implement in practice. However, to determine the length of the stabilizing horizon can be hard, as it also depends, among other parameters, on the reference that is to be tracked. Often, a suitable value of N is determined in simulations. Note that increasing the horizon usually increases the computation time.

Note that each of these methods has drawbacks, e.g. due to being too restrictive/conservative, increasing the complexity of the problem, or shrinking the set of feasible initial conditions. The presented methods are often combined to various hybrid methods.

3 System Model

3.1 Introducing the Robot

This section introduces the two-wheeled balancing robot hardware platform, which all the controllers that are developed during this lab exercise will be implemented and tested on.

The robot is based on a robot kit by the manufacturer Pololu. It consists of a low-level Arduino capable board (based on an Atmel ATmega32u4 microcontroller), which contains several sensors, and a Raspberry Pi 3B+. The low-level board is responsible for the power electronics and controlling the electric motors. Furthermore, it hosts the wheel encoders for each electric motor and the IMU (inertial measurement unit), consisting of a gyroscope and an accelerometer. Lastly, the board provides an I^2C communication interface to access sensor data and to interact with the Arduino. We can program the low-level board with the Arduino interface. Using I^2C, we connect the Raspberry Pi 3B+ to the Arduino board. The Raspberry Pi is responsible for high-level control of the robot. Thus, all of the controllers that you will implement in this laboratory practice today will run on the Raspberry Pi, which runs Ubuntu 18.04. The Lab PC connects to the Raspberry PI via a wireless network, which is generated by the Raspberry Pi using the onboard WiFi chip. To exchange data and send commands, we run a web server featuring a graphical user interface on the Raspberry Pi, which we connect to from the Lab PC. Figure 3 shows an overview on the hardware setup of the robotic system.

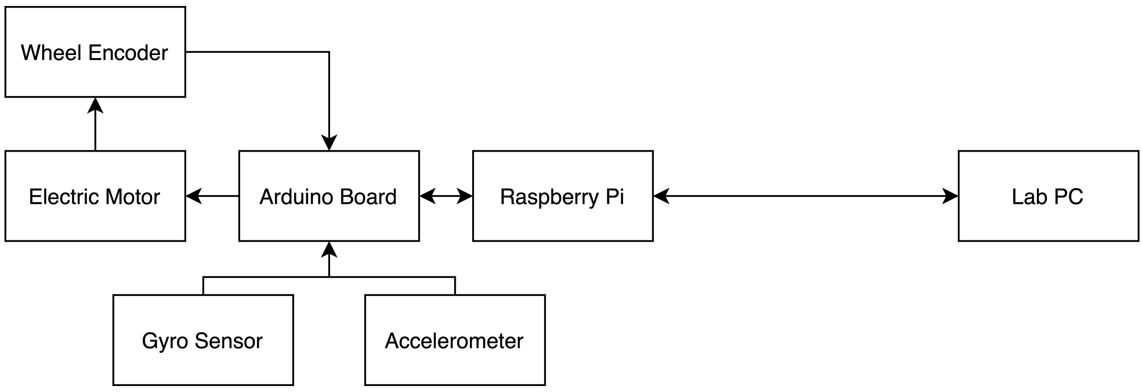


Figure 3: An overview of the hardware setup of the robot.

To design a PID controller for a system, we do not necessarily need to have a system model. If we have the system at hand and it is not too delicate, we can simply use the system to design a PID controller. While this tuning effort is heuristic and does not scale well to large or complex systems, it is still a good first approach to get a working controller.

Exercise 3.1. *PID Tuning Methods*
Name a PID controller tuning method that does not rely on a system model. What advantages and drawbacks do you identify with this method? You may want to refer to chapter 11 in the book Analysis and Synthesis of Single-Input Single-Output Control Systems *(3rd Ed., 2011) [3], in the following referred to as* the Book.

In contrast, designing an MPC controller is not as straightforward. It requires knowledge of the relevant system dynamics. Depending on the requirements, this knowledge has to be relatively accurate.

Our robot is a type of an inverted pendulum, specifically a so-called *two-wheeled inverted pendulum.* Another type of an inverted pendulum, the *inverted pendulum on a cart* (IPC) is often used as a benchmark system in control research. The dynamics of the IPC and our mini-segway share many similarities. However, the dynamics are not identical. You can read about the IPC in the Book. In particular, Figure 1.11 (p.13) shows a schematic diagram of the IPC. Furthermore, Example 4.4.1

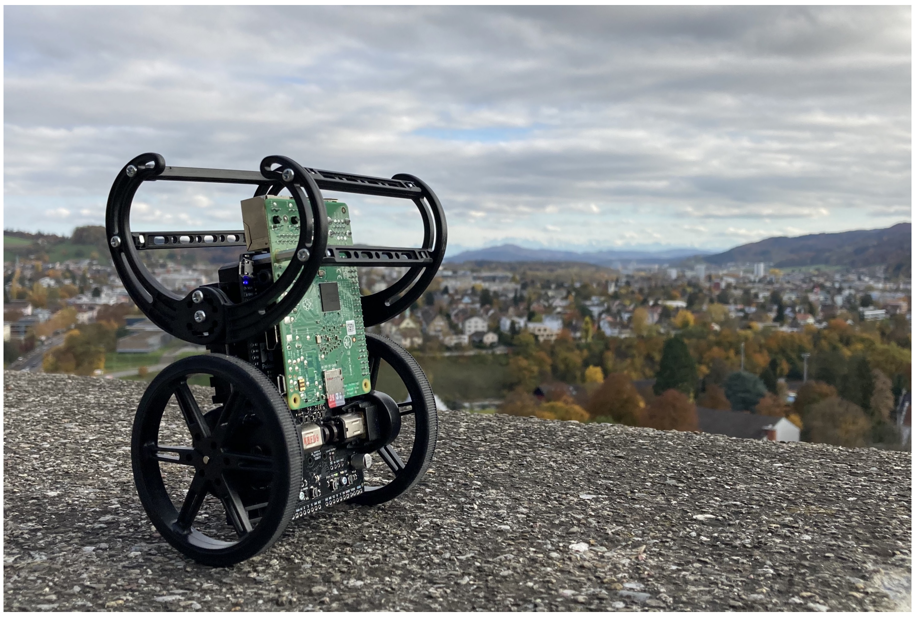


Figure 4: Picture of the robot used in this laboratory practice.

(p.53) presents the nonlinear state dynamics of the system and its linearization. Finally, Exercise 14.4.1 (p.217) shows the controller design procedure using loop shaping. Read these sections if you are interested, or come back to them if you are stuck during the following homework exercises. Read these short sections as preparation for the model derivation that follows.

3.2 Nonlinear System Model

Our robot (i.e. the two-wheeled inverted pendulum) is a system with 3 degrees of freedom (3-DOF). The three states are the pitch angle θ, the yaw angle ψ, and the straight motion (or forward) position x as shown in figure 5. In the following, we present the dynamics of the robot, governed by three differential equations, known as equations of motion (EOM). Have a look at them and try to comprehend their meaning. However, an in-depth understanding is not vital for this exercise. Although our robot may look simple, its dynamics are not. A comprehensive derivation of the EOM can be found in [4].

Table 1: Model quantities of the robot.

	Definition	**Symbol**
Inputs	Wheel torques exerted at (L, R)	T_L, T_R
States	Forward position, pitch angle, yaw angle	x, θ, ψ

The EOM of the 3-DOF model in terms of the generalized coordinates (x, θ, ψ) are as follows [4]:

$$(m_B + 2m_W + \frac{2J}{r^2})\ddot{x} - m_B l(\dot{\psi}^2 + \dot{\theta}^2)\sin\theta + (m_B l\cos\theta)\ddot{\theta} = \frac{T_L + T_R}{r} \tag{18}$$

$$(I_2 + m_B l^2)\ddot{\theta} + (m_B l\cos\theta)\ddot{x} + (I_3 - I_1 - m_B l^2)\dot{\psi}^2\sin\theta\cos\theta - m_B lg\sin\theta = -(T_L + T_R) \tag{19}$$

$$\begin{aligned}(I_3 + 2K + m_W\frac{d^2}{2} + J\frac{d^2}{2r^2} - (I_3 - I_1 - m_B l^2)\sin^2\theta)\ddot{\psi} + (m_B l\dot{x} - 2(I_3 - I_1 - m_B l^2)\dot{\theta}\cos\theta)\dot{\psi}\sin\theta \\ = (T_R - T_L)\frac{d}{2r}\end{aligned} \tag{20}$$

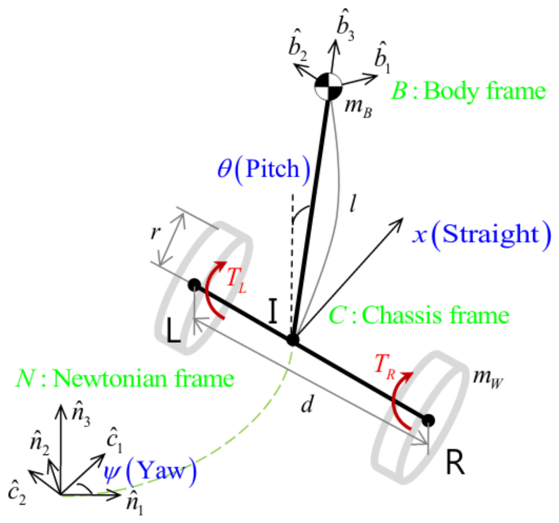


Figure 5: Illustrating how the three variables pitch angle θ, yaw angle ψ and straight motion position x are defined. From [4].

Table 2: Model parameters of the robot.

Definition	**Symbol**	**Value**	**Unit**
Distance between the two wheels	d	0.1	m
Distance of wheel axle to center of mass	l	0.0275	m
Wheel radius	r	0.04	m
Mass of the pendulum body (except wheels)	m_B	0.43	kg
Mass of each wheel	m_W	0.02	kg
Single wheel moment of inertia (MOI) w.r.t. the wheel axis	J	1.92×10^{-4}	$\mathrm{kg\,m^2}$
Earth gravitation	g	9.81	$\mathrm{m/s^2}$
MOI of each wheel w.r.t. the vertical axis	K	1.0×10^{-4}	$\mathrm{kg\,m^2}$
MOI of the pendulum body w.r.t. {B}	I_2	1.49×10^{-3}	$\mathrm{kg\,m^2}$
MOI of the pendulum body w.r.t. {B}	I_1, I_3	*unknown*	

Exercise 3.2. *Dimensionality Reduction*
Can you think of a way to reduce the complexity of the model, i.e. to reduce the number of states, using a few assumptions (Hint: We can reduce the 3-DOF model to a 2-DOF model)? Which variable is no longer necessary? What assumption about the inputs have you made?

In two dimensions (2D), the equations reduce to [5]

$$\ddot{x} = \frac{1}{d_1}\left(aI_O\dot{\theta}^2 \sin\theta - a^2 g \sin\theta\cos\theta + T\left(\frac{I_O}{r} + a\cos\theta\right)\right) \tag{21}$$

$$\ddot{\theta} = \frac{1}{d_1}\left(-a^2\dot{\theta}^2 \sin\theta\cos\theta + am_O g\sin\theta - T\left(m_O + \frac{a}{r}\cos\theta\right)\right) \tag{22}$$

where the following representations have been used to facilitate reading:

$$\begin{aligned} a &= m_B l \\ I_O &= I_2 + m_B l^2 \\ m_{tot} &= m_B + 2m_W \\ m_O &= m_{tot} + \frac{J}{r^2} \\ d_1 &= I_O m_O - a^2 \cos^2\theta. \end{aligned} \tag{23}$$

The states that remain in our model are x and θ. During this laboratory practice, we are concerned with controlling these two states.

Exercise 3.3. *Intuition: Stability of the Nonlinear System*
When controlling the robot, our goal is to keep it upright, i.e. at $\theta = 0$. Using intuition, explain whether this is a stable or unstable equilibrium. Is there another equilibrium point?

The input into our model is the torque T (or T_R, T_L in the 3D case). However, the robot does not have an internal torque control. Instead, our input is the voltage to the electric motors. An electric motor has internal electro-magnetic dynamics which map an input voltage to an output torque. For our robot, these dynamics are extremely fast, which allows us to neglect them. The resulting conversion used to map the input voltage to torque is

$$T = \frac{K}{R}(u - K(\frac{\dot{x}}{r} - \dot{\theta})), \tag{24}$$

where u is the voltage, K is the motor constant, and R is the resistance of the motor coil. From now on, the input always denotes the input voltage.

3.3 Linearized Model

In the different tasks of this experiment you will mostly deal with MPC controllers. However, to have a connection to an area of control theory that is known to you, we come back to classical PID control from time to time. To design PID controllers, we use a simulation platform of the linear system. To this end, we linearize the 2D nonlinear system dynamics around the upper equilibrium point $x = 0,\ \theta = 0,\ T = 0$ and obtain the following linearized system dynamics:

$$\dot{\xi} = A\xi + Bu = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -12.698 & -1.926 & 0.508 \\ 0 & 0 & 0 & 1 \\ 0 & 234.62 & 96.094 & -9.385 \end{bmatrix} \xi + \begin{bmatrix} 0 \\ 4.2695 \\ 0 \\ -78.890 \end{bmatrix} u \tag{25}$$

where u is the motor voltage, and the state vector ξ is defined as follows

$$\xi = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}. \tag{26}$$

Exercise 3.4. *Stability of the Linear System*
Use linear system analysis to determine whether this system is stable. Does this generalize to the nonlinear system? Why? (Hint: If necessary, refer to the Section 4.4 (p.50) of the Book.)

Exercise 3.5. *MPC vs. Classical*
Name three advantages and disadvantages of MPC compared to a simple classical control concept such as PID.

4 PID

Here, we recap some basic PID theory, as well as an introduction to cascaded control. For more details, refer to the Book, in particular Sec. 11.2 for PID and Sec. 13.2 for cascaded control. During the laboratory exercise, you will not explicitly interface with a classical/PID controller, however, comparisons between MPC and PID are drawn and working PID controllers are already implemented for any possible PID aficionados.

4.1 PID Control

PID controllers are part of linear control theory, and therefore the linearized model introduced in section 3.3 is relevant here. A PID controller in its basic formulation consists of three parts, the proportional part (P), the integrative part (I), and the derivative part (D):

$$u(t) = \underbrace{K_p e(t)}_{P} + \underbrace{K_i \int_0^t e(\tau)\mathrm{d}\tau}_{I} + \underbrace{K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}}_{D} \tag{27}$$

where K_p, K_i, and K_d are tunable gains, and $e(t)$ is the control error

$$e(t) = r(t) - y(t) \tag{28}$$

with $r(t)$ the reference signal and $y(t)$ the system output. In the Laplace domain, (27) becomes

$$\frac{U(s)}{E(s)} = C(s) = \left(K_p + K_i \frac{1}{s} + K_d s\right) \tag{29}$$

where we call the system $C(s)$ the *controller*. Some more advanced formulations are possible, for example, it is usually necessary to add a *roll-off term* (to have a causal controller) as shown in (30) or some low-pass filtering to the D-part as in (31).

$$C(s) = \left(K_p + K_i \frac{1}{s} + K_d s\right) \frac{1}{\tau s^2 + 1} \tag{30}$$

$$C(s) = \left(K_p + K_i \frac{1}{s} + K_d \frac{s}{T_f s + 1}\right) \tag{31}$$

4.2 PID Implementation

Here, we show how the controller (31) can be implemented. After tuning in the continuous frequency domain (s-domain), the controller must be transformed into the discrete frequency domain (z-domain), to be implemented on discrete computing hardware. Using Euler Forward as discretization method, i.e. using

$$s = \frac{z-1}{T_s} \tag{32}$$

with T_s the sampling time, it is straightforward to bring the controller into the following form

$$C(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{33}$$

with

$$b_0 = K_d + K_p T_f \tag{34}$$

$$b_1 = K_p T_s - 2K_d + K_i T_f T_s - 2K_p T_f \tag{35}$$

$$b_2 = K_d - K_i T_f T_s + K_i T_s^2 + K_p T_f - K_p T_s \tag{36}$$

$$a_0 = T_f \tag{37}$$

$$a_1 = T_s - 2T_f \tag{38}$$

$$a_2 = T_f - T_s \tag{39}$$

Rearranging and transforming (33) into the discrete time-domain yields

$$u[k] = -\frac{a_1}{a_0}u[k-1] - \frac{a_2}{a_0}u[k-2] + \frac{b_0}{a_0}e[k] + \frac{b_1}{a_0}e[k-1] + \frac{b_2}{a_0}e[k-2] \tag{40}$$

After substitution, the above equation can be implemented on a computer. Other formulations, e.g. (29) or (30) can be transformed using the same principle.

4.3 Cascaded Control

Cascaded control systems can be used to control a subset of single input, multiple outputs (SIMO) systems. If such a system has *time-scale separation*, cascaded control is a suitable concept that allows the control of these plants without requiring advanced MIMO techniques.

To apply a cascaded controller, the plant is separated into a fast and a slow part. The fast dynamics are controlled in an *inner loop*, while the slow dynamics are controlled in the *outer loop*. The output of the slow controller can then be interpreted as the reference for the faster inner control loop. This structure is shown in figure 6.

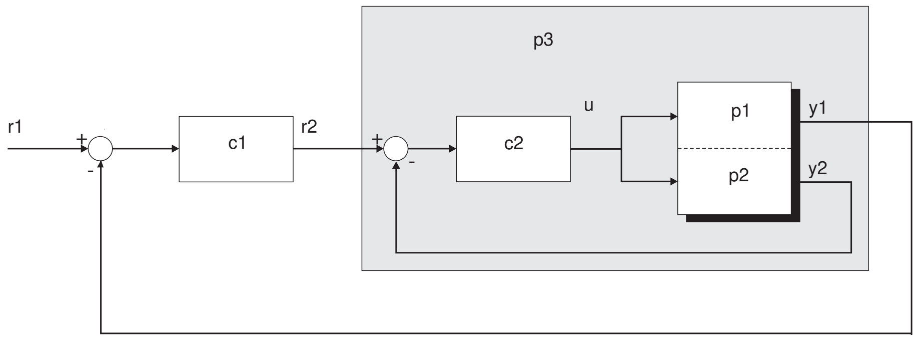


Figure 6: The structure of a cascaded controller. From [3].

Exercise 4.1. *Cascaded Control*
Using figure 6, derive the expression for P_3 in terms of P_1, P_2 and C_2, where

$$P_3(s) = \frac{Y_1(s)}{R_2(s)}$$

This concludes the mandatory homework assignment.

5 Pre-Lab

Here begins the practical part of the laboratory exercise. We begin with a pre-lab, which introduces the hardware (the robot) and the interface that is used to interact with it. Afterwards, you will be able to experiment with different MPC controllers that are implemented on the robot. Work through the script and periodically discuss your answers with your assistant (e.g. after you worked through a section). You are allowed (and expected, within reason) to use any resource available to you to solve the exercises: grab the robot and inspect it, use the script and old summaries from previous courses, and most importantly: **research online**. The login and password for the lab PC in front of you are `.\Sigi` (including the full stop and backslash) and `Laborprakt!kum`, respectively.

5.1 Model Identification

Exercise 5.1. *Identifying the Components*
Thoroughly inspect the robot and all its components. Refer to figure 3 for an overview of the main components of the robot. See whether you can identify all of them on the hardware. Although the IMU is hidden, where would you expect it to be located?

Exercise 5.2. *Model Parameters*
Recall the parametrized nonlinear model equations. Assuming you have only common measurement devices (e.g. scale, ruler, calipers, etc.) to estimate the model parameters, which of the parameters introduced in table 2 and listed again below do you expect to be hard to measure/obtain and therefore most likely to introduce modeling errors? Explain.

- *Wheel: mass, radius, mass moment of inertia (MOI)*
- *Body: mass, distance between the wheels, distance from the wheel axle to the center of mass, MOI*
- *Motor: motor constant, coil resistance*

5.2 Startup

In this section you will first estimate a reasonable upper limit on the operation time of the segway and later boot the robot.

Exercise 5.3. *Power Consumption*
Provide estimates for the expected runtime of the robot on a single battery charge for two scenarios: in idle mode (the two motors not powered) and when the robot operates at maximum power.

Hints: A single motor, as per the data sheet, draws 0.81 A at 6 V at a maximum power operating point. Inspect the battery configuration (parallel or series).

To power up the robot, lay it on its front (i.e. with the batteries facing downwards) and have the wheels pointing towards you. Then press the button located on the very bottom left. The Raspberry Pi is now booting, which takes a few minutes.

Now that the Raspberry Pi has had time to boot, log in at the lab computer and connect to the Wi-Fi network `Segway-Wifi-X`, where `X` is a number. Use the password `segway-wifi`. Next start the program *PuTTY*. As the destination we want to connect to, enter the IP address `192.168.4.1`, port 22 and option `SSH`, then click *Open*. Alternatively double-click on *Sigi*. This opens a remote terminal session on the Raspberry Pi via SSH. In this terminal, enter the username `segway` and the password `segway` when prompted (the typing does not appear on the screen as for the username). You are now in the home directory of the *segway* user on the Raspberry Pi. Navigate to our working directory by entering

```
$ cd git/segway-server
```

Press the tabulator twice for autocomplete suggestions while typing. Finally, start the *Python* script that generates the interface we are going to use to control the robot with the command

```
$ ./main.py
```

Now start *Firefox* on the lab PC and navigate to the webpage served by the Raspberry Pi by entering `192.168.4.1:9000` in the navigation bar or alternatively click on the book mark. You should now see an interface similar to Fig. 7. Referring to the numbers shown in figure 7, the interface is explained in the following:

1. Controller selection: switch to the appropriate controller which will be indicated in the exercises. Select "MOTORS OFF" to view sensor data without control action.
2. Parameter tuning: enter values via keyboard, use the up/down buttons or the slider to change controller parameters on-line (i.e. also during operation).
3. Simplified representation of the selected controller.
4. Supervision buttons: "START" calibrates the sensors, initiates data acquisition and primes the controller. "STOP" ends all control operation (i.e. motor shutdown, no data acquisition). "CALIBRATE" only does calibration without any control action.
5. Additional parameters: controller sampling time T_s, switches for additional yaw controller with angular reference and gain, and bang-bang controller with separate gain.
6. Reference: "APPLY STEP" applies a step to the provided value. "APPLY FILTERED STEP" applies a filtered step. Used for position reference tracking. Value remains until reset. To reset the reference to zero, use "RESET REFERENCE" or enter 0 and apply a (filtered) step.
7. Define displayed time range of the plot and option to reset the live plot.
8. Apply predefined trajectories (used in sections 10 and 11).
9. Live plot of the measured pitch angle, position and the current reference position.

Stay on the "MOTORS OFF" tab for now. Lay the robot flat onto its front (i.e. battery facing downwards). On the interface, press the "START" button and observe a red LED on the Arduino board turning on for a short moment, indicating that a calibration sequence is running. After calibration, a yellow LED turns on, indicating that the robot is ready. You can now move it around.

Exercise 5.4. *Calibration*
Which sensors require calibration, and how is it performed? Think about how we can obtain absolute measurements from the sensors.

5.3 Shutdown

Sometimes, it might be necessary to shut down parts of the software or the Sigi as a whole.

To stop the Tornado server, which is responsible for displaying the user interface, use the sequence "Ctrl + C".

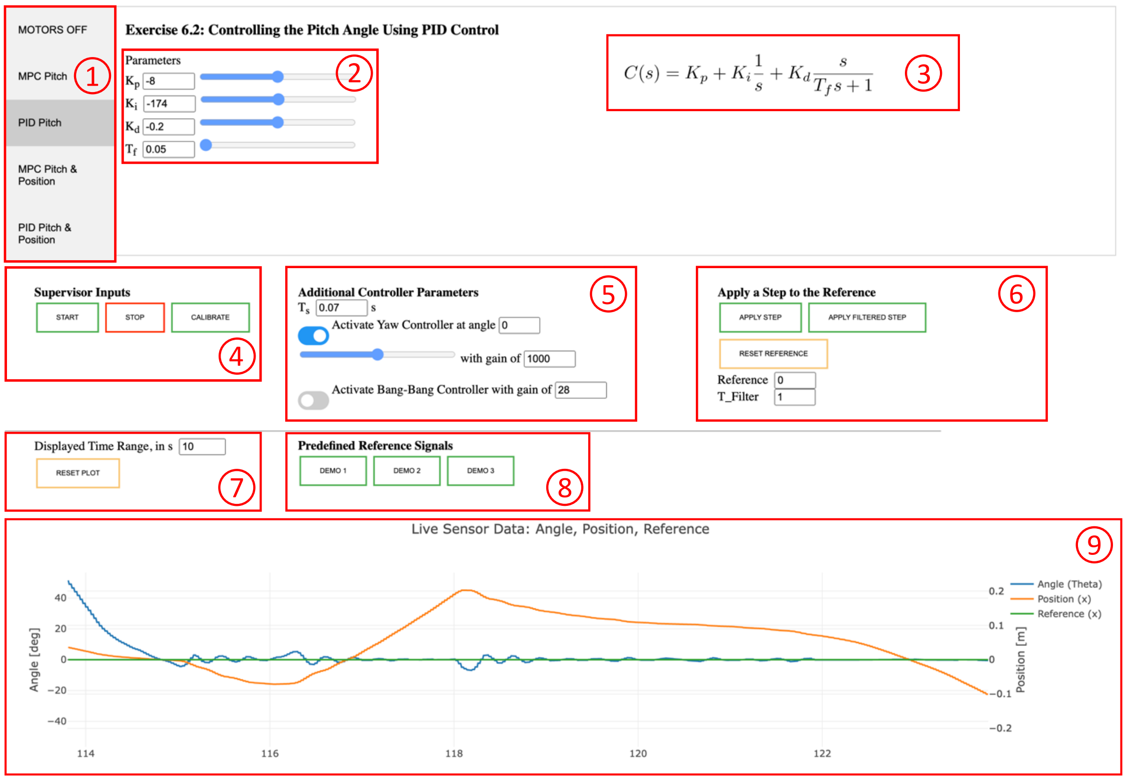
In order to turn off the Sigi completely, write

```
sudo shutdown now
```

into the terminal.

5.4 Sensor Measurements

As explained in section 3.1, our mini-segway is equipped with two sensors (or more precisely, sensor units): the motor encoders, and the IMU, which is mounted on the body of the robot. We look at both sensors separately. Optionally, refer to Example 5.3.1 (p.64) in the Book.



Exercise 6.2: Controlling the Pitch Angle Using PID Control

Parameters
K_p -8
K_i -174
K_d -0.2
T_f 0.05

$$C(s) = K_p + K_i \frac{1}{s} + K_d \frac{s}{T_f s + 1}$$

Additional Controller Parameters
T_s 0.07 s
Activate Yaw Controller at angle 0
with gain of 1000
Activate Bang-Bang Controller with gain of 28

Reference 0
T_Filter 1

Displayed Time Range, in s 10

Figure 7: The interface used to control the robot.

Exercise 5.5. *Inertial Measurement Unit*

1. *First, we investigate the inertial measurement unit (IMU). This sensor unit measures multiple quantities, six in total. Research what said quantities are (Hint: The IMU consists of an accelerometer and a gyroscope).*
2. *Now, knowing that the IMU is mounted on the robot's body, one of the six measurements can be related to one of the model states in a straightforward manner (Hint: Refer to figure 5 for the definition of the body frame). Which one?*
3. *The body pitch angle θ is of primary interest when stabilizing the robot. What is the relation between the measurement determined above and θ?*
4. *The process that is employed here is called dead reckoning; it has a major disadvantage. Can you think of what this might be?*

Exercise 5.6. *Wheel Encoder*

1. *The second sensor is a rotary encoder that is mounted on the shaft of the wheel motor. State which quantity is measured by this sensor (research on your own if necessary).*
2. *This measurement is related to the axle position x, a model state that we may be interested in controlling. Is the encoder measurement enough to determine x? Explain. (Hint: It is not enough, we need information about another state.)*
3. *Provide the equation for x, assuming you have the necessary measurements available.*
4. *The IMU signal had the problem of drift, do you expect this to be the case for the encoder as well?*

6 Pitch Angle Stabilization

In this exercise, we want to control the pitch angle θ of the robot. We will use both MPC and PID control for this task, however the focus of this experiment lies on model predictive control.

6.1 MPC

Before running MPC on the segway, we first bring the MPC theory closer to practice.

Exercise 6.1. *Main Components of an MPC Controller*
Refer to the MPC theory presented in the homework assignment, in particular the generic NLP used in MPC (5). List the main building blocks that characterize an MPC and make it different from "classical" approaches.
Hint: *Think of the cost function, etc...*

In the following, we want to gain some understanding of what these components mean in the context of the actual robotic system, as well as how their "values" may be chosen. Assume that you want to design a controller that is able to stabilize the robot in terms of the pitch angle θ, while the position x shall not be controlled.

Exercise 6.2. *Designing the Controller*
Having discussed your solution to the above exercise with your assistant, propose values where applicable, otherwise discuss possible choices for each of the components of an MPC controller, and what considerations influence these choices. At least, talk about the following points

- *Choice of the predicting horizon: What does N represent? How is it connected to the horizon? How can it be chosen? Propose a range for N.*
  Hints: *N is connected to the integrator step size $T_{\text{int}} = 0.07\,\text{s}$ in the cost function (4). Think about how long the horizon should be to sufficiently predict the system dynamics.*
- *Cost function: Which variables are present in the cost function? What is the influence of the weights when they are increased?*
- *Constraints: Propose constraints on input and state variables. Which are necessary for proper functionality? What would be the benefit of additional, non-essential constraints.*

Exercise 6.3. *Predicting the Behavior*
Equipped with an MPC controller similar to the one you designed above, what behavior do you expect the robot to exhibit? What is the effect of changing the parameters in the cost function (4) on the behavior of the robot?

Finally, we are ready for a demonstration. The controller has already been implemented for you on the robot.

Exercise 6.4. *Testing the MPC Controller*
At the lab PC, click on the tab "MPC Pitch" in the web interface. Play around with the parameters of the MPC controller. Get a feeling for the influences of the available tuning knobs.

6.2 PID

The task of just controlling the pitch angle θ is a SISO problem and therefore a simple PID controller can be used to keep the robot in an upright position. Such a PID controller has been implemented and tuned. Therefore, you don't have to change the parameters.

Exercise 6.5. *Meaning of PID Parameters*
What is the meaning of the PID parameters presented in section 4? Explain what K_p, K_i, K_d, and T_f stand for.

Exercise 6.6. *Testing the PID Controller*
At the lab PC, click on the tab "PID Pitch" in the web interface. Verify that the controller stabilizes

the robot. Try to optimize the performance of the MPC by tuning its respective weights. The PID controller parameters have already been optimized for you. If you wish, you can try to further improve them. Which controller performs better for this task?

7 Position Tracking

In this second part of the laboratory exercise, the goal is to extend the controllers developed in the previous section 6 to also control the position x. Furthermore, we want the position to track a provided reference value. We will again discuss the control problem for both approaches MPC and PID.

7.1 MPC

Exercise 7.1. *Extension to Position Tracking*
What changes to the controller developed in section 6.1 are necessary to be able to also control the position x?

Exercise 7.2. *Position Reference Tracking*
How can reference tracking be implemented in MPC?
(Hint: A modification to the cost function is necessary.)

Examine the behavior of the extended controlled on the robot by selecting the "MPC Pitch & Position" tab. You can apply a step in the position reference using the "APPLY STEP" button. This will change the position reference from the initial value of 0 to any specified value the very moment you click the button. Note that this immediate jump in the reference signal is very challenging for the control of the robot. However, using a model predictive controller with sufficiently fine-tuned parameters, these non-smooth reference signals can be dealt with.

Exercise 7.3. *Testing MPC for Step Reference*
Try to find a suitable set of parameters to handle a step in the reference signal of 1 meter.

Exercise 7.4. *Reference as Constraint*
You have now seen how reference tracking can be achieved by a modification in the cost function. However, in principle, we could also include the reference as a constraint. Why is this usually not a good idea?

7.2 PID

In this subsection we will investigate an extension to the PID controller to also do reference tracking.

Exercise 7.5. *Extending the PID Controller*

- *If you want to use linear techniques to control the position, what type of model do you have to deal with?*
- *Can you think of a SISO technique that is suitable here? Discuss with your assistant.*
- *Compare the necessary changes to the MPC controller to what is needed to use said SISO technique for the position tracking. Explain why MPC is much more adaptable/flexible.*

A cascaded controller is implemented to control the angle θ and the position x.

Exercise 7.6. *Testing the Cascaded Controller*
At the lab PC, click on the tab "PID Pitch & Position" in the web interface. Verify that the controller stabilizes the robot and holds the reference position.

As a next step we investigate the reference tracking properties of the proposed controller.

Exercise 7.7. *Reference Tracking Cascaded Controller*
Can you think of a problem that arises if we use a cascaded PID controller to track a reference step in the position? Verify your concerns by clicking "APPLY STEP" to impose a new shifted reference.

Obviously the cascaded PID controller is not able to deal with sudden changes in the reference, such as for example steps. A possible way around is to filter the reference signal. The following discrete update equation describes a moving average filter with r as the reference signal, x as the hidden state, r_f as the filtered reference signal, T_s as the sampling time and T corresponding to the time constant of the filter:

$$x(n+1) = \left(1 - \frac{T_s}{T}\right) \cdot x(n) + \frac{T_s}{T} \cdot r(n) \qquad (41)$$
$$r_f = x(n)$$

The effect of this smoothing filter is demonstrated on a step reference in figure 8.

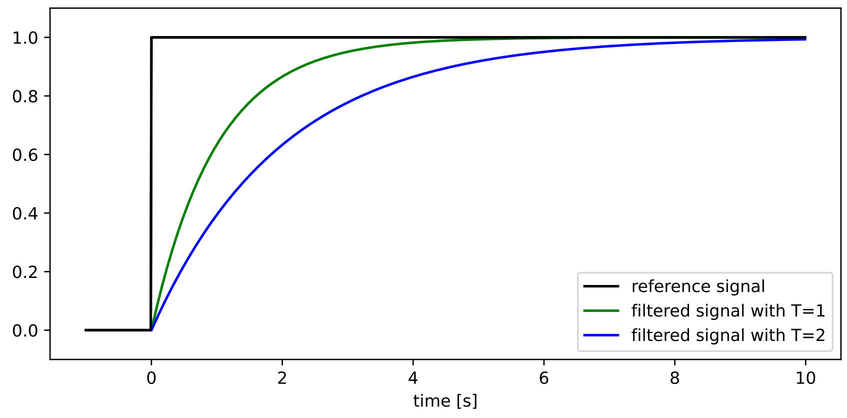


Figure 8: Reference signal filtered by a moving average filter with time constants $T = 1$ and $T = 2$.

The above moving average filter is implemented and applied to the reference signal when clicking the button “APPLY FILTERED STEP” in the web interface. One can alter its time constant with “T_Filter” and observe the resulting reference signal in the live plots of the GUI.

Exercise 7.8. *Filter Time Constant Tuning*
Tune the filter time constant T (“T_Filter”) such that the segway is able to manage a reference step of 1 meter as fast as possible.

8 Yaw Angle Control

During all the experiments so far, a proportional controller for the yaw angle was running on top of the discussed controllers (MPC and PID) to make sure that the robot does not turn around its vertical axis. When disabling this controller (switch "Activate Yaw Controller off") you will notice the robot turning around its vertical (i.e. yaw) axis in a somewhat unpredictable manner. Also for a step in the reference, the robot will move on a curved trajectory, probably even hitting a wall.

Exercise 8.1. *Curved Trajectory*

- *Can you think of the reason why this behavior occurs?*
- *Which of these effects can be measured by the motor encoders, and which cannot?*

Exercise 8.2. *Yaw Angle Control*
What can be done to improve on this problem, i.e. to reduce the turning and the curved trajectories? Can it be completely eliminated?

Exercise 8.3. *Model Augmentation*
This problem could also be tackled on the modeling level, i.e. by augmenting the system model. What are the advantages over the approach discussed above? Why may it not be a good option in our case?

Using the appropriate switch in the interface, inspect the influence of the yaw compensator on the control performance. The superimposed yaw controller can also be used to make turns with the segway. For this purpose, any angle between -180° and +180° can be inserted as angular reference.

Exercise 8.4. *Yaw Control Gain*
Investigate the influence of the controller gain on the turning behaviour of the segway by trying out different gains and reference angles.

9 The Bang-Bang Part

To give some background about where the term bang-bang control comes from, we note that it stems from the field of optimal control and refers to a situation, where the optimal control input is always at the limit. Think for example of a refrigerator that is tasked with keeping the temperature in a certain range. Since the compressor is most efficient when it runs at maximum power, the optimal control strategy is as follows: No cooling action while the temperature is below the maximally allowed temperature, then run the compressor at maximum power until the lower limit of the temperature range is reached, then turn the compressor off and the loop repeats. This is a bang-bang control strategy. We use the term slightly differently: for us, the term bang-bang indicates a modification of the control output, such that no small values are applied to the system. Using a sign function, the input is shifted, as shown in the following equation

$$\bar{u}[k] = u[k] + \kappa_{bb}\ \mathrm{sgn}(u[k]) \tag{42}$$

The constant κ_{bb} is already appropriately chosen, you do not have to tune it yourself.

Exercise 9.1. *Bang-Bang Controller*
What is the benefit of using a bang-bang part in this application? Think about the ideal relation between the input voltage u and the torque T, and what this relation looks like in the real world.

Using the appropriate switch in the interface, inspect the influence of the bang-bang part on the control performance. Note that this is a vital addition to the MPC, since we did not model the voltage-to-torque dynamics occuring at low voltages.

10 Comparison MPC and PID

In this section we will highlight the strengths and weaknesses of MPC and PID control. Performance is evaluated by applying the reference signal in figure 9 to both algorithms. The signal consists of multiple steps, alternating in sign around the starting point at zero, with increasing stepsize by 0.1 m per step. The duration of each individual step is 5 seconds. The stepsize keeps increasing unless either "RESET REFERENCE" is pressed or the maximum stepsize of 1.7 m is reached. Of course, as the PID controller cannot handle steps in the reference, the signal is smoothed with the moving average filter from section 7.2 in that case. Clicking the button "DEMO 1", while the segway is balancing at the zero position, imposes the (filtered) reference signal on the selected controller ("MPC Pitch & Position" or "PID Pitch & Position").

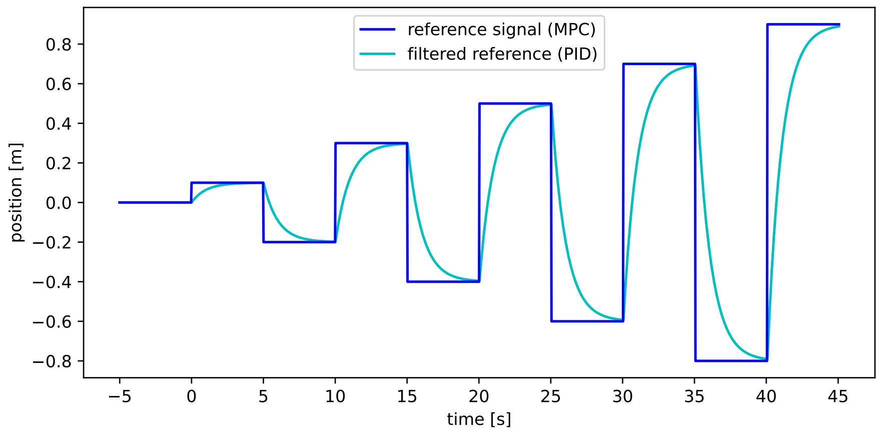


Figure 9: DEMO 1: Reference signal and filtered reference signal ($T = 1$) for times up to 45 seconds.

Exercise 10.1. *Comparing MPC and PID*
Apply the reference signal "DEMO 1" to the MPC Pitch & Position controller and assess its performance. Then switch to the PID Pitch & Position controller, enter the filter time constant found in exercise 7.8 and click "DEMO 1" again. Compare the performance of the PID controller on the filtered signal to the MPC on the raw signal. Also take note of the number of steps each controller can manage before failing because of the increased stepsize.

11 Demonstration: Arbitrary Planar Trajectories

In this final chapter, we generalize to arbitrary planar trajectories instead of just linear motion. This is achieved by imposing references on both, the position and the angle. As described in section 8, a proportional yaw angle controller is running on top of the MPC/PID Pitch & Position algorithm. This setup enables the separation of positional and angular control. Thus the problem is decoupled, and two separate reference signals for position and angle are used to define a two dimensional trajectory. Since any planar curve can be parametrized with position and angle, any planar curve can be imposed as reference trajectory.

Exercise 11.1. *Arbitrary Planar Trajectory*
Figure 10 shows the (filtered) positional reference signal of the planar trajectory "DEMO 2". Additionally, the angular reference steps are indicated by the green (-90°) and red (+90°) circles. Can you figure out which geometric shape the planar trajectory corresponds to?

Exercise 11.2. *Arbitrary Planar Motion*
Apply the reference signal "DEMO 2" to the PID Pitch & Position controller and verify your guess on the shape of the planar trajectory. Use the filter time constant found in exercise 7.8. Then switch to the MPC Pitch & Position controller and click "DEMO 2" again. Compare the performance of the PID controller on the filtered signal to the MPC on the raw signal.

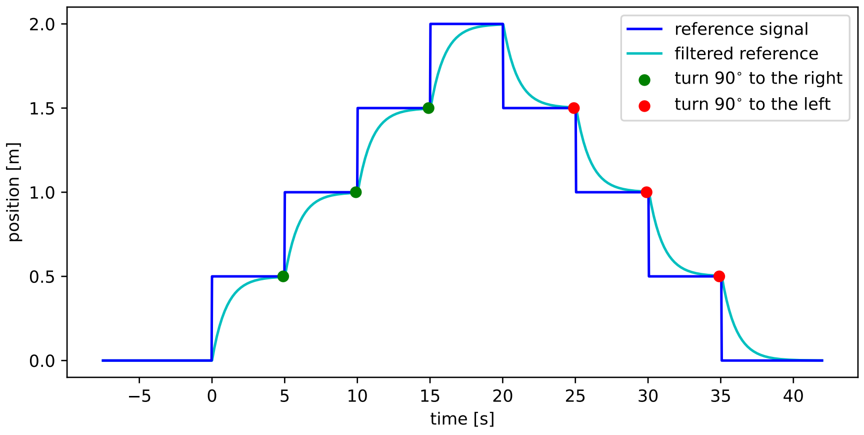


Figure 10: DEMO 2: Positional reference signal and filtered signal ($T = 1$). Additionally, the colored circles indicate the steps in the angular reference.

DEMO 3 has not been implemented yet. We are happy to receive suggestions by you.

References

[1] M. Zeilinger, "Lecture notes from model predictive control course," *ETH, Zurich*, 2020.

[2] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 01 2017.

[3] L. Guzzella, *Analysis and Synthesis of Single-input Single-output Control Systems*, ser. Vdf Vorlesungsskripte. vdf Hochschulvlg, 2011.

[4] S. Kim and S. Kwon, "On the dynamic model of a two-wheeled inverted pendulum robot," in *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE, 2014, pp. 145–148.

[5] K. A. Stol, "Modelling and stability control of two-wheeled robots in low-traction environments," 2010.