Technische Universität Berlin
Institute for Process an Chemical Engineering
Chair of Control Engineering

IESS – Laboratory 3

Control of a self-balancing robot

In this laboratory, we will explore the concept of self-balancing robots and how we can use Linear Quadratic Regulator (LQR) control and state feedback controllers to stabilize them. The difficulty of self-balancing robots is that balancing a robot on two wheels is a challenging task due to the inherent instability of the system. This is where LQR control and state feedback controllers come into play.

LQR control is a widely used control technique in control theory that aims to minimize the cost function while ensuring stability. State feedback controllers, on the other hand, use the full state of the system to determine the control input, making them more effective than traditional controllers that only use partial state information. By combining these two techniques, we can create a robust and stable control system for self-balancing robots.

In this laboratory, we will use our simulation setup from Laboratory 2 – Simulating the dynamic model of a self-balancing robot and the linearized model from Laboratory 1 – Modeling a self-balancing robot to design and implement control for a self-balancing robot. We will explore the theoretical background behind these techniques and demonstrate their effectiveness in stabilizing the system. By the end of this laboratory, you will have a solid understanding of how LQR control and state feedback controllers can be used in the field of robotics to create stable and robust control systems for self-balancing robots.

Linear Quadratic Regulator control

Linear Quadratic Regulator (LQR) control is a mathematical technique used to design a feedback control system that aims to minimize the cost function of a linear system.

In LQR control, the dynamics of the system are described by a set of linear equations. The objective of the controller is to determine the optimal control input that minimizes a quadratic cost function. The cost function is a measure of the system's performance and is defined as the sum of the weighted squared errors between the system's output and a desired reference trajectory, as well as the weighted squared input effort.

The LQR control algorithm calculates the optimal feedback gains for a linear state feedback controller by solving the Riccati equation. The state feedback controller then applies these gains to the current state of the system to generate the optimal control input.

LQR control is widely used in control theory for linear systems due to its simplicity and efficiency. It is also an optimal control strategy that can achieve the best possible performance for a given system, assuming that the system dynamics are accurately modeled and the initial conditions are known.

Consider a linear time-invariant system described by the following state-space equa-

tions:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1}$$
$$y(t) = Cx(t) + Du(t)\,, \tag{2}$$

where $x(t)$ is the system state, $u(t)$ is the control input, $y(t)$ is the output, and A, B, C, and D are matrices that describe the system's dynamics.

The goal of LQR control is to design a state feedback controller that minimizes a quadratic cost function of the form:

$$J(u) = \int_0^\infty x^T(t)Qx(t) + u^T(t)Ru(t)\mathrm{d}t\,, \tag{3}$$

where Q and R are positive semi-definite weighting matrices that determine the relative importance of the state and input terms in the cost function.

The optimal control input $u^*(t)$ that minimizes the cost function can be computed using the algebraic Riccati equation:

$$0 = A^TP + PA - PB^TR^{-1}BP + Q \tag{4}$$
$$K = R^{-1}B^TP \tag{5}$$
$$u^*(t) = -Kx(t) \tag{6}$$

where P is the solution of the algebraic Riccati equation in (4) and K is the optimal feedback gain matrix. Once the optimal feedback gains K are obtained, the state feedback controller can be implemented as:

$$u(t) = -Kx(t) \tag{7}$$

This controller uses the current state of the system $x(t)$ to compute the optimal control input $u(t)$ that minimizes the cost function. The resulting closed-loop system is stable and optimal, provided that the system is controllable and observable.

LQR control is a powerful and widely used control strategy for linear systems, and it has many practical applications in control engineering and robotics. The choice of weighting matrices Q and R plays a crucial role in shaping the controller's behavior and performance. Here's a breakdown of the rationale behind selecting these matrices and how they influence the controller:

Weighting matrix Q is a positive semi-definite matrix that penalizes deviations from the desired state in the system. The choice of Q influences the relative importance of different state variables in the control objective. Higher weights on certain states indicate that deviations from the desired values of those states are less tolerable. The selection of Q can be guided by the system's dynamics, where states that are more critical to performance or stability receive higher weights.

Weighting matrix R is a positive definite matrix that penalizes control effort or the magnitude of control inputs. It represents the cost associated with applying control actions and is used to balance control effort against state deviations. Higher values in RR correspond to more aggressive control efforts. The choice of RR can be influenced by factors such as actuator limitations, desired control smoothness, or cost considerations related to energy consumption.

Impact on performance *Stability*: Properly chosen Q and R matrices ensure stability of the closed-loop system. If the weights are too high, it may lead to instability or excessive control effort. *Response Time*: Balancing the weights in Q and R affects the system's response time. Higher weights in Q can lead to faster response but might increase control effort. Similarly, higher weights in R can lead to an increased response time. *Control Effort*: The choice of R directly impacts control effort. Lower values result in more aggressive control actions, potentially leading to faster response but also possibly causing wear and tear on actuators.

Trade-offs There's often a trade-off between fast response and minimal control effort. High values of Q and R can lead to faster responses but might demand more aggressive control, which could be undesirable or even impractical in some cases. Balancing these trade-offs is essential to design an LQR controller that achieves desired performance while respecting system constraints and limitations.

In summary, the choice of Q and R matrices in an LQR controller is a critical design decision that involves balancing performance objectives, system dynamics, and control constraints. Properly selecting these matrices can lead to stable, efficient, and responsive control systems tailored to specific application requirements.

Task 1 – Choosing the appropriate controller gain

1. Compute the poles and zeros of the linearized model derived in Laboratory 1 – Modeling a self-balancing robot using your system parameters from Laboratory 2 – Simulating the dynamic model of a self-balancing robot. Verify if the linearized model is stable or not.

2. Choose the appropriate $Q \in \mathbb{R}^{4\times4}$ and $R \in \mathbb{R}^{1\times1}$ matrices for the `lqr()` function in MATLAB to find the optimal feedback gain K.

3. Add your controller to your Simulink model from Laboratory 2. The structure of your model should be similar to Figure 1.

4. Verify that controller K stabilizes the model by running simulations.

5. Run a few simulations with different initial positions and estimate the save-angle of your system.

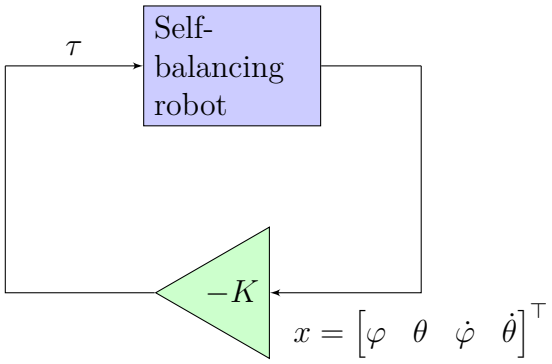

Figure 1: State-feedback regulation block diagram.

Task 2 – Motor control allocation

1. Assume that the self-balancing robot has a single-driven wheel, express the motor current i_a as a function of the demanded torque for $\tau > 0$ and $\tau < 0$.

2. Express the required motor voltage E_a as a function of the motor current i_a and the angular velocity of the wheel $\dot{\varphi}$.

3. Make a MATLAB function for the control allocation that takes the required torque from the controller $\hat{\tau}$ and the angular velocity of the wheel $\dot{\varphi}$ and returns the required motor voltage E_a.

4. Add the DC motor/gearbox model and the control allocation function to your Simulink model from Task 1. The structure of your model should look similar to Figure 2.

5. Run the Simulink model and verify that the system is still stable.

6. Run a few simulations and estimate the save-angle of your model.

Task 3 – Adding a reference signal

1. Add a reference signal to your Simulink model so that the controller acts on the error as shown in Figure 3.

2. Change the reference input and observe the behavior of the system.

Extension – Integral action

Integral action is a common control strategy used in self-balancing robots to improve their stability and reduce errors in their balance control system.

In a self-balancing robot, the goal is to maintain a stable upright position, despite disturbances and external forces acting on the robot. The balance control system measures the robot's orientation, calculates the required corrective action, and applies it to the robot's motors to keep it balanced. However, even with the best sensors and control algorithms, there can still be small errors in the system that can accumulate over time and cause the robot to become unstable.

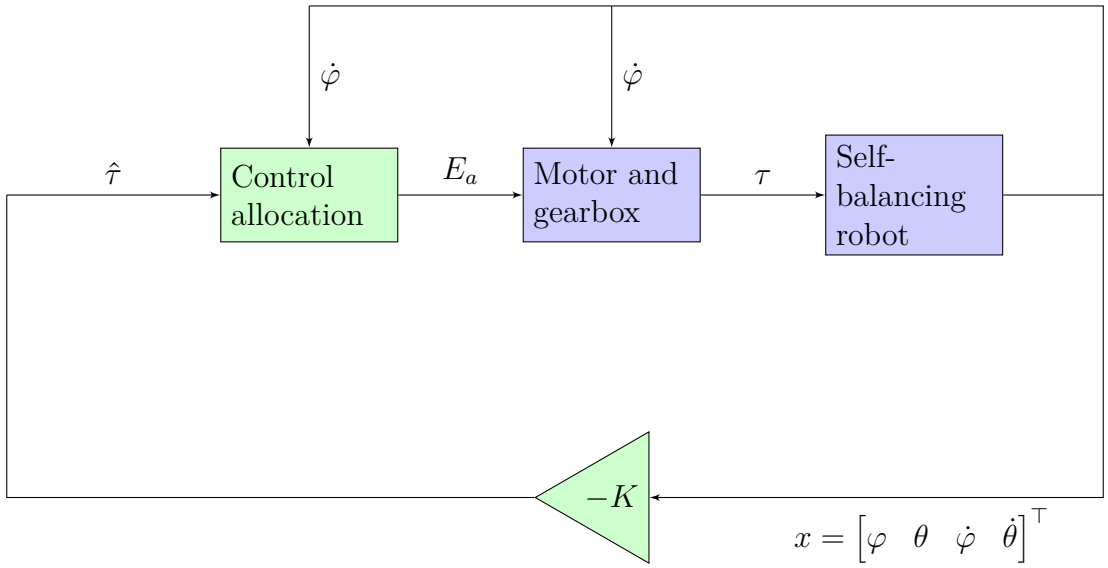

Figure 2: Control allocation for the DC motor with a gearbox connected to the self-balancing robot. The green blocks are the control system and the purple blocks are the model.

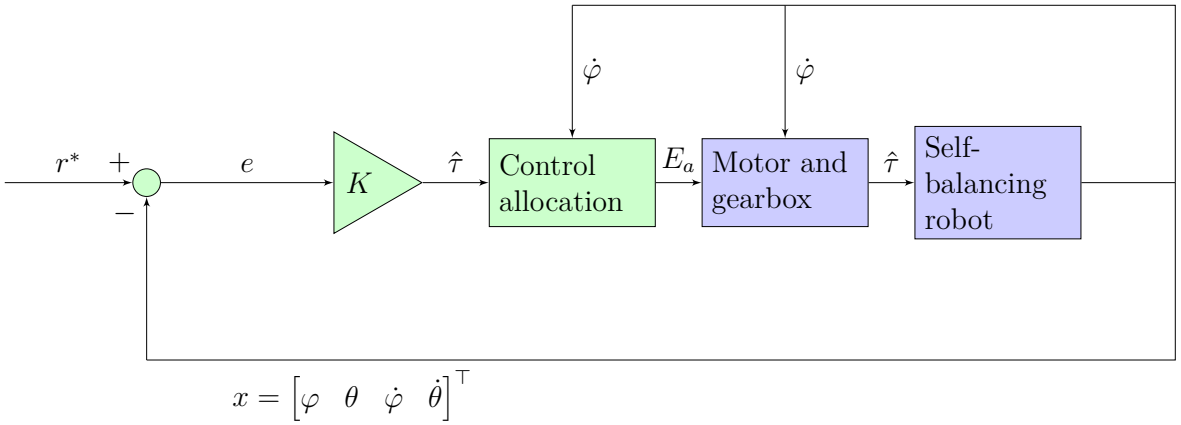

Figure 3: State-feedback with reference block diagram.

This is where integral action comes in. Integral action is a type of feedback control that uses the accumulated error over time to adjust the control output. In the case of a self-balancing robot, the integral action term integrates the error signal over time and adds it to the control signal, allowing the robot to correct for any persistent errors and maintain a stable upright position.

To augment a linear control system with integral action using LQR control, we can modify the cost function of the LQR controller to include an integral term. The new cost function becomes:

$$J(u) = \int_0^\infty \left(x^\top Q x + u^\top R u \right) \mathrm{d}t + K_z \int_0^\infty e^2(t) \mathrm{d}t \,, \tag{8}$$

where x is the state vector, u is the control input, $e(t)$ is the error signal, and Q and R are weighting matrices that determine the trade-off between state and control effort. The last term in the cost function is the integral of the squared error signal multiplied by the gain for the integral action K_I.

For the system in (1)–(2), we have that assuming $D = 0$, then the augmented system dynamics can be written as

$$\dot{x}_a = A_a x_a + B_a u \,, \tag{9}$$

where $x_a = [x^\top \;\; e]^\top$ is the augmented state vector and

$$A_a = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \,, \quad B_a = \begin{bmatrix} B \\ 0 \end{bmatrix} \,. \tag{10}$$

The LQR controller then solves for the optimal control law that minimizes the cost function. The resulting feedback control law is given by:

$$u = -K x_a \tag{11}$$

where K is the optimal feedback gain matrix computed using the LQR algorithm. The first 4 elements of K correspond to the feedback gain for the original system, while the last element corresponds to the gain for the integral action K_z.

The optimal feedback gain matrix K can be computed by solving the algebraic Riccati equation as described in section Linear Quadratic Regulator control.

The resulting closed-loop system will have integral action that helps to eliminate steady-state errors in the system.

Figure 4 shows a block diagram of the self-balancing robot with integral action.

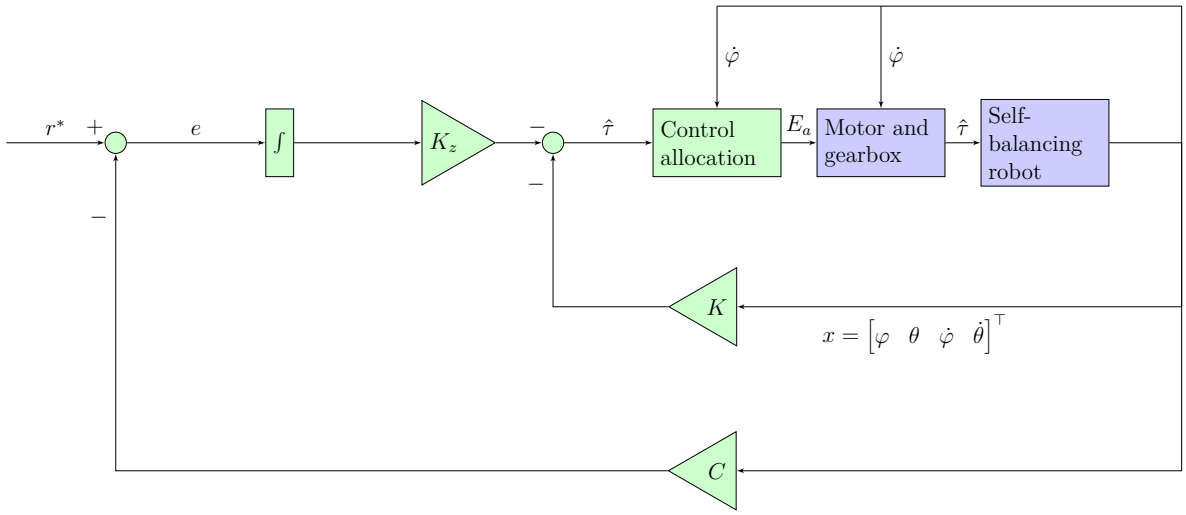

Figure 4: LQR control with integral action.

Extension 1 – Using integral action for a θ reference

1. Define the appropriate C for a reference of the angle of the chassis θ.
2. Form the augmented system like that in (9).
3. Execute the steps in Task 1 again for the new, augmented system and adjust your SIMULINK model to include integral action like the block diagram in Figure 4.
4. Change the signal for the reference input θ and observe the difference in the system's behavior.

Extension 2 – Using integral action for a linear velocity reference

1. Define the appropriate C for a reference of the linear velocity of the robot $\dot{x}$.
2. Perform the steps in Extension 1 for the new, augmented system.

Acknowledgments
This laboratory was developed by Roxanne R. Jackson, Magnus Axelson-Fisk, and Tom J. Jüstel. ChatGPT (AI-generated text) was used to assist in writing the content of this laboratory.